

# **Intel® Integrated Performance Primitives**

**Developer Reference, Volume 2: Image Processing**

[Notices and Disclaimers](#)

# Contents

<b>Notices and Disclaimers.....</b>	<b>16</b>
<b>Chapter 1: Overview</b>	
What's New .....	17
Notational Conventions .....	17
<b>Chapter 2: Intel(R) Integrated Performance Primitives Concepts</b>	
Function Naming .....	19
Data-Domain .....	19
Name .....	19
Data Types .....	19
Descriptors.....	21
Parameters.....	22
Extensions .....	22
Function Prototypes in Intel IPP.....	22
Rounding Mode .....	23
Integer Result Scaling .....	23
Error Reporting .....	24
Platform-Aware Functions for Image Processing .....	27
Threading Layer Functions .....	27
Structures and Enumerators .....	28
Function Context Structures .....	34
Structures and Enumerators for Platform-Aware Functions .....	34
Image Data Types and Ranges .....	36
Major Operation Models.....	37
Neighborhood Operations .....	37
Regions of Interest in Intel IPP.....	37
Tiled Image Processing .....	40
<b>Chapter 3: Support Functions</b>	
Version Information Function .....	41
GetLibVersion .....	41
Status Information Function.....	42
ippGetStatusString .....	42
Memory Allocation Functions .....	43
Malloc.....	43
Free .....	44
Threading Layer Functions .....	45
SplitUniform2D .....	45
ParallelFor .....	46
GetTilePointer .....	46
GetTileParamsByIndex .....	47
GetThreadingType .....	48
GetThreadIdx .....	49
<b>Chapter 4: Image Data Exchange and Initialization Functions</b>	
Convert.....	50
BinToGray, GrayToBin.....	54
Scale .....	56

ScaleC .....	59
Set .....	61
Copy .....	63
CopyManaged .....	67
CopyConstBorder .....	68
CopyMirrorBorder .....	73
CopyReplicateBorder .....	77
CopyWrapBorder .....	80
CopySubpix .....	83
CopySubpixIntersect .....	84
Dup .....	87
Transpose .....	88
SwapChannels .....	90
AddRandUniform .....	93
AddRandGauss .....	94
ImageJaehne .....	95
ImageRamp .....	97
SampleLine .....	99
ZigzagFwd8x8 .....	100
ZigzagInv8x8 .....	102

## Chapter 5: Image Arithmetic and Logical Operations

Arithmetic Operations .....	104
Add .....	104
AddC .....	110
AddSquare .....	117
AddProduct .....	118
AddWeighted .....	119
Mul .....	121
MulC .....	125
MulC64f .....	132
MulScale .....	135
MulCScale .....	136
Sub .....	139
SubC .....	143
Div .....	149
Div_Round .....	154
DivC .....	156
Abs .....	163
AbsDiff .....	164
AbsDiffC .....	165
Sqr .....	166
Sqrt .....	168
Ln .....	170
Exp .....	172
DotProd .....	174
DotProdCol .....	176
Complement .....	177
Logical Operations .....	178
And .....	178
AndC .....	180
Or .....	182
OrC .....	183
Xor .....	185
XorC .....	187

Not .....	189
RShiftC .....	190
LShiftC .....	192
Alpha Composition.....	195
AlphaComp .....	196
AlphaCompC.....	198
AlphaPremul.....	201
AlphaPremulC.....	202

## Chapter 6: Image Color Conversion

Gamma Correction .....	205
CIE Chromaticity Diagram and Color Gamut .....	206
Color Models.....	206
Image Downsampling .....	217
RGB Image Formats.....	218
Pixel and Planar Image Formats .....	218
Color Model Conversion .....	222
RGBToYUV.....	222
YUVToRGB.....	224
RGBToYUV422 .....	225
YUV422ToRGB .....	226
RGBToYUV420 .....	228
YUV420ToRGB .....	229
BGRToYUV420 .....	231
YUV420ToBGR .....	232
YUV422v210ToRGB, YUV422v210ToBGR .....	233
YUV422v210ToGray .....	234
RGBToYCbCr .....	234
YCbCrToRGB.....	236
YCbCrToBGR.....	237
YCbCrToBGR_709CSC .....	238
RGBToYCbCr422.....	239
YCbCr422ToRGB.....	240
RGBToYCrCb422.....	242
YCrCb422ToRGB, YCrCb422ToBGR .....	243
BGRToYCbCr422.....	244
YCbCr422ToBGR.....	245
YCbCr422ToGray .....	247
RGBToCbYCr422, RGBToCbYCr422Gamma .....	248
CbYCr422ToRGB.....	249
BGRToCbYCr422.....	250
BGRToCbYCr422_709HDTV .....	251
CbYCr422ToBGR.....	252
CbYCr422ToBGR_709HDTV .....	253
RGBToYCbCr420.....	254
YCbCr420ToRGB, YCbCr420ToBGR .....	255
RGBToYCrCb420.....	257
YCrCb420ToRGB, YCrCb420ToBGR .....	258
BGRToYCbCr420.....	260
BGRToYCbCr420_709CSC .....	261
BGRToYCbCr420_709HDTV .....	262
BGRToYCrCb420_709CSC .....	263
YCbCr420ToBGR.....	265
YCbCr420ToBGR_709CSC .....	266
YCbCr420ToBGR_709HDTV .....	267

BGRToYCrCb420.....	268
BGRToYCbCr411.....	269
YCbCr411ToBGR.....	270
RGBToXYZ.....	271
XYZToRGB.....	272
RGBToLUV, BGRToLUV.....	273
LUVToRGB, LUVToBGR .....	275
BGRToLab, RGBToLab.....	276
LabToBGR, LabToRGB .....	279
RGBToYCC.....	281
YCCToRGB.....	282
RGBToHLS.....	283
HLSToRGB.....	284
BGRToHLS.....	285
HLSToBGR.....	287
RGBToHSV .....	288
HSVToRGB .....	289
RGBToYCoCg .....	290
YCoCgToRGB .....	290
BGRToYCoCg .....	291
SBGRToYCoCg .....	292
YCoCgToBGR .....	293
YCoCgToSBGR .....	294
BGRToYCoCg_Rev .....	295
SBGRToYCoCg_Rev.....	296
YCoCgToBGR_Rev .....	297
YCoCgToSBGR_Rev .....	298
Color - Gray Scale Conversions.....	299
GrayToRGB.....	299
RGBToGray .....	301
ColorToGray .....	302
CFAToBGRA .....	303
CFAToRGB .....	305
DemosaicAHD.....	306
Format Conversion .....	308
YCbCr422.....	308
YCbCr422ToYCrCb422 .....	309
YCbCr422ToCbYCr422 .....	310
YCbCr422ToYCbCr420 .....	311
YCbCr422To420_Interlace .....	312
YCbCr422ToYCrCb420 .....	313
YCbCr422ToYCbCr411 .....	314
YCrCb422ToYCbCr422 .....	316
YCrCb422ToYCbCr420 .....	316
YCrCb422ToYCbCr411 .....	317
CbYCr422ToYCbCr422 .....	318
CbYCr422ToYCbCr420 .....	319
CbYCr422ToYCbCr420_Interlace .....	320
CbYCr422ToYCrCb420 .....	321
CbYCr422ToYCbCr411 .....	322
YCbCr420.....	323
YCbCr420ToYCbCr422 .....	324
YCbCr420ToYCbCr422_Filter .....	325
YCbCr420To422_Interlace .....	327
YCbCr420ToCbYCr422 .....	328

YCbCr420ToCbYCr422_Interlace .....	329
YCbCr420ToYCrCb420 .....	330
YCbCr420ToYCrCb420_Filter .....	331
YCbCr420ToYCbCr411 .....	333
YCrCb420ToYCbCr422 .....	334
YCrCb420ToYCbCr422_Filter .....	335
YCrCb420ToCbYCr422 .....	336
YCrCb420ToYCbCr420 .....	337
YCrCb420ToYCbCr411 .....	338
YCbCr411 .....	339
YCbCr411ToYCbCr422 .....	340
YCbCr411ToYCrCb422 .....	341
YCbCr411ToYCbCr420, YCbCr411To420 .....	342
YCbCr411ToYCrCb420 .....	343
Color Twist .....	344
ColorTwist .....	345
ColorTwist32f .....	346
Color Keying .....	348
CompColorKey .....	348
AlphaCompColorKey .....	349
Gamma Correction .....	351
GammaFwd .....	351
GammaInv .....	353
Intensity Transformation .....	355
ReduceBitsGetBufferSize .....	355
ReduceBits .....	356
LUT_GetSize .....	359
LUT_Init .....	360
LUT .....	362
LUTPalette, LUTPaletteSwap .....	367
ToneMapLinear, ToneMapMean .....	369
<b>Chapter 7: Threshold and Compare Operations</b>	
Thresholding .....	371
Threshold .....	371
Threshold_GT .....	374
Threshold_LT .....	375
Threshold_Val .....	377
Threshold_GTVal .....	379
Threshold_LTVal .....	381
Threshold_LTValGTVal .....	383
ComputeThreshold_Otsu .....	386
Compare Operations .....	388
Compare .....	388
CompareC .....	389
CompareEqualEps .....	391
CompareEqualEpsC .....	392
<b>Chapter 8: Morphological Operations</b>	
Dilate3x3 .....	397
Dilate .....	398
DilateBorder .....	400
DilateGetBufferSize .....	403
DilateGetSpecSize .....	404
DilateInit .....	405

Erode3x3 .....	406
Erode .....	407
ErodeBorder .....	409
ErodeGetBufferSize.....	411
ErodeGetSpecSize .....	411
ErodeInit.....	412
GrayDilateBorder.....	413
GrayErodeBorder.....	414
MorphAdvInit.....	415
MorphAdvGetSize .....	417
MorphGetBufferSize .....	418
MorphGetSpecSize.....	418
MorphInit .....	419
MorphologyBorderGetSize.....	420
MorphologyBorderInit .....	421
MorphBlackhat .....	422
MorphBlackhatBorder .....	425
MorphClose .....	427
MorphCloseBorder .....	429
MorphGradient .....	432
MorphGradientBorder .....	435
MorphOpen.....	437
MorphOpenBorder .....	439
MorphTophat.....	441
MorphTophatBorder .....	444
MorphGrayInit.....	446
MorphGrayGetSize.....	447
MorphReconstructGetBufferSize.....	448
MorphReconstructDilate.....	449
MorphReconstructErode.....	451
MorphSetMode .....	452

## Chapter 9: Filtering Functions

Borders in Neighborhood Operations .....	454
User-defined Border Types.....	455
Filters with Borders.....	460
FilterBilateral .....	460
FilterBilateralGetBufferSize .....	464
FilterBilateralInit .....	465
FilterBilateralBorderGetBufferSize.....	468
FilterBilateralBorderInit .....	469
FilterBilateralBorder.....	472
FilterBoxBorderGetBufferSize .....	474
FilterBoxBorder .....	475
FilterBox .....	477
FilterGaussianBorder .....	478
SumWindow .....	479
SumWindowGetBufferSize .....	482
SumWindowRow .....	483
SumWindowColumn.....	484
FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize .....	485
FilterMaxBorder, FilterMinBorder .....	486
DecimateFilterRow, DecimateFilterColumn .....	490
Median Filters .....	491
FilterMedianBorderGetBufferSize .....	492

FilterMedianBorder .....	493
FilterMedianGetBufferSize .....	495
FilterMedian .....	496
FilterMedianCross .....	497
FilterMedianWeightedCenter3x3 .....	499
FilterMedianColor .....	500
General Linear Filters .....	501
FilterBorderGetSize .....	501
FilterBorderInit .....	502
FilterBorder .....	504
FilterBorderSetMode .....	506
FilterGetBufSize .....	507
Filter .....	508
Separable Filters .....	509
FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low .....	510
FilterRowBorderPipeline, FilterRowBorderPipeline_Low .....	511
FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low .....	513
FilterColumnPipeline, FilterColumnPipeline_Low .....	514
FilterSeparable .....	516
FilterSeparableGetBufferSize .....	519
FilterSeparableGetSpecSize .....	520
FilterSeparableInit .....	521
Smoothing Filters .....	523
FilterILSInit .....	523
FilterILSGetBufferSize .....	524
FilterILS .....	525
Wiener Filters .....	526
FilterWienerGetBufferSize .....	526
FilterWiener .....	527
Convolution .....	529
ConvGetBufferSize .....	530
Conv .....	531
Deconvolution .....	534
DeconvFFTGetSize .....	534
DeconvFFTInit .....	535
DeconvFFT .....	536
DeconvLRLGetSize .....	537
DeconvLRInit .....	538
DeconvLR .....	539
Fixed Filters .....	540
FilterGaussianGetBufferSize .....	541
FilterGaussianGetSpecSize .....	542
FilterGaussianInit .....	543
FilterGaussian .....	545
FilterHipassBorderGetBufferSize .....	547
FilterHipassBorder .....	548
FilterLaplaceBorderGetBufferSize .....	551
FilterLaplaceBorder .....	552
FilterLaplacianGetBufferSize .....	554
FilterLaplacianBorder .....	555
FilterLowpassGetBufferSize .....	557
FilterLowpassBorder .....	557
FilterPrewittHorizBorderGetBufferSize .....	559

FilterPrewittHorizBorder .....	560
FilterPrewittVertBorderGetBufferSize .....	562
FilterPrewittVertBorder .....	563
FilterRobertsUpBorderGetBufferSize .....	565
FilterRobertsUpBorder .....	566
FilterRobertsDownBorderGetBufferSize .....	568
FilterRobertsDownBorder .....	569
FilterScharrHorizMaskBorderGetBufferSize .....	571
FilterScharrHorizMaskBorder .....	572
FilterScharrVertMaskBorderGetBufferSize .....	574
FilterScharrVertMaskBorder .....	575
FilterSharpenBorderGetBufferSize .....	578
FilterSharpenBorder .....	579
FilterSobelGetBufferSize .....	581
FilterSobelInit .....	583
FilterSobel .....	584
FilterSobelHorizBorderGetBufferSize .....	587
FilterSobelHorizBorder .....	588
FilterSobelHorizSecondBorderGetBufferSize .....	591
FilterSobelHorizSecondBorder .....	592
FilterSobelVertBorderGetBufferSize .....	594
FilterSobelVertBorder .....	595
FilterSobelVertSecondBorderGetBufferSize .....	598
FilterSobelNegVertBorderGetBufferSize .....	599
FilterSobelNegVertBorder .....	600
FilterSobelVertSecondBorder .....	602
FilterSobelCrossGetBufferSize .....	603
FilterSobelCrossBorder .....	604
GenSobelKernel .....	606
Deinterlacing Filters .....	607
DeinterlaceFilterCAVT .....	607
Median .....	608

## Chapter 10: Image Linear Transforms

Fourier Transforms .....	609
Real - Complex Packed (RCPack2D) Format .....	609
FFTGetSize .....	610
FFTInit .....	611
FFT_fwd .....	613
FFT_Inv .....	616
DFTGetSize .....	618
DFTInit .....	619
DFTFwd .....	621
DFTInv .....	623
MulPack .....	624
MulPackConj .....	626
Magnitude .....	628
MagnitudePackGetBufferSize .....	629
MagnitudePack .....	630
Phase .....	631
PhasePackGetBufferSize .....	632
PhasePack .....	632
PolarToCart .....	634
PackToCplxExtend .....	635
CplxExtendToPack .....	636

Windowing Functions .....	637
WinBartlettGetBufferSize, WinBartlettSepGetBufferSize, .....	637
WinBartlett, WinBartlettSep .....	638
WinHammingGetBufferSize, WinHammingSepGetBufferSize, .....	640
WinHamming, WinHammingSep .....	641
Discrete Cosine Transforms.....	643
DCTFwdGetSize, DCTInvGetSize .....	643
DCTFwdInit, DCTInvInit .....	644
DCTFwd .....	645
DCTInv .....	648
DCT8x8Fwd .....	649
DCT8x8Inv, DCT8x8Inv_A10 .....	651
DCT8x8FwdLS .....	652
DCT8x8InvLSSClip .....	653
DCT8x8Inv_2x2, DCT8x8Inv_4x4 .....	654
DCT8x8To2x2Inv, DCT8x8To4x4Inv .....	655

## Chapter 11: Image Statistics Functions

Sum .....	657
Integral.....	659
SqrIntegral .....	661
TiltedIntegral .....	663
TiltedSqrIntegral .....	664
Mean .....	666
Mean_StdDev .....	668
RectStdDev .....	670
TiltedRectStdDev .....	671
HistogramGetBufferSize .....	673
HistogramGetLevels .....	674
HistogramInit, HistogramUniformInit .....	675
Histogram .....	676
CountInRange .....	679
BlockMinMax .....	680
Min .....	682
MinIndx .....	684
Max .....	685
MaxIndx .....	686
MinMax .....	688
MinMaxIndx .....	689
MaxEvery .....	691
MinEvery .....	693
FindPeaks3x3GetBufferSize .....	694
FindPeaks3x3 .....	695
Image Moments .....	697
MomentGetStateSize .....	698
MomentInit .....	698
Moments .....	699
GetSpatialMoment .....	700
GetCentralMoment .....	701
GetNormalizedSpatialMoment .....	702
GetNormalizedCentralMoment .....	703
GetHuMoments .....	704
Image Norms .....	704
Norm_Inf .....	705
Norm_L1 .....	707

Norm_L2.....	709
NormDiff_Inf .....	711
NormDiff_L1 .....	714
NormDiff_L2 .....	717
NormRel_Inf .....	719
NormRel_L1 .....	721
NormRel_L2 .....	724
Image Quality Index .....	727
QualityIndexGetBufferSize.....	728
QualityIndex.....	729
Image Proximity Measures.....	731
SqrDistanceNormGetBufferSize .....	732
SqrDistanceNorm .....	733
CrossCorrNormGetBufferSize .....	736
CrossCorrNorm .....	737
SADGetBufferSize .....	740
SAD .....	741

## Chapter 12: Image Geometry Transforms

ROI Processing in Geometric Transforms.....	746
Geometric Transform Functions .....	747
ResizeYCbCr422GetBufSize.....	747
ResizeYCbCr422 .....	748
Resize Functions with Prior Initialization .....	749
Using Intel® IPP Resize Functions with Prior Initialization .....	749
ResizeGetSize.....	758
ResizeGetBufferSize .....	760
ResizeGetBorderSize .....	762
ResizeGetSrcOffset.....	763
ResizeGetSrcRoi.....	764
ResizeSetMode .....	765
ResizeNearestInit.....	766
ResizeNearest.....	767
ResizeLinearInit .....	769
ResizeLinear.....	771
ResizeCubicInit .....	775
ResizeCubic.....	776
ResizeLanczosInit.....	779
ResizeLanczos .....	781
ResizeSuperShiftInit .....	784
ResizeSuperInit .....	786
ResizeSuper .....	787
ResizeAntialiasingLinearInit .....	790
ResizeAntialiasingCubicInit .....	791
ResizeAntialiasingLanczosInit.....	792
ResizeAntialiasing .....	794
ResizeFilterGetSize.....	797
ResizeFilterInit .....	798
ResizeFilter .....	799
ResizeYUV420GetSize .....	800
ResizeYUV420GetSrcRoi .....	801
ResizeYUV420LanczosInit.....	802
ResizeYUV420SuperInit .....	803
ResizeYUV420GetBorderSize.....	804
ResizeYUV420GetSrcOffset.....	805

ResizeYUV420GetBufferSize .....	806
ResizeYUV420Lanczos .....	807
ResizeYUV420Super .....	809
ResizeYUV422GetSize .....	811
ResizeYUV422GetBorderSize .....	812
ResizeYUV422GetSrcOffset .....	813
ResizeYUV422GetBufSize .....	813
ResizeYUV422GetSrcRoi .....	814
ResizeYUV422NearestInit .....	815
ResizeYUV422LinearInit .....	816
ResizeYUV422Nearest .....	817
ResizeYUV422Linear .....	818
Warp Functions with Prior Initialization .....	820
Using Intel® IPP Warp Affine Functions with Prior Initialization .....	820
Edge Smoothing .....	826
GetAffineQuad .....	827
GetAffineBound .....	828
GetAffineSrcRoi .....	828
GetAffineTransform .....	830
GetRotateTransform .....	831
GetRotateShift .....	832
WarpAffineGetSize .....	832
WarpQuadGetSize .....	834
WarpGetBufferSize .....	836
WarpAffineNearestInit .....	837
WarpQuadNearestInit .....	839
WarpAffineNearest .....	841
WarpAffineLinearInit .....	843
WarpQuadLinearInit .....	845
WarpAffineLinear .....	847
WarpAffineCubicInit .....	850
WarpQuadCubicInit .....	852
WarpAffineCubic .....	854
GetPerspectiveQuad .....	856
GetPerspectiveBound .....	857
GetPerspectiveTransform .....	858
WarpGetRectInfinite .....	859
WarpPerspectiveGetSize .....	860
WarpPerspectiveNearestInit .....	862
WarpPerspectiveNearest .....	864
WarpPerspectiveLinearInit .....	866
WarpPerspectiveLinear .....	869
WarpPerspectiveCubicInit .....	871
WarpPerspectiveCubic .....	873
GetBilinearQuad .....	876
GetBilinearBound .....	877
GetBilinearTransform .....	878
WarpBilinearGetBufferSize .....	879
WarpBilinear .....	880
WarpBilinearBack .....	883
WarpBilinearQuadGetBufferSize .....	885
WarpBilinearQuad .....	886
Mirror .....	888
Remap .....	890

**Chapter 13: Miscellaneous Image Transforms**

ThresholdAdaptiveBoxGetBufferSize.....	895
ThresholdAdaptiveBox.....	896
ThresholdAdaptiveGaussGetBufferSize.....	898
ThresholdAdaptiveGaussInit.....	900
ThresholdAdaptiveGauss .....	901

**Chapter 14: Wavelet Transforms**

WTFwdGetSize .....	907
WTFwdInit.....	908
WTFwd .....	910
WTFwdGetSize .....	914
WTInvInit.....	915
WTInv.....	916

**Chapter 15: Computer Vision**

Using ippiAdd for Background Differencing .....	923
Feature Detection Functions.....	923
Corner Detection.....	924
FastNGetSize .....	924
FastNGetBufferSize.....	926
FastNInit .....	927
FastN.....	928
FastN2DToVec.....	932
HarrisCornerGetBufferSize.....	933
HarrisCorner.....	934
Canny Edge Detector .....	938
CannyBorderGetSize .....	939
CannyBorder .....	940
CannyGetSize.....	941
Canny .....	942
EigenValsVecsGetBufferSize.....	944
EigenValsVecsBorder .....	944
EigenValsVecs.....	947
MinEigenValGetBufferSize .....	949
MinEigenValBorder .....	950
MinEigenVal .....	952
Histogram of Oriented Gradients (HOG) Descriptor .....	953
HOGGetSize .....	955
HOGInit.....	955
HOGGetBufferSize.....	956
HOGGetDescriptorSize.....	957
HOG.....	958
Hough Transform .....	960
HoughLineGetSize.....	960
HoughLine .....	961
HoughLine_Region .....	962
HoughProbLineGetSize.....	963
HoughProbLineInit .....	964
HoughProbLine .....	965
LineSuppressionGetBufferSize.....	966
LineSuppression.....	967
Distance Transform Functions.....	970
DistanceTransform.....	970

GetDistanceTransformMask .....	973
FastMarchingGetBufferSize .....	974
FastMarching .....	975
TrueDistanceTransformGetBufSize.....	977
TrueDistanceTransform.....	977
Image Gradients .....	978
GradientColorToGray.....	979
GradientVectorGetBufferSize.....	980
GradientVectorPrewitt .....	981
GradientVectorScharr.....	984
GradientVectorSobel .....	988
Flood Fill Functions .....	991
FloodFillGetSize .....	992
FloodFillGetSize_Grad .....	993
FloodFill .....	993
FloodFill_Grad.....	995
FloodFill_Range.....	997
Motion Analysis and Object Tracking.....	999
FGMMGetBufferSize .....	999
FGMMInit .....	1000
FGMMForeground .....	1001
FGMMBackground .....	1002
Motion Template Functions .....	1003
Motion Representation .....	1003
Updating MHI Images .....	1004
UpdateMotionHistory .....	1004
Optical Flow .....	1005
OpticalFlowPyrLKGetSize.....	1005
OpticalFlowPyrLKInit .....	1006
OpticalFlowPyrLK .....	1007
Universal Pyramids .....	1009
PyramidGetSize .....	1009
PyramidInit .....	1010
GetPyramidDownROI .....	1011
GetPyramidUpROI .....	1012
PyramidLayerDownGetSize .....	1014
PyramidLayerDownInit .....	1015
PyramidLayerDown.....	1016
PyramidLayerUpGetSize .....	1018
PyramidLayerUpInit .....	1019
PyramidLayerUp.....	1021
Example of Using General Pyramid Functions .....	1022
Image Inpainting.....	1022
InpaintGetSize.....	1022
InpaintInit.....	1023
Inpaint .....	1025
Image Segmentation .....	1027
LabelMarkersGetBufferSize .....	1027
LabelMarkers .....	1028
MarkSpecklesGetBufferSize .....	1029
MarkSpeckles .....	1030
SegmentWatershedGetBufferSize .....	1032
SegmentWatershed .....	1033
SegmentGradientGetBufferSize .....	1036
SegmentGradient .....	1036

BoundSegments.....	1038
Pattern Recognition .....	1039
Object Detection Using Haar-like Features .....	1039
HaarClassifierGetSize .....	1040
HaarClassifierInit .....	1041
GetHaarClassifierSize .....	1043
TiltedHaarClassifierInit.....	1043
TiltHaarFeatures .....	1045
ApplyHaarClassifier .....	1046
ApplyMixedHaarClassifier .....	1047
Local Binary Pattern (LBP) Operator.....	1049
LBPIImageMode.....	1052
LBPIImageHorizCorr .....	1054
Camera Calibration and 3D Reconstruction.....	1055
Correction of Camera Lens Distortion .....	1055
UndistortGetSize.....	1056
UndistortRadial .....	1056
CreateMapCameraUndistort.....	1057

## **Chapter 16: 3D Data Processing Functions**

CopyConstBorder .....	1060
CopyReplicateBorder.....	1062
Filter .....	1065
FilterGetBufSize .....	1066
FilterBorder .....	1067
FilterBorderInit .....	1070
FilterBorderGetSize.....	1072
FilterMedian.....	1073
FilterMedianInit.....	1075
FilterMedianGetSize .....	1076
ResizeGetBufSize.....	1077
GetResizeCuboid .....	1078
Resize.....	1079
Remap .....	1082
WarpAffineGetBufSize .....	1084
WarpAffine .....	1085

## **Appendix A: Handling of Special Cases**

## **Appendix B: Interpolation in Image Geometric Transform Functions**

Overview of Interpolation Modes .....	1090
Mathematical Notation .....	1091
Nearest Neighbor Interpolation.....	1091
Linear Interpolation .....	1092
Cubic Interpolation .....	1092
Super Sampling .....	1093
Lanczos Interpolation .....	1094
Interpolation with Two-Parameter Cubic Filters .....	1095

## **Appendix C: Appendix C: Removed Functions for Image and Video Processing**

## **Appendix D: Bibliography for Image Processing**

## **Appendix E: Glossary**

# Notices and Disclaimers

---

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, AAC, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.167, G.168, G.169, G.723.1, G.726, G.728, G.729, G.729.1, GSM AMR, GSM FR are international standards promoted by ISO, IEC, ITU, ETSI, 3GPP and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

## Third Party

Intel® Integrated Performance Primitives (Intel® IPP) includes content from several 3rd party sources that was originally governed by the licenses referenced below:

- zlib library:

zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013

Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly Mark Adler

[jloup@gzip.org](mailto:jloup@gzip.org) [madler@alumni.caltech.edu](mailto:madler@alumni.caltech.edu)

- bzip2:

Copyright © 1996 - 2015 [julian@bzip.org](mailto:julian@bzip.org)

# Volume Overview

This manual describes the structure, operation, and functions of the Intel® Integrated Performance Primitives (Intel® IPP) that operate on two-dimensional signals and are used for image and video processing. The manual explains the Intel IPP concepts, as well as specific data type definitions and operation models used in the image and video processing domain, and provides detailed descriptions of the Intel IPP image and video processing functions. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate chapter (chapters 3 through 16).

For more information about image and video processing concepts and algorithms, refer to the books and materials listed in the [Bibliography](#).

## What's New

The document has been updated to reflect the following changes to the product:

- Added new threading layer flavors to the `Sqr`, `Sqrt`, and `Add` functions.
- Added new Complement function. See [Arithmetic Operations](#) for details.
- Added new flavors to the `ippiCrossCorrNorm` function. See [CrossCorrNorm](#) for details.

Additionally, minor updates have been made to fix inaccuracies in the document.

## Notational Conventions

The code and syntax used in this manual for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

### Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

This manual uses the following notational conventions:

Convention	Explanation	Example
<code>THIS_TYPE_STYLE</code>	Used in the text for the Intel IPP constant identifiers.	<code>IPPI_INTER_LINEAR</code>
<code>This type style</code>	Mixed with the uppercase in structure names; also used in function names, code examples and call statements.	<code>IppiSize,</code> <code>ippiMomentInitAlloc()</code>
<code>This type style</code>	Parameters in function prototypes and parameters description.	<code>value, srcStep</code>
<code>x(n)</code> and <code>x[n]</code>	Used to represent a discrete 1D signal. The notation <code>x(n)</code> refers to a conceptual signal, while the notation <code>x[n]</code> refers to an actual	<code>x[n], 0 ≤ n &lt; len</code>

Convention	Explanation	Example
	vector. Both of these are annotated to indicate a specific finite range of values.	Typically, the number of elements in vectors is denoted by <code>len</code> . Vector names contain square brackets as distinct from vector elements with current index <code>n</code> .  The expression <code>pDst[n] = pSrc[n] + val</code> implies that each element <code>pDst[n]</code> of the vector <code>pDst</code> is computed for each <code>n</code> in the range from 0 to <code>len-1</code> . Special cases are regarded and described separately.
<code>Ipp&lt;data-domain&gt;</code> and <code>Ipp</code> prefixes	All structures and enumerators, specific for the image and video processing domain have the <code>Ippi</code> prefix, while those common for entire Intel IPP software have the <code>Ipp</code> prefix.	<code>IppiPoint</code> , <code>IppDitherType</code>

## See Also

[Function Naming](#)

# Intel® Integrated Performance Primitives Concepts

2

This chapter explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the document.

## Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains.

Function names in Intel IPP have the following general format:

`ipp<data-domain><name>_<datatypedescriptor>] [<_extension>] (<parameters>);`

The elements of this format are explained in the sections that follow.

### Data-Domain

The *data-domain* is a single character that denotes the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

s	for signals (expected data type is a 1D signal)
i	for images and video (expected data type is a 2D image)
m	for matrices (expected data type is a matrix)
r	for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques)
g	for signals of fixed length

For example, function names that begin with `ippi` signify that respective functions are used for image or video processing.

### Name

The *name* field identifies what function does and has the following format:

`<name> = <operation>[_modifier]`

The *operation* component is one ore more words, acronyms, and abbreviations that describe the core operation.

The *modifier* component, if present, is a word or abbreviation that denotes a slight modification or variation of the given function.

For example, names without modifiers: `Add`, `RGBToYCbCr`, `MorphAddGetSize`; with modifiers: `DCT8x8Inv_2x2`, `DCT8x8Inv_4x4`, `RGBToYCbCr_JPEG`.

### Data Types

The *datatype* field indicates data types used by the function, in the following format:

<bit depth><bit interpretation> ,

where

bit depth = <1|8|16|32|64>

and

bit interpretation = <u|s|f>[c]

Here *u* indicates “unsigned integer”, *s* indicates “signed integer”, *f* indicates “floating point”, and *c* indicates “complex”.

Intel IPP supports the following data types for image and video processing functions:

8u	8 bit, unsigned data
8s	8 bit, signed data
16u	16 bit, unsigned data
16uc	16-bit, complex unsigned short data <sup>†</sup>
16s	16 bit, signed data
16sc	16-bit, complex short data
32u	32 bit, unsigned data
32s	32 bit, signed data
32sc	32-bit, complex int data
32f	32-bit, single-precision real floating point data
32fc	32-bit, single-precision complex floating point data
64s	64-bit, quadword signed data
64f	64-bit, double-precision real floating point data

<sup>†</sup> - only partial support for intermediate result after transforms (in the so-called “time” domain).

---

#### NOTE

For image processing functions that do not support 1u data type, convert bitonal images to 8u gray scale images using the `ippiConvert_1u8u_C1R` function.

---

The formats for complex data are represented in Intel IPP by structures defined as follows:

```
typedef struct { Ipp16s re; Ipp16s im; } Ipp16sc;
typedef struct { Ipp32s re; Ipp32s im; } Ipp32sc;
typedef struct { Ipp32f re; Ipp32f im; } Ipp32fc;
```

where *re*, *im* denote the real and imaginary parts, respectively.

Complex data formats are used by several arithmetic image processing functions. The 32fc format is also used to store input/output data in some Fourier transform functions.

The 64-bit formats, 64s and 64f, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the *datatype* field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

<datatype> = <src1Datatype>[src2Datatype] [dstDatatype]

For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the `8u32f` value for the `datatype` field.

#### **NOTE**

In the lists of function parameters (arguments), the `Ipp` prefix is written in the data type. For example, the 8-bit unsigned data is denoted as `Ipp8u` type. These Intel IPP-specific data types are defined in the respective library header files.

## Descriptors

The `descriptors` field further describes the operation. Descriptors are individual characters that indicate additional details of the operation.

The following descriptors are used in image and video processing functions:

Descriptor	Description
A	Image data contains an alpha channel as the last channel, requires <code>C4</code> , alpha-channel is not processed
A0	Image data contains an alpha channel as the first channel, requires <code>C4</code> , alpha-channel is not processed
C1, C2, C3, C4	Image data is in pixel order and made up of 1, 2, 3 or 4 discrete interleaved channels
C	Channel of interest (COI) is used in the operation
I	Operation is performed in-place - that is result of operation is written back into the source (default is not-in-place)
M	Operation uses a mask to determine pixels to be processed
P	Function works with non-contiguous volume data (the location of the planes is passed to the function using pointers on the plane)
P2, P3, P4	Image data is in planar order and made up of 2, 3 or 4 discrete planar (non-interleaved) channels, with a separate pointer to each plane
R	Function operates on a defined region of interest (ROI) for each source image
Sfs	Saturation and fixed scaling mode is used
s	Saturation and no scaling (default)
V	Function operates on a defined volume of interest (VOI) for each source image

If more than one descriptor is used, they are presented in the function name in alphabetical order.

Every function that operates on image data has a channel count descriptor `Cn` (for interleaved image) or `Pn` (for planar). No default channel count is defined.

If input and output channel layouts are different, both source and destination layouts are listed. For example, the function `ippiHLSToBGR_8u_C3P3R` converts three-channel interleaved HLS image to the three-plane BGR image.

## Parameters

The parameters in functions are in the following order: all source operands, all destination operands, all other operation-specific parameters.

Source parameters are named *Src* or *SrcN*, if there is more than one input image. Destination parameters are named *Dst*. For in-place operations, the input/output parameter contains the name *SrcDst*. All parameters defined as pointers start with lowercase *p*, for example, *pSrc*, *pMean*, *pSpec*.

## Extensions

The *extension* field denotes an Intel IPP extension to which the function belongs. The following extensions are supported in Intel IPP Image Processing functions:

Extension	Description	Example
L	Intel IPP platform-aware functions	ippiAddC_8u_AC4R_L
LT	Intel IPP threading layer (TL) functions based on the Platform Aware API	ippiSubC_8u_C1RSfs_LT
T	Intel IPP threading layer (TL) functions based on the Classic API	ippFilterMedian_64f_C1V_T

### See Also

[Platform-Aware Functions for Image Processing](#)

[Threading Layer Functions](#)

## Function Prototypes in Intel IPP

---

Function names in Intel IPP contain *datatype* and *descriptor* fields after the *name* field (see [Function Naming](#) section in this chapter). Most of the Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description sections of some chapters in this document, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as *mod*:

<mod> = <datatype>\_<datatype>

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:

```
IppStatusippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

where the supported values for *mod* are:

8u\_C1R  
8u\_C3R  
8u\_AC4R

This notation means that the `ippiDilate` function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of `Ipp8u` type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```
IppStatusippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatusippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatusippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Thus, to obtain the full name and parameters list for the specific function flavor, not listed directly, do the following:

1. Choose the function operation mode (denoted in this document as **Case 1**, **2** and so on) and look in the table for the supported data types and descriptors.
2. Set the `mod` field in the function name as the concatenation of the chosen data type and descriptor, delimited by the underscore.
3. Use the respective template, substituting all the `datatype` fields in the parameters list with the chosen data type. Note that `Ipp` prefix is written before the `datatype` in the parameters list (see [Data Types](#) in this chapter for details).

## Example

To get the prototype for the `ippiSet` function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use `datatype = 16s, descriptors = C3R`.

After substituting the `mod` field with `16s_C3R`, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16svalue[3], Ipp16* pDst, int dstStep, IppiSize roiSize);
```

## Rounding Mode

As many Intel IPP functions have to meet the bit-exact requirement, image processing functions use rounding. The default rounding mode for all functions can be described as "nearest even", that is the fixed point number  $x=N + \alpha$ ,  $0 \leq \alpha < 0$ , where  $N$  is an integer number, is rounded as given by:

$$\lceil x \rceil = \begin{cases} N, & 0 \leq \alpha < 0.5 \\ N+1, & 0.5 < \alpha < 1 \\ N, & \alpha = 0.5, N - \text{even} \\ N+1, & \alpha = 0.5, N - \text{odd} \end{cases}$$

For example, 1.5 is rounded to 2, and 2.5 to 2.

Some image processing functions have additional rounding modes, which are set by the parameter `roundMode`.

## Integer Result Scaling

The default for image processing functions is to saturate the results without scaling them.

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer `scaleFactor`, which is specified as one of the function parameters. These functions have the `Sfs` descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.

**NOTE**

The result of integer operations is always saturated to the destination data type range even when scaling is used.

---

The scaling of an integer result is done by multiplying the output pixel values by  $2^{-scaleFactor}$  before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest even integer number (see [Rounding Mode](#)).

For example, the integer `Ipp16s` result of the square operation `ippiSqr` for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value cannot be restored. The scaling of the output value with the factor `scaleFactor = 1` yields the result 20000, which is not saturated, and the exact value can be restored as  $20000 * 2^1 = 40000$ . Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling. The integer square root operation `ippiSqr` (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor `scaleFactor = -3` gives the result 11, and permits the more precise value to be restored as  $11 * 2^{-3} = 1.375$ .

## Error Reporting

---

The Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns. The status codes are of `IppStatus` type and are global constant integers.

The status codes and corresponding messages reported by Intel IPP for image and video processing are listed in Table .

### Status Codes and Messages

Status Code	Message
<code>ippStsNotSupportedModeErr</code>	The requested mode is currently not supported.
<code>ippStsDecimateFractionErr</code>	Unsupported fraction in decimate.
<code>ippStsWeightErr</code>	Incorrect value for weight.
<code>ippStsQualityIndexErr</code>	Quality Index cannot be calculated for an image filled with a constant.
<code>ippStsResizeNoOperationErr</code>	One of the output image dimensions is less than 1 pixel.
<code>ippStsBlockStepErr</code>	Step for Block is less than 8.
<code>ippStsMBStepErr</code>	Step for MB is less than 16.
<code>ippStsNoiseRangeErr</code>	Noise value for Wiener Filter is out of range.
<code>ippStsLPCCalcErr</code>	Cannot evaluate linear prediction.
<code>ippStsJPEG2KBadPassNumber</code>	Pass number exceeds the limits of [0, nOfPasses-1].
<code>ippStsJPEG2KDamagedCodeBlock</code>	Codeblock for decoding is damaged.
<code>ippStsH263CBPYCodeErr</code>	Illegal Huffman code is detected during CVPY stream processing.
<code>ippStsH263MCBPCInterCodeErr</code>	Illegal Huffman code is detected during MCBPC Inter stream processing.
<code>ippStsH263MCBPCIntraCodeErr</code>	Illegal Huffman code is detected during MCBPC Intra stream processing.
<code>ippStsNotEvenStepErr</code>	Step value is not pixel multiple.
<code>ippStsHistoNofLevelsErr</code>	Number of levels for histogram is less than 2.
<code>ippStsLUTNofLevelsErr</code>	Number of levels for LUT is less than 2.
<code>ippStsMP4BitOffsetErr</code>	Incorrect value for bit offset.
<code>ippStsMP4QPErr</code>	Incorrect value for quantization parameter.
<code>ippStsMP4BlockIdxErr</code>	Incorrect value for block index.
<code>ippStsMP4BlockTypeErr</code>	Incorrect block type.

Status Code	Message
ippStsMP4MVCCodeErr	Illegal Huffman code is detected during MV stream processing.
ippStsMP4VLCCodeErr	Illegal Huffman code is detected during VLC stream processing.
ippStsMP4DCCCodeErr	Illegal code is detected during DC stream processing.
ippStsMP4FcodeErr	Incorrect value for fcode.
ippStsMP4AlignErr	Incorrect buffer alignment.
ippStsMP4TempDiffErr	Incorrect temporal difference.
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock.
ippStsMP4ZeroBABErr	All BAB values are equal to zero.
ippStsMP4PredDirErr	Incorrect prediction direction.
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel.
ippStsMP4VideoCompModeErr	Incorrect video component mode.
ippStsMP4LinearModeErr	Incorrect DC linear mode.
ippStsH263PredModeErr	Incorrect value for Prediction Mode.
ippStsH263BlockStepErr	The step value is less than 8.
ippStsH263MBStepErr	The step value is less than 16.
ippStsH263FrameWidthErr	The frame width is less than 8.
ippStsH263FrameHeightErr	The frame height is less than, or equal to zero.
ippStsH263ExpandPelsErr	The expand pixels number is less than 8.
ippStsH263PlaneStepErr	Step value is less than the plane width.
ippStsH263QuantErr	Quantizer value is less than, or equal to zero, or more than 31.
ippStsH263MVCCodeErr	Illegal Huffman code is detected during MV stream processing.
ippStsH263VLCCodeErr	Illegal Huffman code is detected during VLC stream processing.
ippStsH263DCCCodeErr	Illegal code is detected during DC stream processing.
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64.
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed.
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is out of range.
ippStsJPEGOutOfBufErr	An attempt to access out of the buffer.
ippStsChannelOrderErr	Incorrect order of the destination channels.
ippStsZeroMaskValueErr	All values of the mask are equal to zero.
ippStsRangeErr	Incorrect values for bounds: the lower bound is greater than the upper bound.
ippStsQPErr	Incorrect value for a quantizer parameter.
ippStsQuadErr	The quadrangle is nonconvex or degenerates into triangle, line, or point.
ippStsRectErr	Size of the rectangular region is less than, or equal to 1.
ippStsCoeffErr	Incorrect values for the transformation coefficients.
ippStsNoiseValErr	Incorrect value for the noise amplitude for dithering.
ippStsDitherLevelsErr	Number of dithering levels is out of range.
ippStsNumChannelsErr	Incorrect or unsupported number of channels.
ippStsDataTypeErr	Incorrect or unsupported data type.
ippStsCOIErr	COI is out of range.
ippStsOutOfRangeErr	Argument is out of range or point is outside the image.
ippStsDivisorErr	Divisor is equal to zero, function is aborted.
ippStsAlphaTypeErr	Illegal type of image composition operation.
ippStsGammaRangeErr	Gamma range bound is less than, or equal to zero.
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than, or equal to 1.
ippStsChannelErr	Illegal channel number.
ippStsJaehneErr	Magnitude value is negative.
ippStsStepErr	Step value is less than, or equal to zero.
ippStsStrideErr	Stride value is less than the row length.

Status Code	Message
ippStsEpsValErr	Negative epsilon value.
ippStsScaleRangeErr	Scale bounds are out of range.
ippStsThresholdErr	Invalid threshold bounds.
ippStsWtOffsetErr	Invalid offset value for the wavelet filter.
ippStsAnchorErr	Anchor point is outside the mask.
ippStsMaskSizeErr	Invalid mask size.
ippStsShiftErr	Shift value is less than zero.
ippStsSampleFactorErr	Sampling factor is less than, or equal to zero.
ippStsResizeFactorErr	Resize factor(s) is less than, or equal to zero.
ippStsDivByZeroErr	An attempt to divide by zero.
ippStsInterpolationErr	Invalid interpolation mode.
ippStsMirrorFlipErr	Invalid flip mode.
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations.
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation.
ippStsContextMatchErr	Context parameter does not match the operation.
ippStsFftFlagErr	Invalid value of the FFT flag parameter.
ippStsFftOrderErr	Invalid value of the FFT order parameter.
ippStsMemAllocErr	Not enough memory for the operation.
ippStsNullPtrErr	Pointer is NULL.
ippStsSizeErr	Wrong value for the data size.
ippStsBadArgErr	Invalid or bad argument.
ippStsNoErr	No errors.
ippStsNoOperation	No operation has been executed.
ippStsMisalignedBuf	Misaligned pointer in operation in which it must be aligned.
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt.
ippStsInvZero	INF result. Zero value was met by InvThresh with zero level.
ippStsEvenMedianMaskSize	Even size of the Median Filter mask was replaced by the odd number.
ippStsDivByZero	Zero value(s) of the divisor in the function Div .
ippStsLnZeroArg	Zero value(s) of the argument in the function Ln.
ippStsLnNegArg	Negative value(s) of the argument in the function Ln.
ippStsNanArg	Argument value is not a number.
ippStsDoubleSize	Sizes of image are not multiples of 2.
ippStsJPEGMarker	JPEG marker in the bitstream.
ippStsResFloor	All result values are floored.
ippStsAffineQuadChanged	The fourth vertex of destination quad is not equal to the customer's one.
ippStsWrongIntersectROI	Incorrect ROI that has no intersection with the source or destination image. No operation.
ippStsWrongIntersectQuad	Incorrect quadrangle that has no intersection with the source or destination ROI. No operation.
ippStsSymKernelExpected	The kernel is not symmetric.
ippStsEvenMedianWeight	Even weight of the Weighted Median Filter was replaced by the odd one.
ippStsNoAntialiasing	The mode does not support antialiasing.
ippStsAlgTypeErr	The algorithm type is not supported.
ippStsAccurateModeNotSupported	Accurate mode is not supported.

The status codes ending with Err (except for the ippStsNoErr status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code `ippStsNoErr` indicates no error. All other status codes indicate warnings. When a specific case is encountered, the function execution is completed and the corresponding warning status is returned. For example, if the function `ippiDiv` meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status `ippStsDivByZero`. See appendix A *Handling of Special Cases* for more information.

## Platform-Aware Functions for Image Processing

Intel® Integrated Performance Primitives (Intel® IPP) library provides so-called platform-aware functions. These functions use the special data type `IppSizeL` for object sizes. The `IppSizeL` data type represents memory-related quantities: it can be 32- or 64-bit wide depending on the target architecture.

While the rest of Intel IPP functions support only objects of 32-bit integer size, platform-aware functions can work with 64-bit object sizes if it is supported by the platform. The API of platform-aware functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish platform-aware functions by the `L` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_L`.

Currently, the following image processing functions have platform-aware APIs:

Function Group	Header	Function Name
Arithmetic Functions	<code>ippi.h/ippi_l.h</code>	Add
		AddC
		Sub
		SubC
		Mul
		MulC
		Div
Linear Filters	<code>ippi.h/ippi_l.h</code>	FilterBilateralBorderGetBufferSize
		FilterBilateralBorderInit
		FilterBilateralBorder
Resize Transform Functions	<code>ippi.h/ippi_l.h</code>	ResizeGetSize
		ResizeGetBufferSize
		ResizeGetBorderSize
		Resize{Nearest Linear Cubic Lanczos Sinc}
		Resize{Nearest Linear Cubic Lanczos Sinc}
		ResizeAntialiasing{Nearest Linear Cubic}
		ResizeAntialiasing{Nearest Linear Cubic}
Support Functions	<code>ippi.h/ippi_l.h</code>	Malloc

Intel IPP platform-aware functions are documented as additional flavors to the existing functions declared in standard Intel IPP headers (without the `l` suffix). The `ippi_l.h` header is included into `ippi.h`.

## Threading Layer Functions

Intel® Integrated Performance Primitives (Intel® IPP) library provides threading layer (TL) functions for image processing. It is a set of functions that are implemented as wrappers over Intel IPP platform-aware functions by using tiling and multithreading with OpenMP\*. For implementation details, please see the corresponding source code files.

The API of TL functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish Intel IPP TL functions by the `_LT` or `_T` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_LT`. See [Extensions](#) for more information about function naming.

For more information about the TL functions usage, refer to the [Intel IPP Developer Guide](#).

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

## Structures and Enumerators

This section describes the structures and enumerators used by the Intel Integrated Performance Primitives for image processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See section [Error Reporting](#) in this chapter for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as

```
typedef struct {
    int x;
    int y;
} IppiPoint;
```

where `x`, `y` denote the coordinates of the point.

The structure `IppPointPolar` for storing the geometric position of a point in polar coordinates is defined as

```
typedef struct {
    Ipp32f rho;
    Ipp32f theta;
} IppPointPolar;
```

where `rho` - a radial coordinate (radial distance from the origin), `theta` - an angular coordinate (counterclockwise angle from the `x`-axis).

The structure `IppiSize` for storing the size of a rectangle is defined as

```
typedef struct {
    int width;
    int height;
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the `x`- and `y`-directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```
typedef struct {
    int x;
    int y;
    int width;
    int height;
} IppiRect;
```

where `x`, `y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the `x`-direction by `height` in the `y`-direction.

The `IppiConnectedComp` structure used in [Computer Vision functions](#) defines the connected component as follows:

```
typedef struct _IppiConnectedComp {
    Ipp64f area;
    Ipp64f value[3];
    IppiRect rect;
} IppiConnectedComp;
```

where `area` - area of the segmented component; `value[3]` - gray scale value of the segmented component; `rect` - bounding rectangle of the segmented component.

The `IppiMaskSize` enumeration defines the neighborhood area for some [morphological](#) and [filtering](#) functions:

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

The `IppCmpOp` enumeration defines the type of compare operation to be used in image [comparison functions](#):

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The `IppRoundMode` enumeration defines the rounding mode to be used in some conversion, filtering and arithmetic functions:

```
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial,
    ippRndHintAccurate=0x10
} IppRoundMode;
```

The `IppHintAlgorithm` enumeration defines the type of code to be used in some image transform and statistics functions, that is, faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see Table [Hint Arguments for Image Moment Functions](#).

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by [geometric transform functions](#) are defined as follows:

```
enum {
    IPPI_INTER_NN      = 1,
    IPPI_INTER_LINEAR   = 2,
    IPPI_INTER_CUBIC     = 4,
    IPPI_INTER_CUBIC2P_BSPLINE,
    IPPI_INTER_CUBIC2P_CATMULLROM,
    IPPI_INTER_CUBIC2P_B05C03,
    IPPI_INTER_SUPER     = 8,
    IPPI_INTER_LANCZOS   = 16,
    IPPI_ANTIALIASING   =(1 << 29)
    IPPI_SUBPIXEL_EDGE   =(1 << 30)
    IPPI_SMOOTH_EDGE     = IPP_MIN_32S
};
```

The `IppiAlphaType` enumeration defines the type of the compositing operation to be used in the [alpha composition functions](#):

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

The `IppiDitherType` enumeration defines the type of dithering to be used by the [ippiReduceBits](#) function:

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some [image format conversion functions](#) is defined as follows:

```
enum {
    IPP_UPPER      = 1,
    IPP_LEFT       = 2,
    IPP_CENTER     = 4,
    IPP_RIGHT      = 8,
    IPP_LOWER      = 16,
    IPP_UPPER_LEFT = 32,
    IPP_UPPER_RIGHT= 64,
```

```

    IPP_LOWER_LEFT   = 128,
    IPP_LOWER_RIGHT  = 256
};
```

The `IppiAxis` enumeration defines the flip axes for the `ippiMirror` functions or direction of the image intensity ramp for the `ippiImageRamp` functions:

```

typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth,
    ippAxs45,
    ippAxs135
} IppiAxis;
```

The `IppiBorderType` enumeration defines the border type that is used by some [Separable Filters](#) and [Fixed Filters](#) functions:

```

typedef enum _IppiBorderType {
    ippBorderRepl      = 1,
    ippBorderWrap      = 2,
    ippBorderMirror     = 3,
    ippBorderMirrorR    = 4,
    ippBorderDefault    = 5,
    ippBorderConst      = 6,
    ippBorderTransp     = 7,
    ippBorderInMemTop   = 0x0010,
    ippBorderInMemBottom = 0x0020,
    ippBorderInMemLeft   = 0x0040,
    ippBorderInMemRight  = 0x0080,
    ippBorderFirstStageInMemTop = 0x0100,
    ippBorderFirstStageInMemBottom = 0x0200,
    ippBorderFirstStageInMemLeft  = 0x0400,
    ippBorderFirstStageInMemRight = 0x0800,
} IppiBorderType;
```

The `IppiFraction` enumeration defines shapes of the structuring element used in some decimate filter functions:

```

typedef enum {
    ippPolyphase_1_2,
    ippPolyphase_3_5,
    ippPolyphase_2_3,
    ippPolyphase_7_10,
    ippPolyphase_3_4,
} IppiFraction;
```

The `IppiNormOp` enumeration defines the type of normalization that should be applied to the output data:

```

typedef enum {
    ippNormNone          = 0x00000000, // default
    ippNorm              = 0x00000100, // normalized form
    ippNormCoefficient  = 0x00000200, // correlation coefficient in the range [-1.0,...,1.0]
    ippNormMask          = 0x0000FF00,
} IppiNormOp;
```

The `IppiROIShape` enumeration defines the window shape for the two-dimensional convolution-specific functions:

```
typedef enum {
    ippiROIFull    = 0x00000000,
    ippiROIValid   = 0x00010000,
    ippiROISame    = 0x00020000,
    ippiROIMask    = 0x00FF0000
} IppiROIShape;
```

The `IppNormType` enumeration defines the norm type that should be applied when computing the magnitude of the gradient:

```
typedef enum {
    ippNormInf   = 0x00000001, // Infinity norm
    ippNormL1    = 0x00000002, // L1 normalization
    ippNormL2    = 0x00000004 // L2 normalization
} IppNormType;
```

The `IppiHOGConfig` structure defines the configuration parameters for the HOG descriptor:

```
typedef struct {
    int cvCompatible; /* openCV compatible output format */
    int cellSize;     /* square cell size (pixels) */
    int blockSize;    /* square block size (pixels) */
    int blockStride; /* block displacement (the same for x- and y- directions) */
    int nbins;        /* required number of bins */
    Ipp32f sigma;    /* gaussian factor of HOG block weights */
    Ipp32f l2thresh; /* normalization factor */
    IppiSize winSize; /* detection window size (pixels) */
} IppiHOGConfig;
```

The code flags used by the `FastN` functions are defined as follows:

```
enum {
    IPP_FASTN_ORIENTATION = 0x0001,
    IPP_FASTN_NMS         = 0x0002,
    IPP_FASTN_CIRCLE       = 0X0004,
    IPP_FASTN_SCORE_MODE0 = 0X0020
};
```

The `IppiFastNSpec` specification structure is used by the `FastN` function:

```
struct FastNSpec;
typedef struct FastNSpec IppiFastNSpec;
```

The `IppiCornerFastN` structure used by the `FastN2DToVec` function stores the destination vector of structures:

```
typedef struct _IppiCornerFastN {
    int x;
    int y;
    int cornerType;
    int orientation;
    float angle;
    float score;
} IppiCornerFastN;
```

The `IppFGMModel` structure contains parameters for the Gaussian mixture-based segmentation algorithm:

```
typedef struct
{
    unsigned int numFrames; /* length of history */
    unsigned int numGauss; /* maximal number of gaussian components per pixel */
```

```

        /* (numGauss<=maxNumGauss) */
Ipp32f varInit;      /* initial value of variance for new gaussian component */
Ipp32f varMin;       /* minimal bound of variance */
Ipp32f varMax;       /* maximal bound of variance */
Ipp32f varWBRatio;   /* background threshold */
Ipp32f bckgThr;     /* background total weights sum threshold */
Ipp32f varNGRatio;   /* threshold for adding new gaussian component to list */
Ipp32f reduction;    /* speed of reduction non-active gaussian components */
Ipp8u shadowValue;   /* returned shadow value */
char shadowFlag;     /* search shadows flag */
Ipp32f shadowRatio;  /* shadow threshold */
} IppFGMModel

```

The `IppiMorphMode` enumerator defines modes for mask processing at the second stage of advanced morphology operations:

```

typedef enum {
    IPP_MORPH_DEFAULT      = 0x0000,
    IPP_MORPH_MASK_NO_FLIP = 0x0001,
} IppiMorphMode;

```

The `IppChannels` enumerator defines the number of channels in the image:

```

typedef enum {
    ippC0      = 0,
    ippC1      = 1,
    ippC2      = 2,
    ippC3      = 3,
    ippC4      = 4,
    ippP2      = 5,
    ippP3      = 6,
    ippP4      = 7,
    ippAC1     = 8,
    ippAC4     = 9,
    ippA0C4    = 10,
    ippAP4     = 11
} IppChannels

```

The `IppiFilterBilateralType` enumerator defines the type of the bilateral filter that is used by some [Filtering Functions](#):

```

typedef enum {
    ippiFilterBilateralGauss      = 100,
    ippiFilterBilateralGaussFast = 101
} IppiFilterBilateralType

```

The `IppiWarpTransformType` enumerator defines the type of the warp transform for some [Warp Functions with Prior Initialization](#):

```

typedef enum {
    ippWarpAffine,
    ippWarpPerspective,
    ippWarpBilinear
} IppiWarpTransformType

```

The `IppiDistanceMethodType` structure stores the method of defining the difference in intensity between pixels. It is defines as:

```

typedef enum {
    ippDistNormL1 = 0x00000002;
    ippDistNormL2 = 0x00000004;
} IppiDistanceMethodType;

```

## Structures for 3D Data Processing Functions

The `ipprBorderType` enumeration defines the border type that is used by some [3D Data Processing Functions](#):

```
typedef enum _IpprBorderType {
    ipprBorderRepl      = ippBorderRepl,
    ipprBorderConst     = ippBorderConst,

    /* Flags to use source image memory pixels from outside of the border in particular
    directions */
    ipprBorderInMemTop   = 0x0010,
    ipprBorderInMemBottom = 0x0020,
    ipprBorderInMemLeft   = 0x0040,
    ipprBorderInMemRight  = 0x0080,
    ipprBorderInMemFront  = 0x1000,
    ipprBorderInMemBack  = 0x2000,

    ipprBorderInMem       = ipprBorderInMemLeft|ipprBorderInMemTop|ipprBorderInMemRight|
    ipprBorderInMemBottom|ipprBorderInMemFront|ipprBorderInMemBack,
} IpprBorderType;
```

The `IpprVolumeL` structure stores the volume of a three-dimensional space. It is defined as:

```
typedef struct {
    int width;
    int height;
    int depth;
} IpprVolume;
```

The `IpprCuboid` structure stores the volume of interest of a three-dimensional space. It is defined as:

```
typedef struct {
    int x;
    int y;
    int z;
    int width;
    int height;
    int depth;
} IpprCuboid;
```

## Function Context Structures

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the `IppiFFTSpec` structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform.

Two different kinds of structures are used: structures that are not modified during function operation - they have the suffix `Spec` in their names, and structures that are modified during operation - they have the suffix `State` in their names.

These context-related structures are not defined in the public headers, and their fields are not accessible. It was done because the function context interpretation is processor dependent. Thus, you may only use context-related functions and may not create a function context as an automatic variable.

## Structures and Enumerators for Platform-Aware Functions

This topic describes the structures and enumerators used by the Intel IPP platform-aware functions for image processing.

The `IppiPointL` structure stores the geometric position of a point. It is defined as:

```
typedef struct {
    IppSizeL x;
    IppSizeL y;
} IppiPointL;
```

where `x`, `y` denote the coordinates of the point.

The `IppiSizeL` structure stores the size of a rectangle. It is defined as:

```
typedef struct {
    IppSizeL width;
    IppSizeL height;
} IppiSizeL;
```

where `width` and `height` denote the dimensions of the rectangle in the `x`- and `y`-directions, respectively.

The `IppiRectL` structure stores the geometric position and size of a rectangle. It is defined as:

```
typedef struct {
    IppSizeL x;
    IppSizeL y;
    IppSizeL width;
    IppSizeL height;
} IppiRectL;
```

where `x`, `y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the `x`-direction by `height` in the `y`-direction.

The `IpprVolumeL` structure stores the volume of a three-dimensional space. It is defined as:

```
typedef struct {
    IppSizeL width;
    IppSizeL height;
    IppSizeL depth;
} IpprVolumeL;
```

where `width`, `height`, and `depth` denote the dimensions of the three-dimensional space.

The `IpprBorderType` enumerator defines the border type that is used by some [3D Data Processing Functions](#):

```
typedef enum _IpprBorderType {
    ipprBorderRepl      = ippBorderRepl,
    ipprBorderConst     = ippBorderConst,

    ipprBorderInMemTop   = 0x0010,
    ipprBorderInMemBottom = 0x0020,
    ipprBorderInMemLeft   = 0x0040,
    ipprBorderInMemRight  = 0x0080,
    ipprBorderInMemFront  = 0x1000,
    ipprBorderInMemBack   = 0x2000
} IpprBorderType;
```

## Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR...GGG...BBB.

Functions that operate on planar images are identified by the presence of `Pn` descriptor in their names. In this case, `n` pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see [Function Naming](#) in this chapter for details). The data may be signed (`s`), unsigned (`u`), or floating-point real (`f`). For some arithmetic and FFT/DFT functions, data in complex format (`sc` or `fc`) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `8u_C3` descriptor, where `8u` represents 8-bit unsigned data for each channel and `C3` represents three channels.

Some functions operate with images in 16-bit packed RGB format (see [RGB Image Formats](#) in Chapter 6 for more details). In this case data of all 3 channels are represented as `16u` data type.

For example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `16u_C3` descriptor, where `16u` represents 16-bit unsigned data (not a bit depth) for all packed channels together and `C3` stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last or the first one. This data type is indicated by the `AC4` or `A0C4` descriptors respectively. The presence of the alpha channel can modify the function's behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually is not performed on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

### Image Data Types and Ranges

Data Type	Lower Bound		Upper Bound	
	Identifier	Value	Identifier	Value
8s	<code>IPP_MIN_8S</code>	-128	<code>IPP_MAX_8S</code>	127
8u		0	<code>IPP_MAX_8U</code>	255
16s	<code>IPP_MIN_16S</code>	-32768	<code>IPP_MAX_16S</code>	32767
16u		0	<code>IPP_MAX_16U</code>	65535
32s	<code>IPP_MIN_32S</code>	$-2^{31}$	<code>IPP_MAX_32S</code>	$2^{31}-1$
32u		0	<code>IPP_MAX_32U</code>	$2^{32}-1$
32f <sup>†</sup>	<code>IPP_MINABS_32F</code>	$1.175494351e^{-38}$	<code>IPP_MAXABS_32F</code>	$3.402823466e^{38}$

<sup>†</sup> The range for absolute values

## Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image except for alpha channel. It means that the same operation is applied to each channel, and the computed results do not depend on values of other channels. Some exceptions include the [ippiFilterMedianColor](#) function and color conversion functions, which process three channels together.

Intel IPP image processing functions can be divided by two major models of operation:

- **Point operations:** functions operate on one pixel to compute the result, for example, [ippiAdd](#)
- **Neighborhood operations:** functions operate on a group of pixels, for example, [ippiFilterBox](#)

### See Also

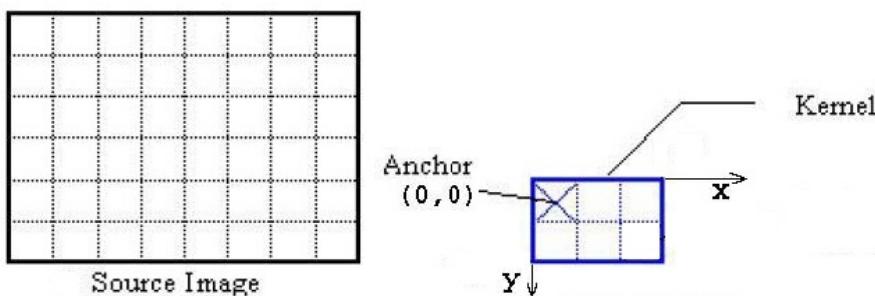
[FilterMedianColor](#) Filters an image using a color median filter.

[Add](#) Adds pixel values of two images.

[FilterBox](#) Blurs an image using a simple box filter.

## Neighborhood Operations

The result of a neighborhood operation is based on values of a certain group of pixels located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel as shown in the following figure.



The anchor cell is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. The kernel is placed on the image in such a way that the anchor cell coincides with the input pixel. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the top left corner of the kernel.

If position of the anchor cell is not specified explicitly in the function description, coordinates of the anchor are computed by the default formula:

$$\text{anchor.x} = (\text{kernel.width}-1)/2$$

$$\text{anchor.y} = (\text{kernel.height}-1)/2$$

where

`kernel.width` and `kernel.height` is the width and height of the filter kernel, respectively.

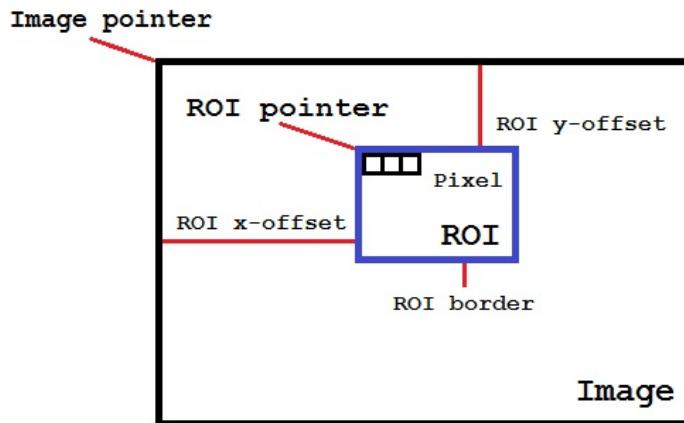
## Regions of Interest in Intel IPP

Most Intel IPP image processing functions operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

The Intel IPP functions with ROI support have an `R` descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in figure . The origin of an image is in the top left corner, with `x` values increasing from left to right and `y` values increasing downwards.

—border—top

## Image, ROI, and Offsets



Both the source and destination images can have a ROI. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. Image processing is performed on data of the source ROI, and the results are written to the destination ROI. In function call sequences, ROI is specified by:

- *roiSize* parameter of the `IppiSize` type
- *pSrc* and *pDst* pointers to the starts of source and destination ROI buffers
- *srcStep* and *dstStep* parameters that are equal to distances in bytes between the starting points of consecutive lines in source and destination images, respectively.

Thus, the *srcStep* and *dstStep* parameters set steps in bytes through image buffers to start processing a new line in the ROI of an image.

The following code example illustrates the use of the *dstStep* parameter in function calls:

### Example

```
IppStatus roi( void ) {
    Ipp8u x[8*3] = {0};
    IppiSize roiSize = {3,2};
    IppiPoint roiPoint = {2,1};
    // place the pointer to the ROI start position
    returnippiSet_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8, roisize );
}
```

The resulting image *x* contains the following data:

```
00 00 00 00 00 00 00 00
00 00 07 07 07 00 00 00
00 00 07 07 07 00 00 00
```

If ROI is present:

- source and destination images can have different sizes;
- lines may have padding at the end for aligning the line sizes;
- application must correctly define the *pSrc*, *pDst*, and *roiSize* parameters.

The *pSrc* and *pDst* parameters are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (`8u_C3R`), *pSrc* points to the start of the source ROI buffer and can be interpreted as follows:

$pSrc = pSrcImg + 3 * (srcImgSize.width * srcRoiOffset.y + srcRoiOffset.x)$ ,

where

---

<i>pSrcImg</i>	points to the start of the source image buffer
<i>srcImgSize</i>	is the image size in pixels (of the IppiSize type)
<i>srcRoiOffset</i>	determines an offset of ROI relative to the start of the image as shown in Figure <a href="#">Image, ROI, and Offsets</a> .

Another example for operations on four-channel image of 32-bit floating point data type `32f_AC4`:

```
pSrc = (Ipp32f*) ((Ipp8u*) pSrcImg + srcImgStep * srcRoiOffset.y
+ 4*SizeOf(Ipp32f)*srcRoiOffset.x.
```

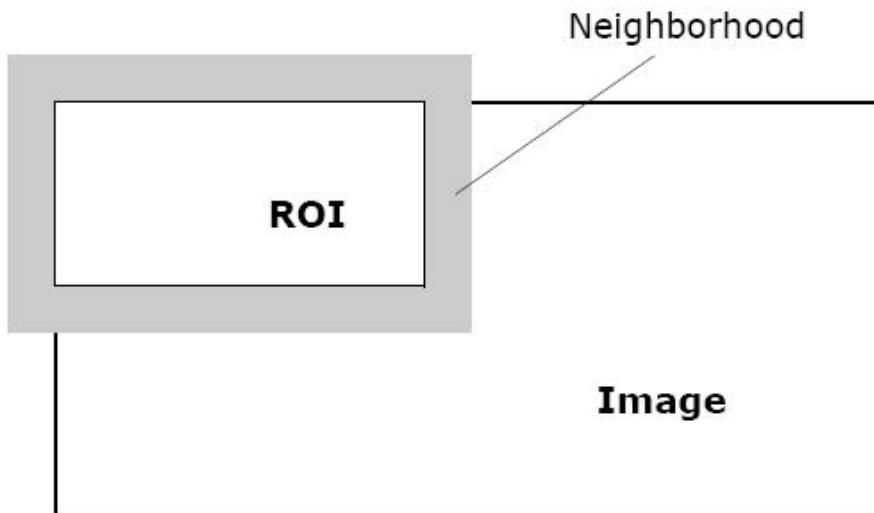
In this example the multiplier 4 is used because the AC4 pixel consists of 4 values - R, G, B, A. Pointer type conversion is required as in Intel IPP all image steps are always in bytes, and it is not recommended to use `step/SizeOf(Ipp32f)` as in a general case step value may be not a multiple of 4.

For functions using ROI with a neighborhood, you should correctly use values of the *pSrc* and *roiSize* parameters. These functions assume that the points in the neighborhood exist and that therefore *pSrc* is almost never equal to *pSrcImg*. Figure Using ROI with Neighborhood illustrates the case when neighborhood pixels can fall outside the source image.

`_border_top`

### Using ROI with Neighborhood

---




---

To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

#### Warning

If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

#### Example

```
IppStatus alignedLine( void
) {
    Ipp8u x[8*3] = {0};
```

```
IppiSize imgSize = {5,3};  
/// The image is of size 5x3. Width 8 has been  
/// chosen by the user to align every line of the image  
returnippiSet_8u_C1R( 7, x, 8, imgSize);  
}
```

The resulting image *x* contains the following data:

```
07 07 07 07 07 00 00 00  
07 07 07 07 07 00 00 00  
07 07 07 07 07 00 00 00
```

## See Also

[Borders in Neighborhood Operations](#)

## Tiled Image Processing

Intel IPP can process images composed from tiles, or tiled images.

# Support Functions

3

This chapter describes the Intel® IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions
- execute Intel IPP Threading Layer service routines.

## Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

### GetLibVersion

*Returns information about the used version of Intel IPP software for image processing.*

#### Syntax

```
const IppLibraryVersion*ippiGetLibVersion(void);
```

#### Include Files

ippi.h

#### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

#### Description

This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of Intel IPP for image processing. You need not release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

<i>major</i>	the major number of the current library version
<i>minor</i>	the minor number of the current library version
<i>majorBuild</i>	the number of builds of the major version
<i>build</i>	current build number
<i>targetCpu[4]</i>	Intel® processor.
<i>Name</i>	the name of the current library version
<i>Version</i>	the library version string

<i>BuildDate</i>	the library version actual build date
------------------	---------------------------------------

For example, if the library version is 9.0, build revision number is 49671, library name is `ippIP AVX2`, target CPU is processor with Intel® Advanced Vector Extensions 2 (Intel® AVX2) and build date is “Dec 7 2015”, then the fields in this structure are set as:

```
major = 9, minor = 0, Name = "ippIP AVX2", Version = "9.0.1 (r49671)", targetCpu[4] = "h9", BuildDate = "Dec 7 2015"
```

#### **NOTE**

Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are:

`ippiGetLibVersion`, `ippcvGetLibVersion`, and `ippccGetLibVersion`. They are declared in the following header files: `ippcore.h`, `ippcv.h`, `ippcc.h`, respectively, and have the same interface as the above described function.

---

## Status Information Function

---

Use this function to get a brief description of the status code returned by the current Intel IPP software.

### **ippGetString**

*Translates a status code into a message.*

---

#### **Syntax**

```
const char* ippGetString(IppStatus StsCode);
```

#### **Include Files**

`ippcore.h`

#### **Parameters**

<i>StsCode</i>	Code that indicates the status type (see Table 2-1)
----------------	---

#### **Description**

This function returns a pointer to the text string associated with a status code *StsCode*. Use this function to produce error and warning messages. The returned pointer is a pointer to an internal static buffer and needs not be released.

The status information function translates this code into the corresponding message Null Pointer Error:

## Example

A code example below shows how to use the `ippiGetStatusString` function. If you call an Intel IPP function, in this example `ippiSet_8u_C1R`, with a NULL pointer, it returns an error code -8.

```
void statusInfo( void ) {
    IppiSize roi = {0};
    IppStatus st =ippiSet_8u_C1R(3, 0, 0, roi );
    printf( " %d : %s\n", st, ippGetStatusString( st ) );
}
Output:
-8, Null Pointer Error
```

## Memory Allocation Functions

This section describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free previously allocated memory.

### NOTE

The only function to free the memory allocated by any of these functions is `ippiFree()`.

## Malloc

*Allocates memory aligned to 64-byte boundary.*

### Syntax

#### Case 1: Memory allocation for blocks of 32-bit sizes

```
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels, int* pStepBytes);
```

Supported values for *mod*:

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3
8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4

#### Case 2: Memory allocation for platform-aware functions

```
Ipp<datatype>* ippiMalloc_<mod>(IppSizeL widthPixels, IppSizeL heightPixels, IppSizeL* pStepBytes);
```

Supported values for *mod*:

8u_C1_L	16u_C1_L	16s_C1_L	32s_C1_L	32f_C1_L	32sc_C1_L	32fc_C1_L
8u_C2_L	16u_C2_L	16s_C2_L	32s_C2_L	32f_C2_L	32sc_C2_L	32fc_C2_L
8u_C3_L	16u_C3_L	16s_C3_L	32s_C3_L	32f_C3_L	32sc_C3_L	32fc_C3_L
8u_C4_L	16u_C4_L	16s_C4_L	32s_C4_L	32f_C4_L	32sc_C4_L	32fc_C4_L

`8u_AC4_L`    `16u_AC4_L`    `16s_AC4_L`    `32s_AC4_L`    `32f_AC4_L`    `32sc_AC4_L`    `32fc_AC4_L`

## Include Files

`ippi.h`

Flavors with the `_L` suffix `ippi_l.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>widthPixels</code>	Width of an image, in pixels.
<code>heightPixels</code>	Height of an image, in pixels.
<code>pStepBytes</code>	Pointer to the distance, in bytes, between the starting points of consecutive lines in the image

## Description

This function allocates a memory block aligned to 64-byte boundary for elements of different data types. Every line of the image is aligned in accordance with the `pStepBytes` parameter, which is calculated by the `Malloc` function and returned for further use.

The function `Malloc` allocates one continuous memory block. Functions that operate on planar images require an array of separate pointers (`IppType* plane[3]`) to each plane as an input. In this case, you should call the `Malloc` function three times.

## Example

The code example below demonstrates how to construct an array and set correct values to the pointers to use the allocated memory block with the Intel IPP functions operating on planar images. You need to specify `pStepBytes` for each plane. The example is given for the `8u` data type.

```
int stepBytes[3];
Ipp8u* plane[3];
plane[0] =ippiMalloc_8u_C1(widthPixels, heightPixels,
                           &(stepBytes [0]));
plane[1] =ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                           &(stepBytes [1]));
plane[2] =ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                           &(stepBytes [2]));
```

## Return Values

The return value of `Malloc` function is a pointer to an aligned memory block.

If no memory is available in the system, the `NULL` value is returned.

To free the allocated memory block, use the `Free` function.

## Free

Frees memory allocated by the function `ippiMalloc`.

## Syntax

```
voidippiFree(void* ptr);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>ptr</i>	Pointer to a memory block to be freed. This block must have been previously allocated by the function <code>ippiMalloc</code> .
------------	---

## Description

This function frees the aligned memory block allocated by the function `ippiMalloc`.

### NOTE

The function `ippiFree` cannot be used to free memory allocated by standard functions like `malloc` or `calloc`, nor can the memory allocated by `ippiMalloc` be freed by `free`.

# Threading Layer Functions

This section described the Intel IPP Threading Layer (TL) functions for image processing. For more information about the TL functions usage, refer to the [Intel IPP Developer Guide](#).

## SplitUniform2D

*Splits an image into tiles.*

### Syntax

#### Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatusippiSplitUniform2D_LT(IppiSizeL roiSize, IppiSizeL minItemNumber, IppiPointL* pSplit, IppiSizeL* pTileSize, IppiSizeL* pTailSize);
```

#### Case 2: Operation with TL functions based on the Classic API

```
IppStatusippiSplitUniform2D_T(IppiSize roiSize, int minItemNumber, IppiPoint* pSplit, IppiSize* pTileSize, IppiSize* pTailSize);
```

## Include Files

ippi\_tl.h

## Parameters

<i>roiSize</i>	Image ROI size.
----------------	-----------------

<i>minItemNumber</i>	Minimal size of a tile, in pixels.
----------------------	------------------------------------

<i>pSplit</i>	Number of split parts along x and y axes.
<i>pTileSize</i>	Size of a tile.
<i>pTailSize</i>	Size of the last corner tile.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function splits an image into tiles.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the <i>pSplit</i> , <i>pTileSize</i> , or <i>pTailSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>minItemNumber</i> is less than zero.

## ParallelFor

*Performs parallel iterations for a processing function.*

---

## Syntax

### Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippParallelFor_LT(IppSizeL numTiles, void* arg, functype_l func);
```

### Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippParallelFor_T(int numTiles, void* arg, functype func);
```

## Include Files

`ippcore_tl.h`

## Parameters

<i>numTiles</i>	Number of tiles.
<i>arg</i>	Pointer to the structure that contains arguments for the processing function.
<i>func</i>	Pointer to the processing function used in the "parallel for" loop.

## Description

This function performs parallel iterations of a processing function, which is passed as an argument for each tile.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
	Any IPP error that the processing function can return.

## GetTilePointer

*Returns a pointer to the specified image tile.*

---

## Syntax

### Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatusippiGetTilePointer_32f_LT(const Ipp32f* pSrc, Ipp32f** pDst, IppSizeL
srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
```

```
IppStatusippiGetTilePointer_64f_LT(const Ipp64f* pSrc, Ipp64f** pDst, IppSizeL
srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
```

### Case 2: Operation with TL functions based on the Classic API

```
IppStatusippiGetTilePointer_32f_T(const Ipp32f* pSrc, Ipp32f** pDst, int srcStep, int
x, int y, Ipp32s numChannels);
```

```
IppStatusippiGetTilePointer_64f_T(const Ipp64f* pSrc, Ipp64f** pDst, int srcStep, int
x, int y, Ipp32s numChannels);
```

## Include Files

`ippi_tl.h`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the memory location of the pointer to the destination image.
<i>srcStep</i>	Distance in bytes between consecutive lines in the source image.
<i>x</i>	x coordinate of a tile, in pixels.
<i>y</i>	y coordinate of a tile, in pixels.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, and 4.

## Description

This function returns a pointer to the specified image tile.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>x</i> or <i>y</i> is less than zero.
<code>ippStsNumChannelsErr</code>	Indicates an error condition when <i>numChannels</i> has an illegal value.

## GetTileParamsByIndex

Returns the offset and size of a tile by a given index.

## Syntax

### Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatusippiGetTileParamsByIndex_LT(IppSizeL index, IppiPointL splitImage, IppiSizeL
tileSize, IppiSizeL tailSize, IppiPointL* pTileOffset, IppiSizeL* pTilesSize);
```

**Case 2: Operation with TL functions based on the Classic API**

```
IppStatusippiGetTileParamsByIndex_T(int index, IppiPoint splitImage, IppiSize tileSize, IppiSize tailSize, IppiPoint* pTileOffset, IppiSize* pTileSize);
```

**Include Files**

`ippi_tl.h`

**Parameters**

<code>index</code>	Ordinal index of a tile.
<code>splitImage</code>	Split of the image by x and y axis correspondingly.
<code>tileSize</code>	Size of a tile.
<code>tailSize</code>	Size of the last bottom right tile.
<code>pTileOffset</code>	Offset of the tile corresponding to the top left image corner.
<code>pTileSize</code>	Size of a tile.

**Description**

This function returns the offset and size of a tile by a given index.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when the <code>pTileOffset</code> or <code>pTileSize</code> pointer is NULL.

**GetThreadingType**

*Returns type of the threading layer.*

---

**Syntax****Case 1: Operation with TL functions based on the Platform Aware API**

```
IppStatusippiGetThreadingType_LT(IppThreadingType* thrType);
```

**Case 2: Operation with TL functions based on the Classic API**

```
IppStatusippiGetThreadingType_T(IppThreadingType* thrType);
```

**Include Files**

`ippcore_tl.h`

**Parameters**

<code>thrType</code>	Pointer to the threading type.
----------------------	--------------------------------

**Description**

This function returns `OMP` if the OpenMP\* Threading Layer and `TBB` if the TBB\* Threading Layer is used.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

ippStsNullPtrErr                    Indicates an error condition when the *thrType* pointer is NULL.

## GetThreadId

*Returns a unique thread identification number.*

---

### Syntax

#### Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippGetThreadId_LT(int* pThrIdx);
```

#### Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippGetThreadId_T(int* pThrIdx);
```

### Include Files

ippcore\_tl.h

### Parameters

*pThrIdx*                            Pointer to the index of a thread.

### Description

This function returns a unique thread identification number.

### Return Values

ippStsNoErr                            Indicates no error.

ippStsNullPtrErr                    Indicates an error condition when the *pThrIdx* pointer is NULL.

# Image Data Exchange and Initialization Functions

4

This chapter describes the Intel® IPP image processing functions that perform image data manipulation, exchange and initialization operations.

## Convert

*Converts image pixel values from one data type to another.*

### Syntax

#### Case 1: Conversion to increase bit depth and change signed to unsigned type

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u16u_C1R	8u16s_C1R	8u32s_C1R	8u32f_C1R	8s32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8u32f_C3R	8s32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8u32f_C4R	8s32s_C4R
8s32f_C1R	16u32s_C1R	16u32f_C1R	16s32s_C1R	16s32f_C1R
8s32f_C3R	16u32s_C3R	16u32f_C3R	16s32s_C3R	16s32f_C3R
8s32f_C4R	16u32s_C4R	16u32f_C4R	16s32s_C4R	16s32f_C4R
8s8u_C1Rs	16s16u_C1Rs	16u32u_C1R	32s32u_C1Rs	32u32f_C1R
8s16u_C1Rs	16s32u_C1Rs		32s32f_C1R	
8s16s_C1R				
8s32u_C1Rs				
8s64f_C1R				

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8u32f_AC4R	8s32s_AC4R
8s32f_AC4R	16u32s_AC4R	16u32f_AC4R	16s32s_AC4R	16s32f_AC4R

#### Case 2: Conversion to reduce bit depth and change unsigned to signed type: integer to integer type

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R

---

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int
scaleFactor);
```

**Supported values for mod:**

8u8s_C1RSfs	16u8s_C1RSfs	32u8u_C1RSfs	32s16u_C1RSfs
	16s8s_C1RSfs	32u8s_C1RSfs	32s16s_C1RSfs
	16u16s_C1RSfs	32u16u_C1RSfs	
		32u16s_C1RSfs	
		32u32s_C1RSfs	

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

**Supported values for mod:**

16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R
------------	------------	------------	------------

#### Floating point to integer type:

```
IppStatusippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst,
int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

**Supported values for mod:**

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R

---

```
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int
scaleFactor);
```

**Supported values for mod:**

32f8u_C1RSfs	32f8s_C1RSfs	32f16u_C1RSfs	32f16s_C1RSfs	32f32u_C1RSfs	3
					2
					f
					3
					2
					s
					—
					C
					1
					R
					S
					f
					s
64f8u_C1RSfs	64f8s_C1RSfs	64f16u_C1RSfs	64f16s_C1RSfs		

```
IppStatusippiConvert_32f32u_C1IRSfs(Ipp32u* pSrcDst, int srcDstStep, IppiSize roiSize,
IppRoundMode roundMode, int scaleFactor);
```

```
IppStatusippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst,
int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

Supported values for mod:

32f8u\_AC4R

32f8s\_AC4R

32f16u\_AC4R

32f16s\_AC4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operation.						
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.						
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.						
<i>roiSize</i>	Size of the source and destination ROI in pixels.						
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).						
<i>roundMode</i>	Rounding mode, the following values are possible: <table border="0" style="margin-left: 20px;"> <tr> <td><i>ippRndZero</i></td><td>specifies that floating-point values are truncated to zero,</td></tr> <tr> <td><i>ippRndNear</i></td><td>specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,</td></tr> <tr> <td><i>ippRndFinancial</i></td><td>specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.</td></tr> </table>	<i>ippRndZero</i>	specifies that floating-point values are truncated to zero,	<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,	<i>ippRndFinancial</i>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
<i>ippRndZero</i>	specifies that floating-point values are truncated to zero,						
<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,						
<i>ippRndFinancial</i>	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.						

## Description

This function operates with ROI.

This function converts pixel values in the source image ROI  $pSrc$  to a different data type and writes them to the destination image ROI  $pDst$ .

The result of integer operations is always saturated to the destination data type range. It means that if the value of the source pixel is out of the data range of the destination image, the value of the corresponding destination pixel is set to the value of the lower or upper bound (minimum or maximum) of the destination data range:

```
x = pSrc[i, j]
if (x > MAX_VAL) x = MAX_VAL
if (x < MIN_VAL) x = MIN_VAL
pDst[i, j] = (CASTING) x
```

If you want to shift data from the signed range to the unsigned range and vice-versa, see "Application Notes" below.

The function flavors with the `Sfs` descriptor in their names perform scaling of the internally computed results in accordance with the `scaleFactor` parameter.

When converting from floating-point to integer type, rounding defined by `roundMode` is performed, and the result is saturated to the destination data type range.

---

### NOTE

The bit order of each byte in the source image is inverse to the pixel order. It means that the first pixel in a row represents the last (seventh) bit of the first byte in a row.

---

## Application Notes

When data is converted from the signed integer to the corresponding unsigned integer and vice versa (`8s -->8u`, `16u --> 16s`), the pixel information may be lost because all negative values will be set to zero (signed-unsigned conversion), or unsigned values from the high half of the range will be set to the maximum value of the signed range (unsigned - signed conversion).

If you need just to shift the data from the signed range to the unsigned range and vice versa, use the function `ippiXorC` with the parameter `value` specified in such a way that the most significant bit is set to 1, and all other bits are set to 0. For example, if you want to convert pixel values from `Ipps16s` type to `Ipp16u` type with the range shift call the function:

```
ippiXorC_16u_C1R( (Ipp16u *)pSrc, srcStep, 0x8000, pDst, dstStep, roiSize);
```

In this case the pixels values are converted as follows:

```
-32768 --> 0
-32767 --> 1
...
-1 --> 32767
0 --> 32768
1 --> 32769
...
```

32766 --> 65534

32767 --> 65535

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in Case 4.
ippStsSizeErr	Indicates an error when <code>roiSize</code> has a field with zero or negative value, or <code>srcBitOffset/dstBitOffset</code> is less than zero.
ippStsStepErr	Indicates an error when <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
ippStsMemAllocErr	Indicates an error when memory allocation fails.

## Example

The code example below shows data conversion without scaling.

```
IppStatus convert( void ) {

    IppiSize roi={5,4};
    Ipp32f x[5*4];

    Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;

    returnippiConvert_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, ippRndNear );
}
```

The destination image `y` contains:

```
00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

## See Also

[Integer Result Scaling](#)

[Regions of Interest in Intel IPP](#)

## BinToGray, GrayToBin

*Converts a bitonal image to a grayscale image and vice versa.*

---

## Syntax

### Case 1: Conversion of a bitonal image to a grayscale image

```
IppStatusippiBinToGray_lu<dstDataType>_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize, Ipp<dstDataType>
loVal, Ipp<dstDataType> hiVal);
```

Supported values for `dstDataType`:

8u	16u	16s	32f
----	-----	-----	-----

### Case 2: Conversion of a grayscale image to a bitonal image

```
IppStatusippiGrayToBin_<srcDataType>lu_C1R(const Ipp<srcDataType>* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, Ipp<sourceDataType>
threshold);
```

Supported values for `srcDataType`:

8u	16u	16s	32f
----	-----	-----	-----

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>srcBitOffset</code>	Offset, in bits, from the first byte of the source image row.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>dstBitOffset</code>	Offset, in bits, from the first byte of the destination image row.
<code>roiSize</code>	Size of the ROI in pixels.
<code>loVal</code>	Destination value that corresponds to the "0" value of the corresponding source element.
<code>hiVal</code>	Destination value that corresponds to the "1" value of the corresponding source element.
<code>threshold</code>	Threshold level.

## Description

These functions operate with ROI.

The `ippiBinToGray` function converts a bitonal image to grayscale, and the `ippiGrayToBin` function converts a grayscale image to bitonal. The data type of the bitonal image is `8u`. It means that each byte consists of eight consecutive pixels of the image (1 bit per pixel). You need to specify the start position of the ROI buffer in the `srcBitOffset` and `dstBitOffset` parameters.

The `ippiBinToGray` function transforms each bit of the source image into the pixel of the destination image in the following way:

- If the input pixel is equal to 0, the corresponding output pixel is set to `loVal`.
- If the input pixel is equal to 1, the corresponding output pixel is set to `hiVal`.

The `ippiGrayToBin` function transforms each pixel of the source image into the bit of the destination image in the following way:

- If the input pixel is more than the `threshold` value, the corresponding output bit is set to 1.
- If the input pixel is less than, or equal to the `threshold` value, the corresponding output bit is set to 0.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when the <code>srcStep</code> or <code>dstStep</code> value is less than, or equal to zero.
<code>ippStsSizeErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>roiSize</code> has a zero or negative value</li> <li>• the <code>srcBitOffset</code> or <code>dstBitOffset</code> value is less than zero</li> </ul>

## See Also

[Regions of Interest in Intel IPP](#)

## Scale

---

*Scales pixel values of an image and converts them to another bit depth.*

### Syntax

#### Case 1: Scaling with conversion to integer data of increased bit depth

```
IppStatusippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u16u_C1R</code>	<code>8u16s_C1R</code>	<code>8u32s_C1R</code>
<code>8u16u_C3R</code>	<code>8u16s_C3R</code>	<code>8u32s_C3R</code>
<code>8u16u_C4R</code>	<code>8u16s_C4R</code>	<code>8u32s_C4R</code>

```
IppStatusippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u16u_AC4R</code>	<code>8u16s_AC4R</code>	<code>8u32s_AC4R</code>
-------------------------	-------------------------	-------------------------

**Case 2: Scaling with conversion to floating-point data**

```
IppStatusippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

8u32f\_C1R  
8u32f\_C3R  
8u32f\_C4R

```
IppStatusippiScale_8u32f_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 3: Scaling of integer data with conversion to reduced bit depth**

```
IppStatusippiScale_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*>
pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for mod:

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R

```
IppStatusippiScale_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*>
pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for mod:

16u8u_AC4R	16s8u_AC4R	32s8u_AC4R
------------	------------	------------

**Case 4: Scaling of floating-point data with conversion to integer data type**

```
IppStatusippiScale_<mod>(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

32f8u\_C1R  
32f8u\_C3R  
32f8u\_C4R

```
IppStatusippiScale_32f8u_AC4R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting bytes of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin, vMax</i>	Minimum and maximum values of the input data.
<i>hint</i>	Option to select the algorithmic implementation of the function (see <a href="#">Hint Arguments for Image Moment Functions</a> ).

## Description

This function operates with ROI.

This function converts pixel values of a source image ROI *pSrc* to the destination data type, using a linear mapping. The computation algorithm is specified by the *hint* argument. For conversion between integer data types, the whole range [*src\_Min..src\_Max*] of the input data type is mapped onto the range [*dst\_Min..dst\_Max*] of the output data type.

The source pixel *p* is mapped to the destination pixel *p'* by the following formula:

$$p' = dst\_Min + k * (p - src\_Min)$$

where

$$k = (dst\_Max - dst\_Min) / (src\_Max - src\_Min)$$

For conversions to and from floating-point data type, the user-defined floating-point data range [*vMin..vMax*] is mapped onto the source or destination data type range.

If the conversion is from Ipp32f type and some of the input floating-point values are outside the specified input data range [*vMin..vMax*], the corresponding output values saturate. To determine the actual floating-point data range in your image, use the `ippiMinMax` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsScaleRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is, <i>vMax</i> is less than or equal to <i>vMin</i> .

## Example

The code example below shows how to use scaling to preserve the data range.

```
IppStatus scale( void ) {
    IppiSize roi = {5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}
```

The destination image *y* contains:

```
00 FF 80 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

## See Also

[Regions of Interest in Intel IPP](#)

[Image Moments](#)

[Intel® Integrated Performance Primitives Concepts](#)

[MinMax](#) Computes the minimum and maximum of image pixel values.

## ScaleC

*Scales pixel values of an image and converts them to another bit depth.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiScaleC_<mod>_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp64f mVal,
Ipp64f aVal, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm
hint);
```

Supported values for *mod*:

8u	8u8s	8u16u	8u16s	8u32s	8u32f	8u64f
8s8u	8s	8s16u	8s16s	8s32s	8s32f	8s64f
16u8u	16u8s	16u	16u16s	16u32s	16u32f	16u64f
16s8u	16s8s	16s16u	16s	16s32s	16s32f	16s64f
32s8u	32s8s	32s16u	32s16s	32s	32s32f	32s64f
32f8u	32f8s	32f16u	32f16s	32f32s	32f	32f64f
64f8u	64f8s	64f16u	64f16s	64f32s	64f32f	64f

where the first value is `srcDatatype` and the second value is `dstDatatype`.

### Case 2: In-place operation

```
IppStatusippiScaleC_<mod>_C1IR(const Ipp<datatype>* pSrcDst, int srcDstStep, Ipp64f mVal, Ipp64f aVal, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for `mod`:

8u	8s	16u	16s	32s	32f	64f
----	----	-----	-----	-----	-----	-----

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>pSrcDst</code>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
<code>srcDstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>mVal</code>	Value of the multiplier used for scaling.
<code>aVal</code>	Offset value for scaling.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting bytes of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI, in pixels.
<code>hint</code>	Option to select the algorithmic implementation of the function. Supported values are <code>ippAlgHintFast</code> (default) and <code>ippAlgHintAccurate</code> .

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function scales pixel values of the source image ROI and converts them to the destination data type according to the following formula:

```
dst = saturate_to_dstType(src * mVal + aVal)
```

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.
ippStsStepErr	Indicates an error when the step value is less than, or equal to zero.

## Example

To better understand usage of this function, refer to the `ScaleC.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

## Set

*Sets pixels of an array to a constant value.*

### Syntax

#### Case 1: Setting one-channel data to a value

```
IppStatusippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize  
roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
--------	---------	---------	---------	---------

#### Case 2: Setting each color channel to a specified value

```
IppStatusippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,  
IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 3: Setting color channels and alpha channel to specified values

```
IppStatusippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,  
IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

**Case 4: Setting masked one-channel data to a value**

```
IppStatusippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize  
roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
---------	----------	----------	----------	----------

**Case 5: Setting color channels of masked multi-channel data to specified values**

```
IppStatusippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,  
IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

**Case 6: Setting all channels of masked multi-channel data to specified values**

```
IppStatusippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,  
IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
---------	----------	----------	----------	----------

**Case 7: Setting selected channel of multi-channel data to a value**

```
IppStatusippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize  
roiSize);
```

Supported values for mod:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>value</i>	Constant value to assign to each pixel in the destination image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.

*maskStep*

Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

## Description

This function operates with ROI.

This function sets pixels in the destination image ROI *pDst* to the *value* constant. Either all pixels in a rectangular ROI, or only those selected by the specified mask *pMask*, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero. When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see **Case 7**), the *pDst* pointer points to the start of ROI buffer in the required channel. If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen *ippiSet* function flavor.

This function supports negative step value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> or <i>maskStep</i> has a zero value.

## Example

The code example below shows how to use the function `ippiSet_8u_C1R`.

```
void func_set()
{
    IppiSize roi = {5,4};
    Ipp8u x[8*4] = {0};

   ippiSet_8u_C1R(1, x, 8, roi);
}
```

Result:

```
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
```

## See Also

[Regions of Interest in Intel IPP](#)

## Copy

---

*Copies pixel values between two buffers.*

## Syntax

### Case 1: Copying all pixels of all color channels

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R
8u_C3AC4R	16u_C3AC4R	16s_C3AC4R	32s_C3AC4R	32f_C3AC4R
8u_AC4C3R	16u_AC4C3R	16s_AC4C3R	32s_AC4C3R	32f_AC4C3R

### Case 2: Copying masked pixels only

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

### Case 3: Copying a selected channel in a multi-channel image

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

### Case 4: Copying a selected channel to a one-channel image

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32s_C3C1R	32f_C3C1R
8u_C4C1R	16u_C4C1R	16s_C4C1R	32s_C4C1R	32f_C4C1R

### Case 5: Copying a one-channel image to a multi-channel image

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1C3R	16u_C1C3R	16s_C1C3R	32s_C1C3R	32f_C1C3R
8u_C1C4R	16u_C1C4R	16s_C1C4R	32s_C1C4R	32f_C1C4R

**Case 6: Splitting color image into separate planes**

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3P3R	16u_C3P3R	16s_C3P3R	32s_C3P3R	32f_C3P3R
----------	-----------	-----------	-----------	-----------

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4P4R	16u_C4P4R	16s_C4P4R	32s_C4P4R	32f_C4P4R
----------	-----------	-----------	-----------	-----------

**Case 7: Composing color image from separate planes**

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P3C3R	16u_P3C3R	16s_P3C3R	32s_P3C3R	32f_P3C3R
----------	-----------	-----------	-----------	-----------

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P4C4R	16u_P4C4R	16s_P4C4R	32s_P4C4R	32f_P4C4R
----------	-----------	-----------	-----------	-----------

**Case 8: Copying all pixels of all color channels with platform-aware functions**

```
IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

8u_C1R_L	16s_C1R_L	16u_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16s_C3R_L	16u_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16s_C4R_L	16u_C4R_L	32s_C4R_L	32f_C4R_L
8u_AC4R_L	16s_AC4R_L	16u_AC4R_L	32s_AC4R_L	32f_AC4R_L

**Include Files**

ippi.h

Flavors with the \_L suffix: ippi\_l.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI. The array storing pointers to the color planes of the source planar image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. The array storing pointers to the color planes of the destination planar image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

## Description

This function operates with ROI.

This function copies data from the source image *pSrc* to the destination image *pDst*. Copying pixels selected by a mask *pMask* is supported as well.

For masked operation (**Case 2**), the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors operating with the channel of interest (descriptor **C**) copy only one specified channel of a source multi-channel image to the channel of another multi-channel image (see **Case 3**). For these functions, the *pSrc* and *pDst* pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Some function flavors add alpha channel to the 3-channel source image (flavors with the **\_C3AC4R** descriptor), or discard alpha channel from the source image (flavors with the **\_AC4C3R** descriptor) - see **Case 1**.

Special function flavors copy data from only one specified channel *pSrc* of a multi-channel image to a one-channel image *pDst* (see **Case 4**), as well as to copy data from a one-channel image *pSrc* to only one specified channel of a multi-channel image *pDst* (see **Case 5**).

You can also use the `ippiCopy` function to convert the interleaved color image into separate planes and vice versa (see **Case 6** and **Case 7**).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in <b>Case 4</b> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> for <b>Cases 4</b> and <b>5</b> .

## Example

The code example below shows how to copy masked data.

```
IppStatus copyWithMask( void ) {
    Ipp8u mask[3*3], x[5*4], y[5*4]={0};
    IppiSize imgroi={5,4}, mskroi={3,3};
   ippiSet_8u_C1R( 3, x, 5, imgroi );
    /// set mask with a hole in upper left corner
    ippiSet_8u_C1R( 1, mask, 3, mskroi );
    mask[0] = 0;
    /// copy roi with mask
    return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
```

The destination image *y* contains:

```
00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00
```

## See Also

[Regions of Interest in Intel IPP](#)

## CopyManaged

*Copies pixel values between two images in accordance with the specified type of copying.*

### Syntax

```
IppStatus ippiCopyManaged_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int flags);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>flags</i>	Specifies the type of copying. Possible values are:

- `IPP_TEMPORAL_COPY` - standard copying
- `IPP_NONTEMPORAL_STORE` - copying without caching the destination image
- `IPP_NONTEMPORAL_LOAD` - processor uses non-temporal load instructions

## Description

This function operates with ROI.

This function copies data from a source image ROI `pSrc` to the destination image ROI `pDst`. The `flags` parameter specifies the type of copying that the function performs:

- When `flags` is set to `IPP_TEMPORAL_COPY`, the function is identical to the `ippiCopy_8u_C1R` function.
- When `flags` is set to `IPP_NONTEMPORAL_STORE`, the processor uses non-temporal store instructions. Copying is performed without caching the data of the destination image.
- When `flags` is set to `IPP_NONTEMPORAL_LOAD`, the processor uses non-temporal load instructions.

To achieve better performance, align data to 64-byte boundary.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a> .
Notice revision #20201201

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.

## See Also

[Regions of Interest in Intel IPP](#)

## CopyConstBorder

*Copies pixels values between two images and adds the border pixels with a constant value.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, Ipp<datatype> value);
```

Supported values for `mod`:

`8u_C1R`

`16u_C1R`

`16s_C1R`

`32s_C1R`

`32f_C1R`

**Case 2: In-place operation on one-channel data**

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
---------	----------	----------	----------	----------

**Case 3: Not-in-place operation on multi-channel data**

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
---------	----------	----------	----------	----------

**Case 5: Not-in-place operation on one-channel data with platform-aware functions**

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, Ipp<datatype> value);
```

Supported values for mod:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L
----------	-----------	-----------	-----------	-----------

**Case 6: In-place operation on one-channel data with platform-aware functions**

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L
-----------	------------	------------	------------	------------

**Case 7: Not-in-place operation on multi-channel data with platform-aware functions**

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_AC4R_L	16u_AC4R_L	16s_AC4R_L	32s_AC4R_L	32f_AC4R_L

```
IppStatusippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L
----------	-----------	-----------	-----------	-----------

**Case 8: In-place operation on multi-channel data with platform-aware functions**

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3IR_L	16u_C3IR_L	16s_C3IR_L	32s_C3IR_L	32f_C3IR_L
8u_AC4IR_L	16u_AC4IR_L	16s_AC4IR_L	32s_AC4IR_L	32f_AC4IR_L

```
IppStatusippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4IR_L	16u_C4IR_L	16s_C4IR_L	32s_C4IR_L	32f_C4IR_L
-----------	------------	------------	------------	------------

**Include Files**

ippi.h

Flavors with the \_L suffix: ippi\_l.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>pSrcDst</i>	Pointer to the source/destination image (for in-place flavors).
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source/destination image (for in-place flavors).
<i>srcRoiSize</i>	Size of the source ROI, in pixels.
<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>topBorderHeight</i>	Height of the top border, in pixels.
<i>leftBorderWidth</i>	Width of the left border, in pixels.
<i>value</i>	The constant value to assign to the border pixels (constant vector in case of multi-channel images).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the *value* argument. The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI (see Figure Creating a Border of Pixels with Constant Value.) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

### Creating a Border of Pixels with Constant Value

v	v	v	v	v	v	v	v	v
v	v	v	v	v	v	v	v	v
v	v	1	2	3	4	5	v	v
v	v	6	7	8	9	10	v	v
v	v	11	12	13	14	15	v	v
v	v	16	17	18	19	20	v	v
v	v	v	v	v	v	v	v	v

*topBorderHeight*=2  
*leftBorderWidth*=2

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with a zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

### Example

The code example below shows how to use the function `ippiCopyConstBorder_8u_C1R`.

```
Ipp8u src[8*4] = {3, 3, 3, 3, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 3, 3, 3, 3, 8, 8, 8};

Ipp8u dst[8*6];
IppiSize srcRoi = { 5, 4 };
```

```
IppiSize dstRoi = { 7, 6 };
int borderWidth = 1;
int borderHeight = 1;
int borderVal = 0;

ippiCopyConstBorder_8u_C1R(src, 8, srcRoi, dst, 8, dstRoi, borderHeight, borderWidth, borderVal);

Results
source image:
3 3 3 3 3 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8      src
3 3 3 3 3 8 8 8

destination image:
0 0 0 0 0 0 0
0 3 3 3 3 3 0
0 3 2 1 2 3 0
0 3 2 1 2 3 0      dst
0 3 3 3 3 3 0
0 0 0 0 0 0 0
```

## CopyMirrorBorder

*Copies pixels values between two images and adds the mirrored border pixels.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

#### Case 2: In-place operation

```
IppStatusippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep,
IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

**Case 3: Not-in-place operation with platform-aware functions**

```
IppStatusippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
```

Supported values for mod:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L

**Case 4: In-place operation with platform-aware functions**

```
IppStatusippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

Supported values for mod:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L
8u_C3IR_L	16u_C3IR_L	16s_C3IR_L	32s_C3IR_L	32f_C3IR_L
8u_C4IR_L	16u_C4IR_L	16s_C4IR_L	32s_C4IR_L	32f_C4IR_L

**Include Files**

ippi.h

Flavors with the \_L suffix: ippi\_l.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
<i>srcRoiSize</i>	Size of the source ROI, in pixels.
<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>topBorderHeight</i>	Height of the top border, in pixels.
<i>leftBorderWidth</i>	Width of the left border, in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image  $pSrc$  to the destination image  $pDst$  and fills pixels outside the copied area (border pixels) in the destination image with the values of the source image pixels according to the scheme illustrated in the figure below. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

### Creating a Mirrored Border

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13

`topBorderWidth=2`

`topBorderHeight=2`

#### NOTE

In-place flavors actually add border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters `topBorderHeight` and `leftBorderWidth` specify the position of the first pixel of the source ROI in the destination image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the `topBorderHeight` (`leftBorderWidth`) parameter.

#### NOTE

If border width is greater than the image size in the corresponding dimension, multiple reflections are obtained for this border.

**NOTE**

If you use this function for a tiled image, note that to perform correct mirroring, the size of a tile must be more than the size of the used border. For example, if the image referenced above is divided into two tiles of size 3x3 and 2x3, the second tile (cells are highlighted in yellow) is extended with top, right, and bottom borders (highlighted in gray). The width of the right border is not less than the second tile width, so the pixels (blue) required for constructing the border of the tiled image are out of the processed tile. Therefore the last column (red) of the border extended image is computed incorrectly:

13	12	11	12	13	14	15	14	15
8	7	6	7	8	9	10	9	10
3	2	1	2	3	4	5	4	5
8	7	6	7	8	9	10	9	10
13	12	11	12	13	14	15	14	15
18	17	16	17	18	19	20	19	20
13	12	11	12	13	14	15	14	15

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error in any of the following cases: <ul style="list-style-type: none"> <li><code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with a zero or negative value</li> <li><code>topBorderHeight</code> or <code>leftBorderWidth</code> is less than zero</li> <li><code>dstRoiSize.width &lt; srcRoiSize.width + leftBorderWidth</code></li> <li><code>dstRoiSize.height &lt; srcRoiSize.height + topBorderHeight</code></li> </ul>
ippStsStepErr	Indicates an error when <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

**Example**

The code example below shows how to use the `ippiCopyMirrorBorder_8u_C1R` function.

```
Ipp8u src[8*4] = { 1, 2, 3, 8, 8, 8, 8, 8,
 10, 9, 8, 8, 8, 8, 8,
 11, 12, 13, 8, 8, 8, 8,
 19, 18, 17, 8, 8, 8, 8 };
Ipp8u dst[10*8];
IppiSize srcRoi = { 3, 4 }; IppiSize dstRoi = { 10, 8 };
int topBorderHeight = 2;
```

```
int leftBorderWidth = 2;
ippiCopyMirrorBorder_8u_C1R(src, 8, srcRoi, dst, 10, dstRoi, topBorderHeight, leftBorderWidth);
```

**Result:**

```
source image:
 1 2 3 8 8 8 8 8
10 9 8 8 8 8 8 8
11 12 13 8 8 8 8 8
19 18 17 8 8 8 8 8

destination image:
13 12 11 12 13 12 11 12 13 12
 8 9 10 9 8 9 10 9 8 9
 3 2 1 2 3 2 1 2 3 2
 8 9 10 9 8 9 10 9 8 9
13 12 11 12 13 12 11 12 13 12
17 18 19 18 17 18 19 18 17 18
13 12 11 12 13 12 11 12 13 12
 8 9 10 9 8 9 10 9 8 9
```

## See Also

[Regions of Interest in Intel IPP](#)

## CopyReplicateBorder

*Copies pixels values between two images and adds the replicated border pixels.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int
topBorderHeight, int leftBorderWidth);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 2: In-place operation

```
IppStatusippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep,
IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR
----------	-----------	-----------	-----------	-----------

### Case 3: Not-in-place operation for platform-aware functions

```
IppStatusippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
```

Supported values for mod:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L
8u_AC4R_L	16u_AC4R_L	16s_AC4R_L	32s_AC4R_L	32f_AC4R_L

### Case 4: In-place operation for platform-aware functions

```
IppStatusippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

Supported values for mod:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L
8u_C3IR_L	16u_C3IR_L	16s_C3IR_L	32s_C3IR_L	32f_C3IR_L
8u_C4IR_L	16u_C4IR_L	16s_C4IR_L	32s_C4IR_L	32f_C4IR_L
8u_AC4IR_L	16u_AC4IR_L	16s_AC4IR_L	32s_AC4IR_L	32f_AC4IR_L

## Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in destination image.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and the destination image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.

*leftBorderWidth* Width of the left border in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Replicated Border. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

## Creating a Replicated Border

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

*topBorderHeight*=2  
*leftBorderWidth*=2

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with a zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Example

The code example below shows how to use the `ippiCopyReplicateBorder_8u_C1R` function.

```
Ipp8u src[8*4] = {5, 4, 3, 4, 5, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  5, 4, 3, 4, 5, 8, 8, 8};

Ipp8u dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight = 2;
int leftborderWidth = 2;

ippiCopyReplicateBorder_8u_C1R(src, 8, srcRoi, dst, 9, dstRoi, topBorderHeight, leftBorderWidth);

Results
source image:
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8

destination image:
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
3 3 3 2 1 2 3 3 3
3 3 3 2 1 2 3 3 3
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
```

## CopyWrapBorder

---

*Copies pixels values between two images and adds the border pixels.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

```
IppStatusippiCopyWrapBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

#### Case 2: In-place operation

```
IppStatusippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

```
IppStatusippiCopyWrapBorder_32f_C1IR(const Ipp32f* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

### Case 3: Not-in-place operation with platform-aware functions

```
IppStatusippiCopyWrapBorder_32s_C1R_L(const Ipp32s* pSrc, IppSizeL srcStep, IppSizeL
srcRoiSize, Ipp32s* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppSizeL
topBorderHeight, IppSizeL leftBorderWidth);

IppStatusippiCopyWrapBorder_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, IppSizeL
srcRoiSize, Ipp32f* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppSizeL
topBorderHeight, IppSizeL leftBorderWidth);
```

### Case 4: In-place operation with platform-aware functions

```
IppStatusippiCopyWrapBorder_32s_C1IR_L(const Ipp32s* pSrc, IppSizeL srcDstStep,
IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);

IppStatusippiCopyWrapBorder_32f_C1IR_L(const Ipp32f* pSrc, IppSizeL srcDstStep,
IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

## Include Files

ippi.h

Flavors with the `_L` suffix: ippi\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place flavor.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Border of Pixels by ippiCopyWrapBorder Function. Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters `topBorderHeight` and `leftBorderWidth` specify the position of the first pixel of the source ROI in the destination image ROI.

### Creating a Border of Pixels by `ippiCopyWrapBorder` Function

14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6

`topBorderHeight=2`  
`leftBorderWidth=2`

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the `topBorderHeight` (`leftBorderWidth`) parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with a zero or negative value, or <code>topBorderHeight</code> or <code>leftBorderWidth</code> is less than zero, or <code>dstRoiSize.width &lt; srcRoiSize.width + leftBorderWidth</code> , or <code>dstRoiSize.height &lt; srcRoiSize.height + topBorderHeight</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating point images, or by 2 for short integer images.

### Example

The code example below shows how to use the `ippiCopyWrapBorder_32s_C1R` function.

```
Ipp32s src[8*4] = {
    5, 4, 3, 4, 5, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    5, 4, 3, 4, 5, 8, 8, 8
```

```

};

Ipp32s dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight = 2;
int leftborderWidth = 2;

ippiCopyWrapBorder_32s_C1R (src, 8*sizeof(Ipp32s), srcRoi, dst,
                           9*sizeof(Ipp32s), dstRoi, topBorderHeight, leftBorderWidth);

```

**Results****source image:**

```

5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8

```

**destination image:**

```

2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2

```

## CopySubpix

*Copies pixel values between two images with subpixel precision.*

### Syntax

#### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatusippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u32f_C1R	16u32f_C1R	

#### Case 2: Copying with conversion to integer data

```
IppStatusippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy, int scaleFactor);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dx</i>	Fractional part of the <i>x</i> -coordinate in the source image.
<i>dy</i>	Fractional part of the <i>y</i> -coordinate in the source image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the pixel value of the destination image using linear interpolation (see [Linear Interpolation](#) in Appendix B) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{j+dx,i+dy}$$

where  $i = 0, \dots, roiSize.height - 1$ ,  $j = 0, \dots, roiSize.width - 1$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of <i>srcStep</i> or <i>dstStep</i> is not divisible by 4 for floating point images, or by 2 for short integer images.

## CopySubpixIntersect

*Copies pixel values of the intersection with specified window with subpixel precision.*

---

## Syntax

### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatusippiCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u32f_C1R	16u32f_C1R	

### Case 2: Copying with conversion to integer data

```
IppStatusippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiPoint_32f
point, IppiPoint* pMin, IppiPoint* pMax, int scaleFactor);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>point</i>	Center point of the window.
<i>pMin</i>	Pointer to coordinates of the top left pixel of the intersection in the destination image.
<i>pMax</i>	Pointer to coordinates of the bottom right pixel of the intersection in the destination image.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

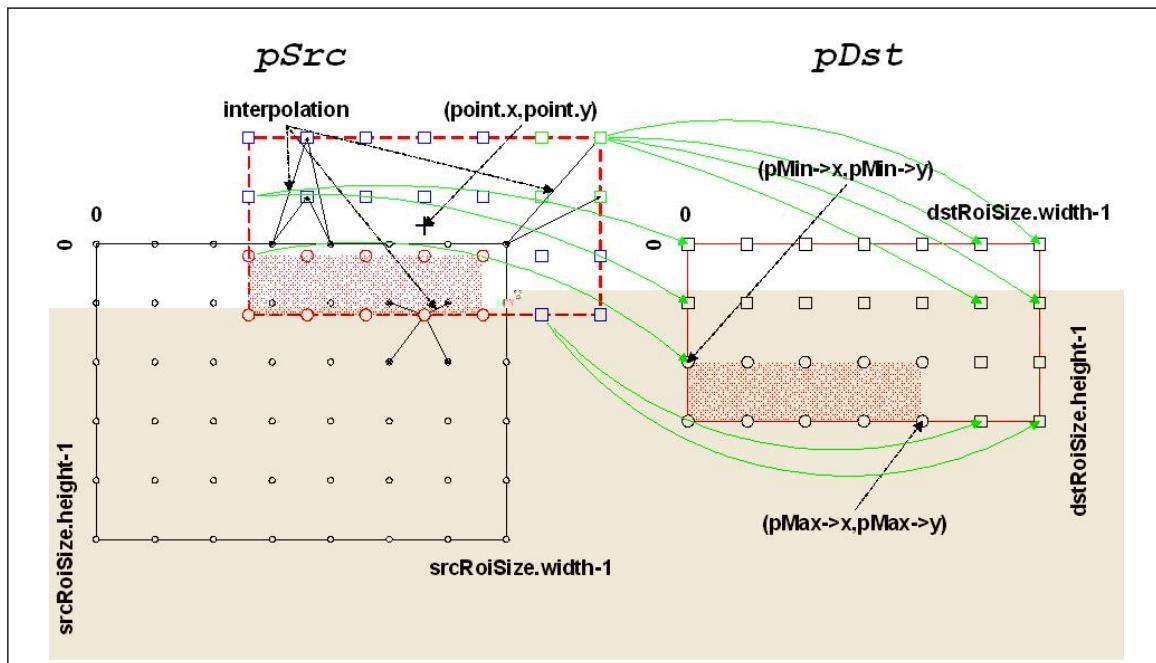
This function determines the intersection of the source image and the window of size *dstRoiSize* centered in point *point*. The corresponding pixels of the destination image are calculated using linear interpolation (see [Linear Interpolation](#) in Appendix B) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{Xsubpix(j), Ysubpix(i)}$$

where  $Xsubpix(j) = \min(\max(point.x + j - 0.5 * (dstRoiSize.width - 1), 0), srcRoiSize.width - 1)$ ,  $srcRoiSize.height - 1$ ,  
 $Ysubpix(i) = \min(\max(point.y + i - 0.5 * (dstRoiSize.height - 1), 0), srcRoiSize.height - 1)$ ,  
 $i = 0, \dots dstRoiSize.height - 1$ ,  $j = 0, \dots dstRoiSize.width - 1$ .

Minimal values of  $j$  and  $i$  (coordinates of the top left calculated destination pixel) are assigned to  $pMin.x$  and  $pMin.y$ , maximal values (coordinates of the top left calculated destination pixel) - to  $pMax.x$  and  $pMax.y$ .  
(See Figure Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function.)

### Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function



The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the `topBorderHeight` (`leftBorderWidth`) parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>srcRoiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>dstRoiSize.width * &lt;pixelSize&gt;</code> .

### Example

The code example below shows how to use the function `ippiCopySubpixIntersect_8u_C1R`.

```
Ipp8u src[8*6] = {
    7, 7, 6, 6, 6, 6, 7, 7,
    6, 5, 5, 5, 5, 5, 5, 6,
```

```

6, 5, 4, 3, 3, 4, 5, 6,
6, 5, 4, 3, 3, 4, 5, 6,
6, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 6, 6, 6, 6
};

Ipp8u dst[7*4];
IppiSize srcRoi = { 8, 6 };
IppiSize dstRoi = { 7, 4 };
IppiPoint_32f point = { 4, 1 };
IppiPoint min;
IppiPoint max;

ippiCopySubpixIntersect_8u_C1R (src, 8, srcRoi, dst, 7, dstRoi, point, &min, &max );

Results
source image:
7 7 6 6 6 6 7 7
6 5 5 5 5 5 5 6
6 5 4 3 3 4 5 6
6 5 4 3 3 4 5 6
6 5 5 5 5 5 5 6
6 6 6 6 6 6 6 6

destination image:
7 6 6 6 6 7 7
6 6 6 6 6 6 7
5 5 4 4 5 5 6
5 4 3 3 4 5 6

min = { 0, 1 }
max = { 5, 3 }

```

## Dup

*Copies a gray scale image to all channels of the color image.*

### Syntax

```

IppStatusippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiDup_8u_C1C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies a one-channel (gray scale) image *pSrc* to each channel of the multi-channel image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>RoiSize</i> has a field with a zero or negative value.

## Transpose

---

*Transpose* a source image.

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

#### Case 2: In-place operation

```
IppStatusippiTranspose_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination ROI for in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.

## Description

This function operates with ROI.

This function transposes the source image *pSrc* (*pSrcDst* for in-place flavors) and stores the result in *pDst* (*pSrcDst*). The destination image is obtained from the source image by transforming the columns to the rows:  $pDst(x,y) = pSrc(y,x)$

The parameter *roiSize* is specified for the source image. The value of the *roiSize.width* parameter for the destination image is equal to *roiSize.height* for the source image, and *roiSize.height* for the destination image is equal to *roiSize.width* for the source image.

### NOTE

For in-place operations, *roiSize.width* must be equal to *roiSize.height*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error when any of the specified pointers is NULL, with the exception of second mode in Case 4.
<i>ippStsSizeErr</i>	Indicates an error when: <ul style="list-style-type: none"> <li>• <i>roiSize</i> has a field with a zero or negative value</li> <li>• <i>roiSize.width</i> is not equal to <i>roiSize.height</i> for in-place flavors</li> </ul>

## Example

The code example below shows how to use the `ippiTranspose_8u_C1R` function.

```
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
IppiSize srcRoi = { 4, 4 };

ippiTranspose_8u_C1R ( src, 8, dst, 4, srcRoi );
```

**Result:**

```
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8      src
1 2 3 4 8 8 8 8

1 1 1 1
2 2 2 2
3 3 3 3      dst
4 4 4 4
```

## See Also

[Regions of Interest in Intel IPP](#)

## SwapChannels

---

*Copies channels of the source image to the destination image.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatusippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int dstOrder[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------	---------

#### Case 2: In-place operation

```
IppStatusippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, const int dstOrder[3]);
```

```
IppStatusippiSwapChannels_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize,
const int dstOrder[4]);
```

### Case 3: Operation with converting 3-channel image to the 4-channel image

```
IppStatusippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int dstOrder[4], Ipp<datatype> val);
```

Supported values for mod:

8u_C3C4R	16u_C3C4R	16s_C3C4R	32s_C3C4R	32f_C3C4R
----------	-----------	-----------	-----------	-----------

### Case 4: Operation with converting 4-channel image to the 3-channel image

```
IppStatusippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

Supported values for mod:

8u_C4C3R	16u_C4C3R	16s_C4C3R	32s_C4C3R	32f_C4C3R
----------	-----------	-----------	-----------	-----------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dstOrder</i>	Order of channels in the destination image.
<i>val</i>	Constant value.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the data from specified channels of the source image ROI *pSrc* to the specified channels of the destination image ROI *pDst*.

The first channel in the destination image is determined by the first component of *dstOrder*. Its value lies in the range [0..2] for a 3-channel image, and [0..3] for a 4-channel image, and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if the sequence of channels in the source 3-channel image is A, B, C, and *dstOrder*[0]=2, *dstOrder*[1]=0, *dstOrder*[2]=1, then the order of channels in the 3-channel destination image is C, A, B. Some or all components of *dstOrder* may have the same values. It means that data from a certain channel of the source image may be copied to several channels of the destination image.

Some functions flavors convert a 3-channel source image to the 4-channel destination image (see Case 3). In this case an additional channel contains data from any specified source channel, or its pixel values are set to the specified constant value *val* (corresponding component *dstOrder*[n] should be set to 3), or its pixel values are not set at all (corresponding component *dstOrder*[n] should be set to an arbitrary value greater than 3). For example, the sequence of channels in the source 3-channel image is A, B, C, if *dstOrder*[0]=1, *dstOrder*[1]=0, *dstOrder*[2]=1, *dstOrder*[3]=2, then the order of channels in the 4-channel destination image will be B, A, B, C; if *dstOrder*[0]=4, *dstOrder*[1]=0, *dstOrder*[2]=1, *dstOrder*[3]=2, then the order of channels in the 4-channel destination image will be D, A, B, C, where D is a channel whose pixel values are not set.

The function flavors that support image with the alpha channel do not perform operation on it.

This function supports negative step values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero value.

## Example

The code example below shows how to use the function `ippiSwapChannels_8u_C3R`.

```
Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                     0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                     0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255 };
Ipp8u dst[12*3];
IppiSize roiSize = { 4, 3 };
int order[3] = { 2, 1, 0 }

ippiSwapChannels_8u_C3R ( src, 12, dst, 12, roiSize, order );
```

Result:

```
src      [rgb]
255 0 0 255 0 0 255 0 0 255 0 0
0 255 0 0 255 0 0 255 0 0 255 0
0 0 255 0 0 255 0 0 255 0 0 255

dst      [bgr]
0 0 255 0 0 255 0 0 255 0 0 255
```

```
0 255 0 0 255 0 0 255 0 0 255 0
255 0 0 255 0 0 255 0 0 255 0 0
```

## AddRandUniform

*Generates random samples with uniform distribution and adds them to an image data.*

### Syntax

```
IppStatusippiAddRandUniform_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, Ippisize  
roiSize, Ipp<datatype> low, Ipp<datatype> high, unsigned int* pSeed);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<code>pSrcDst</code>	Pointer to the source and destination image ROI.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>low</code>	The lower bound for the range of uniformly distributed values.
<code>high</code>	The upper bound for the range of uniformly distributed values.
<code>pSeed</code>	The initial seed value for the pseudo-random number generator.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function generates samples with uniform distribution over the range [`low`, `high`] and adds them to a source image pointed to by `pSrcDst`.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image that contains pure noise with uniform distribution, call `ippiAddRandUniform` using a source image with zero data as input.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below shows data conversion without scaling.

```
IppStatus randUniform( void ) {
    unsigned int seed = 123456;
    Ipp8u img[2048], mn, mx;
    IppiSize roi={2048,1};
    Ipp64f mean;
    IppStatus st;
   ippiSet_8u_C1R( 0, img, 2048, roi );
    st =ippiAddRandUniform_8u_C1IR(img, 2048, roi, 0, 255, &seed);
   ippiMean_8u_C1R( img, 2048, roi, &mean );
   ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
    printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
    return st;
}
```

## AddRandGauss

*Generates random samples with Gaussian distribution and adds them to an image data.*

### Syntax

```
IppStatusippiAddRandGauss_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize  
roiSize, Ipp<datatype> mean, Ipp<datatype> stDev, unsigned int* pSeed);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>mean</i>	The mean of the Gaussian distribution.
<i>stDev</i>	The standard deviation of the Gaussian distribution.
<i>pSeed</i>	The initial seed value for the pseudo-random number generator.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function generates samples with Gaussian distribution that have the mean value *mean* and standard deviation *stDev* and adds them to a source image ROI pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, call `ippiAddRandGauss` using a source image with zero data as input.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> or <i>pSeed</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.

## ImageJaehne

---

*Creates Jaehne test image.*

---

### Syntax

```
IppStatusippiImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the destination image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [[Jae95](#)]).

The destination image pixel values are computed according to the following formula:

$$\text{Dst}(x, y) = A * \sin(0.5 * \text{IPP\_PI} * (x_2^2 + y_2^2) / \text{roiSize.height}),$$

where  $x, y$  are pixel coordinates varying in the range

$$0 \leq x \leq \text{roiSize.width}-1, 0 \leq y \leq \text{roiSize.height}-1;$$

`IPP_PI` is the library constant that stands for  $\pi$  value.

$$x_2 = (x - \text{roiSize.width} + 1) / 2.0,$$

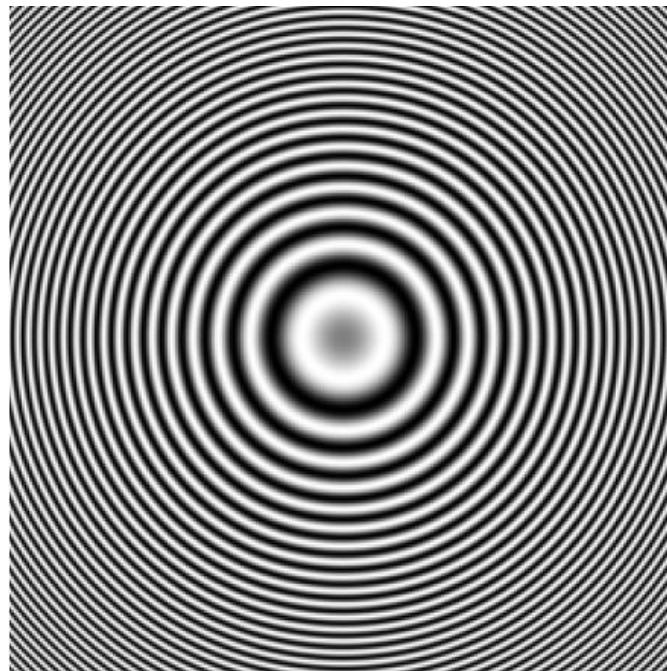
$$y_2 = (y - \text{roiSize.height} + 1) / 2.0,$$

$A$  is the constant value that depends upon the image type being created.

For the `32f` floating point data, the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).

Figure Example of a Generated Jaehne's Test Image illustrates an example of a test image generated by the `ippiImageJaehne` function.

### Example of a Generated Jaehne's Test Image



These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [Jae95].

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value, or if <code>dstStep</code> is less than or equal to zero.

## ImageRamp

Creates a test image that has an intensity ramp.

### Syntax

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, float offset, float slope, IppiAxis axis);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>

`8u_AC4R`      `16u_AC4R`      `16s_AC4R`      `32f_AC4R`

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the destination image ROI in pixels.
<code>offset</code>	Offset value.
<code>slope</code>	Slope coefficient.
<code>axis</code>	Specifies the direction of the image intensity ramp; can be one of the following:  <code>ippAxsHorizontal</code> for the ramp in <i>X</i> -direction, <code>ippAxsVertical</code> for the ramp in <i>Y</i> -direction, <code>ippAxsBoth</code> for the ramp in both <i>X</i> and <i>Y</i> -directions.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.

The destination image pixel values are computed according to one of the following formulas:

```
dst(x,y) = offset + slope * x, if axis = ippAxsHorizontal,
dst(x,y) = offset + slope * y, if axis = ippAxsVertical,
dst(x,y) = offset + slope * x * y, if axis = ippAxsBoth,
```

where *x*, *y* are pixel coordinates varying in the range

$0 \leq x \leq \text{roiSize.width}-1$ ,  $0 \leq y \leq \text{roiSize.height}-1$ ;

Note that linear transform coefficients `offset` and `slope` have floating-point values for all function flavors.

The computed pixel values that exceed the image data range are saturated to the respective data-range limits.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pDst</code> pointer is <code>NULL</code> .

---

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if <i>dstStep</i> is less than or equal to zero.
---------------	---

## Example

The code example below illustrates how to use the `ippiImageRamp` function.

```
IppStatus ramp( void ){
    Ipp8u dst[8*4];
    IppiSize roiSize = { 8, 4 };
    return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7, ippAxsHorizontal );
}
```

The destination image contains the following data:

```
00 25 49 6E 92 B7 DB FF
```

## SampleLine

*Puts a raster line into buffer.*

### Syntax

```
IppStatus ippiSampleLine_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, Ipp<datatype>* pDst, IppiPoint pt1, IppiPoint pt2);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source raster image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the raster image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pDst</i>	Pointer to the destination buffer. The buffer is to store at least $\max( pt2.x - pt1.x  + 1,  pt2.y - pt1.y  + 1)$ points.
<i>pt1</i>	Starting point of the line.

*pt2*

Ending point of the line.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and puts the resulting pixels into the destination buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> or <code>roiSize.height</code> is less than or equal to zero.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when the step for the floating-point image cannot be divided by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error when any of the line ending points is outside the image.

## Example

The code example below shows how to use the function `ippiSampleLine_8u_C1R` .

```
void func_sampleline()
{
    Ipp8u pSrc[5*4] = { 0, 1, 2, 3, 4,
                        5, 6, 7, 8, 9,
                        0, 9, 8, 7, 6,
                        5, 4, 3, 2, 1 };
    IppiSize roiSize = {5, 4};
    IppiPoint pt1 = {1, 1};
    IppiPoint pt2 = {2, 3};
    Ipp8u pDst[3];
    int srcStep = 5;

   ippiSampleLine_8u_C1R( pSrc, srcStep, roiSize, pDst, pt1, pt2 );

    printf("%Result: d, %d, %d\n", pDst[0], pDst[1], pDst[2] ); // << this wrong line
    printf("%Result: %d, %d, %d\n", pDst[0], pDst[1], pDst[2] ); // this is correct line
}
```

Result: 6, 9, 3

## ZigzagFwd8x8

---

*Converts a conventional order to the zigzag order.*

### Syntax

```
IppStatusippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

*pSrc* Pointer to the source data.

*pDst* Pointer to the destination data.

## Description

This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag sequence.

Figure Zigzag Sequence specifies the resulting zigzag sequence.

### Zigzag Sequence

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

## Return Values

ippStsNoErr Indicates no error.

ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL.

## Example

The code example below shows how to use the `ippiZigzagFwd8x8_16s_C1` function.

```
Ipp16s src[8*8] = {
    0, 1, 5, 7, 9, 2, 4, 1,
    5, 4, 8, 6, 3, 8, 0, 3,
    6, 2, 6, 8, 1, 4, 2, 8,
    4, 3, 2, 9, 3, 0, 6, 6,
    7, 7, 3, 0, 4, 1, 0, 9,
    5, 1, 9, 2, 5, 7, 1, 7,
    0, 3, 5, 0, 7, 5, 9, 8,
    2, 9, 1, 4, 6, 8, 2, 3
};
Ipp16s dst[8*8];
```

```
ippiZigzagFwd8x8_16s_C1 ( src, dst );
```

Result:

0	1	5	7	9	2	4	1
5	4	8	6	3	8	0	3
6	2	6	8	1	4	2	8
4	3	2	9	3	0	6	6
7	7	3	0	4	1	0	9
5	1	9	2	5	7	1	7
0	3	5	0	7	5	9	8
2	9	1	4	6	8	2	3

src //conventional order

0	1	5	6	4	5	7	8
2	4	7	3	6	6	9	2
3	8	2	7	5	0	1	3
9	1	8	4	1	0	4	3
0	9	3	2	9	5	2	4
0	2	3	8	6	1	5	0
1	4	7	7	0	6	9	1
5	6	8	9	7	8	2	3

dst //zigzag order

# ZigzagInv8x8

*Converts a zigzag order to the conventional order.*

## Syntax

```
IppStatusippiZiqzaqInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippym.lib, ipps.lib

## Parameters

*pSrc* Pointer to the source data.

*pDst* Pointer to the destination data.

## Description

This function rearranges data in an 8x8 block from a zigzag sequence to the conventional order (left-to-right, top-to-bottom).

Figure Zigzag Sequence specifies the resulting zigzag sequence.

## Return Values

`ippStsNoErr` Indicates no error.

ippStsNullPtrErr

Indicates an error when any of the specified pointers is `NULL`.

# Image Arithmetic and Logical Operations

5

This chapter describes functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

An additional suffix `c`, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.

**NOTE**

Most arithmetic and logical functions support data with 1-, 3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

---

## Arithmetic Operations

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of an absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

### Add

*Adds pixel values of two images.*

---

### Syntax

#### Case 1: Not-in-place operation on integer or complex data

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*  
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int  
scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

**Case 2: Not-in-place operation on floating point or complex data**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R  
32f\_C3R  
32f\_C4R

```
IppStatusippiAdd_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 3: In-place operation on integer or complex data**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

**Case 4: In-place operation on floating point or complex data**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR  
32f\_C3IR  
32f\_AC4IR  
32f\_C4IR

**Case 5: In-place operation using a floating point accumulator image**

```
IppStatusippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IR 16u32f\_C1IR

**Case 6: Masked in-place operation using a floating point accumulator image**

```
IppStatusippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IMR 16u32f\_C1IMR 32f\_C1IMR

**Case 7: Not-in-place operation on integer data with platform-aware functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_L 16u\_C1RSfs\_L 16s\_C1RSfs\_L  
 8u\_C3RSfs\_L 16u\_C3RSfs\_L 16s\_C3RSfs\_L  
 8u\_C4RSfs\_L 16u\_C4RSfs\_L 16s\_C4RSfs\_L  
 8u\_AC4RSfs\_L 16u\_AC4RSfs\_L 16s\_AC4RSfs\_L

**Case 8: Not-in-place operation on floating point data with platform-aware functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_L  
 32f\_C3R\_L  
 32f\_C4R\_L  
 32f\_AC4R\_L

**Case 9: In-place operation on integer data with platform-aware functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_L 16u\_C1IRSfs\_L 16s\_C1IRSfs\_L  
 8u\_C3IRSfs\_L 16u\_C3IRSfs\_L 16s\_C3IRSfs\_L  
 8u\_C4IRSfs\_L 16u\_C4IRSfs\_L 16s\_C4IRSfs\_L  
 8u\_AC4IRSfs\_L 16u\_AC4IRSfs\_L 16s\_AC4IRSfs\_L

**Case 10: In-place operation on floating point data with platform-aware functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

Supported values for mod:

32f\_C1IR\_L  
32f\_C3IR\_L  
32f\_C4IR\_L  
32f\_AC4IR\_L

**Case 11: Not-in-place operation on integer data with threading layer (TL) functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_LT 16u\_C1RSfs\_LT 16s\_C1RSfs\_LT  
8u\_C3RSfs\_LT 16u\_C3RSfs\_LT 16s\_C3RSfs\_LT  
8u\_C4RSfs\_LT 16u\_C4RSfs\_LT 16s\_C4RSfs\_LT  
8u\_AC4RSfs\_LT 16u\_AC4RSfs\_LT 16s\_AC4RSfs\_LT

**Case 12: Not-in-place operation on floating point data with TL functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_LT  
32f\_C3R\_LT  
32f\_C4R\_LT  
32f\_AC4R\_LT

**Case 13: In-place operation on integer data with TL functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT 16u\_C1IRSfs\_LT 16s\_C1IRSfs\_LT  
8u\_C3IRSfs\_LT 16u\_C3IRSfs\_LT 16s\_C3IRSfs\_LT  
8u\_C4IRSfs\_LT 16u\_C4IRSfs\_LT 16s\_C4IRSfs\_LT

8u\_AC4IRSfs\_LT 16u\_AC4IRSfs\_LT 16s\_AC4IRSfs\_LT

#### **Case 14: In-place operation on floating point data with TL functions**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

Supported values for mod:

32f\_C1IR\_LT

32f\_C3IR\_LT

32f\_C4IR\_LT

32f\_AC4IR\_LT

#### **Case 15: In-place operation on integer data with TL functions based on classic API**

```
IppStatusippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSize srcStep, Ipp<datatype>* pSrcDst, IppSize srcDstStep, IppSize roiSize, int scaleFactor);
```

Supported values for mod:

16s\_C1IRSfs\_T 32s\_C1IRSfs\_T

16s\_C3IRSfs\_T

16s\_C4IRSfs\_T

### **Include Files**

ippcv.h

ippi.h

ippi\_1.h

ippi\_tl.h

### **Domain Dependencies**

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi64x.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>srcStep, src1Step, src2Step</i>	Distance, in bytes, between the starting points of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the first source image for the in-place operation.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>pMask</i>	Pointer to the mask image ROI for the masked operation.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the mask image for the masked operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI.

This function adds corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*. For complex data, the function processes both real and imaginary parts of pixel values. Some function flavors add 8u, 8s, 16u, or 32f source image pixel values to a floating point accumulator image in-place. Addition of pixel values in case of a masked operation is performed only if the respective mask value is non-zero; otherwise, the accumulator pixel value remains unchanged.

---

### NOTE

For the functions that operate on complex data, step values must be positive. For the functions that use an accumulator image, step values must be no less than *roiSize.width\*pixelSize*.

---

Functions with AC4 descriptor do not process alpha channels.

Function flavors described in Case 5 and Case 6 are declared in the `ippcv.h`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when <i>roiSize</i> has a field with zero or negative value.

`ippStsStepErr`

Indicates an error condition in the following cases:

- For functions that operate on complex data, if any of the specified step values is zero or negative.
- For functions using an accumulator image, if any of the specified step values is less than `roiSize.width * <pixelSize>`.

`ippStsNotEvenStepErr`

Indicates an error condition when one of step values for floating-point images cannot be divided by 4.

## Example

The code example below shows how to use the function `ippiAdd_8u_C1RSfs`.

```
Ipp8u src1[8*4] = {8, 4, 2, 1, 0, 0, 0, 0,
                    8, 4, 2, 1, 0, 0, 0, 0,
                    8, 4, 2, 1, 0, 0, 0, 0,
                    8, 4, 2, 1, 0, 0, 0, 0};
Ipp8u src2[8*4] = {4, 3, 2, 1, 0, 0, 0, 0,
                    4, 3, 2, 1, 0, 0, 0, 0,
                    4, 3, 2, 1, 0, 0, 0, 0,
                    4, 3, 2, 1, 0, 0, 0, 0};
Ipp8u dst[8*4];
IppiSize srcRoi = { 4, 4 };
Int scaleFactor = 1; // later examples for 2 and -2 values

ippiAdd_8u_C1RSfs (src1, 8, src2, 8, dst, 4, srcRoi, scaleFactor );

Result:
      src1           src2
8 4 2 1 0 0 0 0   4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0   4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0   4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0   4 3 2 1 0 0 0 0

dst >> scaleFactor = 1      scaleFactor = 2      scaleFactor = -2
      6 4 2 1           3 2 1 0           48 28 16 8
      6 4 2 1           3 2 1 0           48 28 16 8
      6 4 2 1           3 2 1 0           48 28 16 8
      6 4 2 1           3 2 1 0           48 28 16 8
```

## See Also

[Regions of Interest in Intel IPP](#)

## AddC

Adds a constant to pixel values of an image.

## Syntax

### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
                           Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
-----------	------------	------------

**Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs

16u\_C3RSfs

16s\_C3RSfs

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs

16u\_C4RSfs

16s\_C4RSfs

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_AC4RSfs

16u\_AC4RSfs

16s\_AC4RSfs

**Case 3: Not-in-place operation on one-channel floating-point or complex data**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C3R

```
IppStatusippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_AC4R

**Case 5: In-place operation on one-channel integer or complex data**

```
IppStatusippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,  
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
------------	-------------	-------------

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs
------------	-------------	-------------

**Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatusippiAddC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize  
roiSize);
```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C3IR	32f_AC4IR	
----------	-----------	--

```
IppStatusippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,  
IppiSize roiSize);
```

**Case 9: Not-in-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>  
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
-------------	--------------	--------------

**Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L
-------------	--------------	--------------

**Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiAddC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions**

```
IppStatusippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f_C3R_L
32f_AC4R_L

```
IppStatusippiAddC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 13: In-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
--------------	---------------	---------------

**Case 14: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C4IRSfs_L      16u_C4IRSfs_L      16s_C4IRSfs_L
```

### **Case 15: In-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiAddC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);
```

### **Case 16: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);
```

Supported values for mod:

```
32f_C3IR_L  
32f_AC4IR_L
```

```
IppStatusippiAddC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);
```

### **Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs_LT      16u_C1RSfs_LT      16s_C1RSfs_LT
```

### **Case 18: Not-in-place operation on multi-channel integer data with TL functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C3RSfs_LT      16u_C3RSfs_LT      16s_C3RSfs_LT  
8u_AC4RSfs_LT     16u_AC4RSfs_LT     16s_AC4RSfs_LT
```

```
IppStatusippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C4RSfs_LT      16u_C4RSfs_LT      16s_C4RSfs_LT
```

### **Case 19: Not-in-place operation on one-channel floating point data with TL functions**

```
IppStatusippiAddC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

**Case 20: Not-in-place operation on multi-channel floating point data with TL functions**

```
IppStatusippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],  
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_LT

32f\_AC4R\_LT

```
IppStatusippiAddC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f  
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 21: In-place operation on one-channel integer data with TL functions**

```
IppStatusippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT    16u\_C1IRSfs\_LT    16s\_C1IRSfs\_LT

**Case 22: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs\_LT    16u\_C3IRSfs\_LT    16s\_C3IRSfs\_LT

8u\_AC4IRSfs\_LT    16u\_AC4IRSfs\_LT    16s\_AC4IRSfs\_LT

```
IppStatusippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_LT    16u\_C4IRSfs\_LT    16s\_C4IRSfs\_LT

**Case 23: In-place operation on one-channel floating point data with TL functions**

```
IppStatusippiAddC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

**Case 24: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_LT

32f\_AC4IR\_LT

```
IppStatusippiAddC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize);
```

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_t1.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_t1.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_t1.lib, ippi\_t1.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the image intensity by adding *value* to image pixel values. For one-channel images, a positive *value* brightens the image (increases the intensity); a negative value darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector *value* are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

---

### NOTE

Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## AddSquare

*Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.*

### Syntax

#### Case 1: In-place operation

```
IppStatusippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IR

16u32f\_C1IR

32f\_C1IR

#### Case 2: Masked in-place operation

```
IppStatusippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f\_C1IMR

16u32f\_C1IMR

32f\_C1IMR

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.

<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds squared pixel values of the source image *pSrc* to floating-point pixel values of the accumulator image *pSrcDst* as follows:

$$pSrcDst(x, y) = pSrcDst(x, y) + pSrc(x, y)^2$$

Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is negative.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> , <i>maskStep</i> , or <i>srcDstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AddProduct

*Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.*

## Syntax

### Case 1: In-place operation

```
IppStatusippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for *mod*:

`8u32f_C1IR`

`16u32f_C1IR`

`32f_C1IR`

### Case 2: Masked in-place operation

```
IppStatusippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

`8u32f_C1IMR`

`16u32f_C1IMR`

`32f_C1IMR`

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds the product of pixel values of two source images *pSrc1* and *pSrc2* to floating-point pixel values of the accumulator image *pSrcDst* as given by:

$$pSrcDst(x, y) = pSrcDst(x, y) + pSrc1(x, y) * pSrc2(x, y)$$

The products of pixel values in case of a masked operation are added only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when any of the specified pointers is <i>null</i> .
<i>ippStsSizeErr</i>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is negative.
<i>ippStsStepErr</i>	Indicates an error if <i>src1Step</i> , <i>src2Step</i> , <i>maskStep</i> , or <i>srcDstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<i>ippStsNotEvenStepErr</i>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AddWeighted

Adds weighted pixel values of a source image to floating-point pixel values of an accumulator image.

## Syntax

### Case 1: In-place operation

```
IppStatusippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for mod:

8u32f\_C1IR

16u32f\_C1IR

32f\_C1IR

### Case 2: Masked in-place operation

```
IppStatusippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for mod:

8u32f\_C1IMR

16u32f\_C1IMR

32f\_C1IMR

### Case 3: Not-in-place operation

```
IppStatusippiAddWeighted_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f alpha);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for the in-place operation.
<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the ROI in the source images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for the in-place operation.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.

*alpha*Weight *a* of the source image.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds pixel values of the source image *pSrc1* multiplied by a weight factor *alpha* to pixel values of the image *pSrc2* multiplied by (1-*alpha*) and stores result in the *pDst* as follows:

$$pDst(x, y) = pSrc1(x, y) * \alpha + pSrc2(x, y) * (1 - \alpha)$$

The in-place flavors of the function adds pixel values of the source image *pSrc* multiplied by a weight factor *alpha* to floating-point pixel values of the accumulator image *pSrcDst* multiplied by (1-*alpha*) as follows:

$$pSrcDst(x, y) = pSrcDst(x, y) * (1 - \alpha) + pSrc(x, y) * \alpha$$

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize.width</code> or <code>roiSize.height</code> is equal to 0 or negative.
<code>ippStsStepErr</code>	Indicates an error when one of the step values is equal to zero, or is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of step values for floating-point images cannot be divided by 4.

## Mul

*Multiples pixel values of two images.*

### Case 1: Not-in-place operation on integer or complex data

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>

**Case 2: Not-in-place operation on floating-point or complex data**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R	32f_AC4R
32f_C3R	32f_C4R

**Case 3: In-place operation on integer or complex data**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

**Case 4: In-place operation on floating-point or complex data**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

**Case 5: Not-in-place operation on integer data with platform-aware functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

**Case 6: Not-in-place operation on floating-point data with platform-aware functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_L  
32f\_C3R\_L  
32f\_C4R\_L  
32f\_AC4R\_L

**Case 7: In-place operation on integer data with platform-aware functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

**Case 8: In-place operation on floating-point data with platform-aware functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1IR\_L  
32f\_C3IR\_L  
32f\_AC4IR\_L  
32f\_C4IR\_L

**Case 9: Not-in-place operation on integer data with threading layer (TL) functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

8u\_C4RSfs\_LT

16u\_C4RSfs\_LT

16s\_C4RSfs\_LT

**Case 10: Not-in-place operation on floating-point data with TL functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_LT

32f\_C3R\_LT

32f\_C4R\_LT

32f\_AC4R\_LT

**Case 11: In-place operation on integer data with TL functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT

16u\_C1IRSfs\_LT

16s\_C1IRSfs\_LT

8u\_C3IRSfs\_LT

16u\_C3IRSfs\_LT

16s\_C3IRSfs\_LT

8u\_AC4IRSfs\_LT

16u\_AC4IRSfs\_LT

16s\_AC4IRSfs\_LT

8u\_C4IRSfs\_LT

16u\_C4IRSfs\_LT

16s\_C4IRSfs\_LT

**Case 12: In-place operation on floating-point data with TL functions**

```
IppStatusippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1IR\_LT

32f\_C3IR\_LT

32f\_AC4IR\_LT

32f\_C4IR\_LT

**Include Files**

ippi.h

Flavors with the \_LT suffix:ippi\_tl.h

Flavors with the \_L suffix:ippi\_l.h

**Domain Dependencies**

Flavors declared in ippi.h:

Headers:ippcore.h,ippvm.h,ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in `ippi_tl.h`:

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

## Parameters

<code>pSrc, pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>srcStep, src1Step, src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>scaleFactor</code>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

For complex data, the function processes both real and imaginary parts of pixel values.

---

### NOTE

Step values must be positive for functions that operate on complex data.

---

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## MulC

[Multiplies pixel values of an image by a constant.](#)

## Syntax

### **Case 1: Not-in-place operation on one-channel integer or complex data**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs

16u\_C1RSfs

16s\_C1RSfs

### **Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs

16u\_C3RSfs

16s\_C3RSfs

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs

16u\_C4RSfs

16s\_C4RSfs

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_AC4RSfs

16u\_AC4RSfs

16s\_AC4RSfs

### **Case 3: Not-in-place operation on one-channel floating-point or complex data**

```
IppStatusippiMulC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

### **Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatusippiMulC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_AC4R

```
IppStatusippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on one-channel integer or complex data**

```
IppStatusippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,  
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs

16u\_C1IRSfs

16s\_C1IRSfs

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs

16u\_C3IRSfs

16s\_C3IRSfs

8u\_AC4IRSfs

16u\_AC4IRSfs

16s\_AC4IRSfs

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs 16u\_C4IRSfs 16s\_C4IRSfs

**Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatusippiMulC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize  
roiSize);
```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C3IR

32f\_AC4IR

```
IppStatusippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,  
IppiSize roiSize);
```

**Case 9: Not-in-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>  
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_L

16u\_C1RSfs\_L

16s\_C1RSfs\_L

**Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L
-------------	--------------	--------------

**Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiMulC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions**

```
IppStatusippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f_C3R_L
32f_AC4R_L

```
IppStatusippiMulC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 13: In-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
--------------	---------------	---------------

**Case 14: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

---

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_L      16u\_C4IRSfs\_L      16s\_C4IRSfs\_L

#### **Case 15: In-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiMulC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

#### **Case 16: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_L

32f\_AC4IR\_L

```
IppStatusippiMulC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

#### **Case 17: Not-in-place operation on one-channel integer data with threading layer (TL)functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_LT      16u\_C1RSfs\_LT      16s\_C1RSfs\_LT

#### **Case 18: Not-in-place operation on multi-channel integer data with TL functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C3RSfs\_LT      16u\_C3RSfs\_LT      16s\_C3RSfs\_LT

8u\_AC4RSfs\_LT      16u\_AC4RSfs\_LT      16s\_AC4RSfs\_LT

```
IppStatusippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C4RSfs\_LT      16u\_C4RSfs\_LT      16s\_C4RSfs\_LT

#### **Case 19: Not-in-place operation on one-channel floating point data with TL functions**

```
IppStatusippiMulC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

**Case 20: Not-in-place operation on multi-channel floating point data with TL functions**

```
IppStatusippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],  
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_LT

32f\_AC4R\_LT

```
IppStatusippiMulC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f  
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 21: In-place operation on one-channel integer data with TL functions**

```
IppStatusippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT      16u\_C1IRSfs\_LT      16s\_C1IRSfs\_LT

**Case 22: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs\_LT      16u\_C3IRSfs\_LT      16s\_C3IRSfs\_LT

8u\_AC4IRSfs\_LT      16u\_AC4IRSfs\_LT      16s\_AC4IRSfs\_LT

```
IppStatusippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_LT      16u\_C4IRSfs\_LT      16s\_C4IRSfs\_LT

**Case 23: In-place operation on one-channel floating point data with TL functions**

```
IppStatusippiMulC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

**Case 24: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_LT

32f\_AC4IR\_LT

```
IppStatusippiMulC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize);
```

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_t1.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_t1.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_t1.lib, ippi\_t1.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>value</i>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see Regions of [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values of an image by a constant *value*. For multi-channel images, pixel channel values are multiplied by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values.

---

### NOTE

Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## MulC64f

*Multiplies pixel values of an image by a constant array.*

---

### Syntax

#### Not-in-place operations

```
IppStatusippiMulC64f_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[1],  
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[3],  
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[4],  
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16u_C1R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[1],  
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16u_C3R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[3],  
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16u_C4R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[4],  
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16s_C1R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[1],  
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16s_C3R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[3],  
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);  
  
IppStatusippiMulC64f_16s_C4R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[4],  
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode  
rndMode);
```

```
IppStatusippiMulC64f_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[1],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatusippiMulC64f_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[3],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatusippiMulC64f_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);
```

### In-place operations

```
IppStatusippiMulC64f_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16u_C1IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16u_C4IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatusippiMulC64f_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);
```

### Include Files

ippi.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

### Parameters

*pSrc*

Pointer to the source image.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operations.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operations.
<i>value</i>	Constant vector to add to image pixel values.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI, in pixels.
<i>hint</i>	Option to select the algorithmic implementation of the function, the following values are supported:  ippAlgHintAccurate    All output pixels are exact; accuracy takes precedence over performance. ippAlgHintFast, ippAlgHintNone    Function performance takes precedence over accuracy and some output pixels can differ by +1 from the exact result.
<i>rndMode</i>	Rounding mode, the following values are supported:  ippRndZero    Floating-point values are truncated to zero. ippRndNear    Floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer. ippRndFinancial    Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal to or greater than 0.5.

## Description

This function multiplies pixel values of the source image by the specified constant array and places the scaled results to the same image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width or height of the image is less than, or equal to zero.

## Example

To better understand usage of this function, refer to the `MulC64f.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## MulScale

*Multiples pixel values of two images and scales the products.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

#### Case 2: In-place operation

```
IppStatusippiMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

$$\text{dst\_pixel} = \text{src1\_pixel} * \text{src2\_pixel} / \text{max\_val},$$

where *src1\_pixel* and *src2\_pixel* are pixel values of the source buffers, *dst\_pixel* is the resultant pixel value, and *max\_val* is the maximum value of the pixel data range (see Table “Image Data Types and Ranges” for details). The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with AC4 descriptor do not process alpha channelss.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## MulCScale

*Multiples pixel values of an image by a constant and scales the products.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R      16u\_C1R

### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      16u\_C3R

8u\_AC4R    16u\_AC4R

```
IppStatusippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R    16u\_C4R

### Case 3: In-place operation on one-channel data

```
IppStatusippiMulCScale_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR    16u\_C1IR

### Case 4: In-place operation on multi-channel data

```
IppStatusippiMulCScale_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR    16u\_C3IR

8u\_AC4IR    16u\_AC4IR

```
IppStatusippiMulCScale_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR    16u\_C4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to multiply each pixel value in a source image (constant vector in case of 3- or four-channel images).
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values in the input buffer by a constant *value* and scales the products using the following formula:

$$\text{dst\_pixel} = \text{src\_pixel} * \text{value} / \text{max\_val},$$

where *src\_pixel* is a pixel values of the source buffer, *dst\_pixel* is the resultant pixel value, and *max\_val* is the maximum value of the pixel data range (see [Table "Image Data Types and Ranges"](#) for details).

The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.

Note that the functions with AC4 descriptor do not process alpha channelss.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Example

The code example below shows how to use the function `ippiMulCScale_8u_C1R`.

```
void func_mulcscale()
{
    IppiSize ROI = {8,4};
    IppiSize ROI2 = {5,4};
    Ipp8u src[8*4];
    Ipp8u dst[8*4];
    Ipp8u v = 100;

   ippiSet_8u_C1R(100,src,8,ROI);
   ippiSet_8u_C1R(0,dst,8,ROI);
   ippiMulCScale_8u_C1R(src,8,v,dst,8,ROI2);
}
```

**Result:**

src1	dst
100 100 100 100 100 100 100 100	39 39 39 39 39 0 0 0

100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	39	39	39	39	39	39	0	0	0
100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	39	39	39	39	39	39	0	0	0
100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	39	39	39	39	39	39	0	0	0

## Sub

*Subtracts pixel values of two images.*

### Syntax

#### Case 1: Not-in-place operation on integer or complex data

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

#### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_AC4R
32f_C4R

#### Case 3: In-place operation on integer or complex data

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

**Case 4: In-place operation on floating-point or complex data**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR  
32f\_C3IR  
32f\_AC4IR  
32f\_C4IR

**Case 5: Not-in-place operation on integer data with platform-aware functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

**Case 6: Not-in-place operation on floating-point data with platform-aware functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_L  
32f\_C3R\_L  
32f\_C4R\_L  
32f\_AC4R\_L

**Case 7: In-place operation on integer data with platform-aware functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst,
IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L
--------------	---------------	---------------

#### **Case 8: In-place operation on floating-point data with platform-aware functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

Supported values for mod:

32f_C1IR_L	32f_C3IR_L	32f_AC4IR_L
32f_C4IR_L		

#### **Case 9: Not-in-place operation on integer data with threading layer (TL) functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

#### **Case 10: Not-in-place operation on floating-point data with TL functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f_C1R_LT	32f_C3R_LT	32f_C4R_LT
32f_AC4R_LT		

#### **Case 11: In-place operation on integer data with TL functions**

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT

8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

### Case 12: In-place operation on floating-point data with TL functions

```
IppStatusippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

Supported values for mod:

```
32f_C1IR_LT  
32f_C3IR_LT  
32f_AC4IR_LT  
32f_C4IR_LT
```

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

### Parameters

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function subtracts pixel values of the source buffer *pSrc1* from the corresponding pixel values of the buffer *pSrc2* and places the result in the destination buffer *pDst*. For in-place operations, the values in *pSrc* are subtracted from the values in *pSrcDst* and the results are placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values.

### **NOTE**

Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channels.

### **Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## **SubC**

*Subtracts a constant from pixel values of an image.*

### **Syntax**

#### **Case 1: Not-in-place operation on one-channel integer or complex data**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs

16u\_C1RSfs

16s\_C1RSfs

#### **Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs

16u\_C3RSfs

16s\_C3RSfs

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
-----------	------------	------------

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
------------	-------------	-------------

### **Case 3: Not-in-place operation on one-channel floating-point or complex data**

```
IppStatusippiSubC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

### **Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatusippiSubC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_AC4R
----------

### **Case 5: In-place operation on one-channel integer or complex data**

```
IppStatusippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
------------	-------------	-------------

### **Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
------------	-------------	-------------

8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
-------------	--------------	--------------

---

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C4IRSfs    16u_C4IRSfs    16s_C4IRSfs
```

#### **Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatusippiSubC_32f_C1IR(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

#### **Case 8: In-place operation on multi-channel floating-point or complex data:**

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C3IR
32f_AC4IR
```

```
IppStatusippiSubC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

#### **Case 9: Not-in-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs_L    16u_C1RSfs_L    16s_C1RSfs_L
```

#### **Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

```
8u_C3RSfs_L    16u_C3RSfs_L    16s_C3RSfs_L
8u_AC4RSfs_L   16u_AC4RSfs_L   16s_AC4RSfs_L
```

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

```
8u_C4RSfs_L    16u_C4RSfs_L    16s_C4RSfs_L
```

#### **Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiSubC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions**

```
IppStatusippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],  
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_L

32f\_AC4R\_L

```
IppStatusippiSubC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f  
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 13: In-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_L      16u\_C1IRSfs\_L      16s\_C1IRSfs\_L

**Case 14: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs\_L      16u\_C3IRSfs\_L      16s\_C3IRSfs\_L

8u\_AC4IRSfs\_L      16u\_AC4IRSfs\_L      16s\_AC4IRSfs\_L

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_L      16u\_C4IRSfs\_L      16s\_C4IRSfs\_L

**Case 15: In-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiSubC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

**Case 16: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,  
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_L

32f\_AC4IR\_L

```
IppStatusippiSubC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL  
srcDstStep, IppiSizeL roiSize);
```

**Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_LT      16u\_C1RSfs\_LT      16s\_C1RSfs\_LT

**Case 18: Not-in-place operation on multi-channel integer data with TL functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs\_LT      16u\_C3RSfs\_LT      16s\_C3RSfs\_LT

8u\_AC4RSfs\_LT      16u\_AC4RSfs\_LT      16s\_AC4RSfs\_LT

```
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs\_LT      16u\_C4RSfs\_LT      16s\_C4RSfs\_LT

**Case 19: Not-in-place operation on one-channel floating point data with TL functions**

```
IppStatusippiSubC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 20: Not-in-place operation on multi-channel floating point data with TL functions**

```
IppStatusippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_LT

32f\_AC4R\_LT

```
IppStatusippiSubC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 21: In-place operation on one-channel integer data with TL functions**

```
IppStatusippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT      16u\_C1IRSfs\_LT      16s\_C1IRSfs\_LT

**Case 22: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT

```
IppStatusippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT
---------------	----------------	----------------

**Case 23: In-place operation on one-channel floating point data with TL functions**

```
IppStatusippiSubC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

**Case 24: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f_C3IR_LT
32f_AC4IR_LT

```
IppStatusippiSubC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

**Include Files**

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

**Domain Dependencies**

Flavors declared in ippi.h:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

---

<i>value</i>	The constant value to subtract from each pixel value in a source image (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by subtracting the constant *value* from pixel values of an image buffer. For multi-channel images, the components of a constant vector *value* are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

---

### NOTE

Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## Div

*Divides pixel values of an image by pixel values of another image.*

---

## Syntax

### Case 1: Not-in-place operation on integer or complex data

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_AC4R
32f_C4R

### Case 3: In-place operation on integer or complex data

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

### Case 4: In-place operation on floating-point or complex data

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR
32f_C3IR
32f_AC4IR

32f\_C4IR

#### **Case 5: Not-in-place operation on integer data with platform-aware functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

#### **Case 6: Not-in-place operation on floating-point data with platform-aware functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f_C1R_L
32f_C3R_L
32f_AC4R_L
32f_C4R_L

#### **Case 7: In-place operation on integer data with platform-aware functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

#### **Case 8: In-place operation on floating-point data with platform-aware functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f_C1IR_L
32f_C3IR_L

32f\_AC4IR\_L  
32f\_C4IR\_L

### **Case 9: Not-in-place operation on integer data with threading layer (TL) functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

### **Case 10: Not-in-place operation on floating-point data with TL functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1R\_LT  
32f\_C3R\_LT  
32f\_C4R\_LT  
32f\_AC4R\_LT

### **Case 11: In-place operation on integer data with TL functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

### **Case 12: In-place operation on floating-point data with TL functions**

```
IppStatusippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C1IR\_LT

---

```
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT
```

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>pSrc</code> , <code>pSrc1</code> , <code>pSrc2</code>	Pointers to the source images ROI.
<code>srcStep</code> , <code>src1Step</code> , <code>src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>scaleFactor</code>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function divides pixel values of the source buffer `pSrc2` by the corresponding pixel values of the buffer `pSrc1` and places the result in a destination buffer `pDst`. For in-place operations, the values in `pSrcDst` are divided by the values in `pSrc` and placed into `pSrcDst`. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by `scaleFactor` and rounded (not truncated). When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.

## Example

The code example below illustrates how the function `ippiDiv` can be used.

```
IppStatus div32f( void ) {
    Ipp32f a[4*3], b[4*3];
    IppiSize roi = {2,2};
    int i;
    for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
    returnippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
                           4*sizeof(Ipp32f), roi );
}
```

The destination image *b* contains

```
-1.#IND +1.000 +2.000 +3.000
+1.000 +1.000 +6.000 +7.000
+8.000 +9.000 +10.00 +11.00
```

Console output:

```
-- warning in div32f:
(6) Zero value(s) of divisor in the function Div
```

## Div\_Round

*Divides pixel values of an image by pixel values of another image with different rounding modes.*

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatusippiDiv_Round_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
IppRoundMode rndMode, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

### Case 2: In-place operation on integer data

```
IppStatusippiDiv_Round_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*  
pSrcDst, int srcDstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);
```

Supported values for mod:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.				
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.				
<i>pDst</i>	Pointer to the destination image ROI.				
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.				
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.				
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.				
<i>roiSize</i>	Size of the source and destination ROI in pixels.				
<i>roundMode</i>	Rounding mode, the following values are possible: <table border="0"> <tr> <td><i>ippRndZero</i></td> <td>specifies that floating-point values are truncated toward zero,</td> </tr> <tr> <td><i>ippRndNear</i></td> <td>specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,</td> </tr> </table>	<i>ippRndZero</i>	specifies that floating-point values are truncated toward zero,	<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,
<i>ippRndZero</i>	specifies that floating-point values are truncated toward zero,				
<i>ippRndNear</i>	specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,				

	ippRndFinancial	specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
scaleFactor		Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. The resulting values are scaled by *scaleFactor* and rounded using the rounding method specified by the parameter *roundMode*. When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A "[Handling of Special Cases](#)" for more information).

Note that the functions with the `AC4` descriptor do not process alpha channelss.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.
ippStsStsRoundModeNotSupported Err	Indicates an error condition if the <i>roundMode</i> has an illegal value.

## DivC

*Divides pixel values of an image by a constant.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1RSfs

16u\_C1RSfs

16s\_C1RSfs

**Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3RSfs

16u\_C3RSfs

16s\_C3RSfs

16s\_AC4RSfs

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4RSfs 16u\_C4RSfs 16s\_C4RSfs

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_AC4RSfs

16u\_AC4RSfs

**Case 3: Not-in-place operation on one-channel floating-point or complex data**

```
IppStatusippiDivC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatusippiDivC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_AC4R

```
IppStatusippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on one-channel integer or complex data**

```
IppStatusippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs

16u\_C1IRSfs

16s\_C1IRSfs

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs
------------	-------------	-------------

**Case 7: In-place operation on one-channel floating-point or complex data**

```
IppStatusippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR	32f_C3IR	32f_AC4IR
----------	----------	-----------

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C3IR	32f_C4IR	32f_AC4IR
----------	----------	-----------

```
IppStatusippiDivC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

**Case 9: Not-in-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
-------------	--------------	--------------

**Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
-------------	--------------	--------------

8u\_AC4RSfs\_L      16u\_AC4RSfs\_L      16s\_AC4RSfs\_L

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C4RSfs\_L      16u\_C4RSfs\_L      16s\_C4RSfs\_L

### **Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiDivC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

### **Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions**

```
IppStatusippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_L

32f\_AC4R\_L

```
IppStatusippiDivC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);
```

### **Case 13: In-place operation on one-channel integer data with platform-aware functions**

```
IppStatusippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_L      16u\_C1IRSfs\_L      16s\_C1IRSfs\_L

### **Case 14: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs\_L      16u\_C3IRSfs\_L      16s\_C3IRSfs\_L

8u\_AC4IRSfs\_L      16u\_AC4IRSfs\_L      16s\_AC4IRSfs\_L

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_L      16u\_C4IRSfs\_L      16s\_C4IRSfs\_L

**Case 15: In-place operation on one-channel floating point data with platform-aware functions**

```
IppStatusippiDivC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

**Case 16: In-place operation on multi-channel integer data with platform-aware functions**

```
IppStatusippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_L

32f\_AC4IR\_L

```
IppStatusippiDivC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

**Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1RSfs\_LT      16u\_C1RSfs\_LT      16s\_C1RSfs\_LT

**Case 18: Not-in-place operation on multi-channel integer data with TL functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C3RSfs\_LT      16u\_C3RSfs\_LT      16s\_C3RSfs\_LT

8u\_AC4RSfs\_LT      16u\_AC4RSfs\_LT      16s\_AC4RSfs\_LT

```
IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C4RSfs\_LT      16u\_C4RSfs\_LT      16s\_C4RSfs\_LT

**Case 19: Not-in-place operation on one-channel floating point data with TL functions**

```
IppStatusippiDivC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

**Case 20: Not-in-place operation on multi-channel floating point data with TL functions**

```
IppStatusippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3R\_LT

32f\_AC4R\_LT

```
IppStatusippiDivC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

### **Case 21: In-place operation on one-channel integer data with TL functions**

```
IppStatusippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs\_LT    16u\_C1IRSfs\_LT    16s\_C1IRSfs\_LT

### **Case 22: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C3IRSfs\_LT    16u\_C3IRSfs\_LT    16s\_C3IRSfs\_LT

8u\_AC4IRSfs\_LT    16u\_AC4IRSfs\_LT    16s\_AC4IRSfs\_LT

```
IppStatusippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for mod:

8u\_C4IRSfs\_LT    16u\_C4IRSfs\_LT    16s\_C4IRSfs\_LT

### **Case 23: In-place operation on one-channel floating point data with TL functions**

```
IppStatusippiDivC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

### **Case 24: In-place operation on multi-channel integer data with TL functions**

```
IppStatusippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for mod:

32f\_C3IR\_LT

32f\_AC4IR\_LT

```
IppStatusippiDivC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

## **Include Files**

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_t1.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_t1.lib`, `ippi_t1.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>value</code>	The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or four-channel images).
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>scaleFactor</code>	Scale factor (see <a href="#">InInteger Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by dividing pixel values of an image buffer by the constant `value`. For multi-channel images, pixel channel values are divided by the components of a constant vector `value`. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by `scaleFactor` and rounded (not truncated).

When the divisor value is zero, the function execution is aborted and the `ippStsDivByZeroErr` error status is set. Note that in the alpha channel case (AC4), the alpha channels are not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.
<code>ippStsDivByZeroErr</code>	Indicates an error condition if the divisor value is zero.

## Abs

*Computes absolute pixel values of a source image and places them into the destination image.*

### Case 1: Not-in-place operation

```
IppStatusippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

16s\_C1R    32f\_C1R

16s\_C3R    32f\_C3R

16s\_C4R    32f\_C4R

              32f\_AC4R

```
IppStatusippiAbs_16s_AC4R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatusippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

16s\_C1IR    32f\_C1IR

16s\_C3IR    32f\_C3IR

16s\_C4IR    32f\_C4IR

16s\_AC4IR    32f\_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## AbsDiff

Calculates absolute difference between two images.

### Syntax

```
IppStatusippiAbsDiff_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize);
```

Supported values for mod:

8u\_C1R    16u\_C1R    32f\_C1R

8u\_C3R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc1</i>	Pointer to the first source image.
<i>src1Step</i>	Distance in bytes between starts of consecutive lines in the first source image.
<i>pSrc2</i>	Pointer to second source image.
<i>src2Step</i>	Distance in bytes between starts of consecutive lines in the second source image.

---

<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between two images by the formula:

$$pDst(x, y) = \text{abs}(pSrc1(x, y) - pSrc2(x, y)).$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <code>src1Step</code> , <code>src2Step</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## AbsDiffC

*Calculates absolute difference between image and scalar value.*

---

## Syntax

```
IppStatusippiAbsDiffC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int value);
```

Supported values for `mod`:

`8u_C1R`      `16u_C1R`

```
IppStatusippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
-------------	------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>value</i>	Scalar value used to decrement each element of the source image.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between the source image *pSrc* and the scalar *value* by the formula:

$$pDst(x, y) = \text{abs}(pSrc(x, y) - \text{value}).$$

The function clips the *value* to the range [0, 255] for the 8u data type, and to the range [0, 65535] for the 16u data type.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

## Sqr

*Squares pixel values of an image and writes them into the destination image.*

---

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatusippiSqr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
8u_C4RSfs 16u_C4RSfs 16s_C4RSfs
8u_AC4RSfs 16u_AC4RSfs 16s_AC4RSfs
```

**Case 2: Not-in-place operation on floating-point data**

```
IppStatusippiSqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

32f\_C1R  
32f\_C3R  
32f\_C4R  
32f\_AC4R

**Case 3: In-place operation on integer data**

```
IppStatusippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs 16u\_C1IRSfs 16s\_C1IRSfs  
8u\_C3IRSfs 16u\_C3IRSfs 16s\_C3IRSfs  
8u\_C4IRSfs 16u\_C4IRSfs 16s\_C4IRSfs  
8u\_AC4IRSfs 16u\_AC4IRSfs 16s\_AC4IRSfs

**Case 4: In-place operation on floating-point data**

```
IppStatusippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

**Case 5: Threading layer functions based on classic API**

```
IppStatusippiSqr_16s_C1IRSfs_T(Ipp16s* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

```
IppStatusippiSqr_16s32s_C1RSfs_T(const Ipp16s* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, int scaleFactor);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

ippStsNoErr	Indicates an error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Sqrt

Computes square roots of pixel values of a source image and writes them into the destination image.

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatusippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
8u_AC4RSfs 16u_AC4RSfs 16s_AC4RSfs
```

**Case 2: Not-in-place operation on floating-point data**

```
IppStatusippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

32f\_C1R  
32f\_C3R  
32f\_AC4R

**Case 3: In-place operation on integer data**

```
IppStatusippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs 16u\_C1IRSfs 16s\_C1IRSfs  
8u\_C3IRSfs 16u\_C3IRSfs 16s\_C3IRSfs  
8u\_AC4IRSfs 16u\_AC4IRSfs 16s\_AC4IRSfs

**Case 4: In-place operation on floating-point data**

```
IppStatusippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

**Case 5: Threading layer functions based on classic API**

```
IppStatusippiSqrt_16s_C1IRSfs_T(Ipp16s* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

```
IppStatusippiSqrt_16s32s_C1RSfs_T(const Ipp16s* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, int scaleFactor);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

*pSrc* Pointer to the source image ROI.

<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes square roots of pixel values of the source image ROI and writes them into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI. If a source pixel value is negative, the function issues a warning and continues execution with the corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information ).

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsSqrtNegArg</i>	Indicates a warning that a source pixel has a negative value.

## Ln

*Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.*

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatusippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

**Case 2: Not-in-place operation on floating-point data**

```
IppStatusippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

32f\_C1R

32f\_C3R

**Case 3: In-place operation on integer data**

```
IppStatusippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

8u\_C1IRSfs 16u\_C1IRSfs 16s\_C1IRSfs

8u\_C3IRSfs 16u\_C3IRSfs 16s\_C3IRSfs

**Case 4: In-place operation on floating-point data**

```
IppStatusippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code example below.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
ippStsLnZeroArg	Indicates a warning that a source pixel has a zero value.
ippStsLnNegArg	Indicates a warning that a source pixel has a negative value.

## Example

The code example below shows how to use `Ln` function.

```
IppStatus ln( void ) {
    Ipp32f img[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
   ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
    img[0] = -0;
    img[1] = -1;
    st =ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
    printf( "%f %f %f\n", img[0], img[1], img[2] );
    return st;
}
```

Output values:

```
-1.#INF00 -1.#IND00 1.000000
```

Status value and message:

```
(7) Zero value(s) of argument in the Ln function
```

## Exp

*Computes the exponential of pixel values in a source image and writes the results into the destination image.*

---

## Syntax

### Case 1: Not-in-place operation on integer data

```
IppStatusippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

```
8u_C1RSfs 16u_C1RSfs 16s_C1RSfs  
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
```

### Case 2: Not-in-place operation on floating-point data

```
IppStatusippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

```
32f_C1R  
32f_C3R
```

### Case 3: In-place operation on integer data

```
IppStatusippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

```
8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs  
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
```

### Case 4: In-place operation on floating-point data

```
IppStatusippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
32f_C1IR  
32f_C3IR
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes  $e$  to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix A “[Handling of Special Cases](#)” for more information).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## DotProd

*Computes the dot product of pixel values of two source images.*

---

## Syntax

### Case 1: Operation on one-channel integer data

```
IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp);
```

Supported values for *mod*:

8u64f_C1R	16u64f_C1R	16s64f_C1R	32u64f_C1R	32s64f_C1R
-----------	------------	------------	------------	------------

**Case 2: Operation on three-channel integer data**

```
IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);
```

Supported values for mod:

8u64f_C3R	16u64f_C3R	16s64f_C3R	32u64f_C3R	32s64f_C3R
-----------	------------	------------	------------	------------

**Case 3: Operation on four-channel integer data**

```
IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4]);
```

Supported values for mod:

8u64f_C4R	16u64f_C4R	16s64f_C4R	32u64f_C4R	32s64f_C4R
-----------	------------	------------	------------	------------

```
IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);
```

Supported values for mod:

8u64f_AC4R	16u64f_AC4R	32u64f_AC4R
16s64f_AC4R	32s64f_AC4R	

**Case 4: Operation on floating-point data**

```
IppStatusippiDotProd_32f64f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp, IppHintAlgorithm hint);
```

```
IppStatusippiDotProd_32f64f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);
```

```
IppStatusippiDotProd_32f64f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4], IppHintAlgorithm hint);
```

```
IppStatusippiDotProd_32f64f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc1</i> , <i>pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step</i> , <i>src2Step</i>	Distance, in bytes, between the starting points of consecutive lines in the source images.
<i>roiSize</i>	Size of the source image ROI.

<i>pDp</i>	Pointer to the dot product or to the array containing the computed dot products for multi-channel images.
<i>hint</i>	Option to select the algorithmic implementation of the function, see Table “ <a href="#">Hint Arguments for Image Moment Functions</a> ”.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the dot product of pixel values of two source images *pSrc1* and *pSrc2* using algorithm indicated by the hint argument (see Table “[Hint Arguments for Image Moment Functions](#)”) and stores the result in *pDp*. In case of multi-channel images, the dot product is computed for each channel separately and stored in the array *pDp*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the step values has zero or negative value.

## DotProdCol

*Calculates the dot product of taps vector and columns of the specified set of rows.*

---

## Syntax

```
IppStatusippiDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const Ipp32f* pTaps,
int tapsLen, Ipp32f* pDst, int width);

IppStatusippiDotProdCol_64f_L2(const Ipp64f* const ppSrcRow[], const Ipp64f* pTaps,
int tapsLen, Ipp64f* pDst, int width);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>ppSrcRow</i>	Pointer to the set of rows.
<i>pTaps</i>	Pointer to the taps vector.
<i>tapsLen</i>	Length of taps vector, is equal to the number of rows.
<i>pDst</i>	Pointer to the destination row.
<i>width</i>	Width of the source and destination rows.

## Description

This function calculates the dot product of taps vector *pTaps* and columns of the specified set of the rows *ppSrcRow*. It is useful for external vertical filtering pipeline implementation.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>width</i> is less than or equal to 0.

## Complement

*Converts pixel values from two's complement to binary representation.*

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiComplement_<mod>(Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,  
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8s_C1R	16s_C1R	32s_C1R
8s_C3R	16s_C3R	32s_C3R
8s_C4R	16s_C4R	32s_C4R
8s_AC4R	16s_AC4R	32s_AC4R

### Case 2: In-place operation

```
IppStatusippiComplement_<mod>(Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize);
```

Supported values for mod:

8s_C1IR	16s_C1IR	32s_C3IR
8s_C3IR	16s_C3IR	32s_C3IR
8s_C4IR	16s_C4IR	32s_C4IR
8s_AC4IR	16s_AC4IR	32s_AC4IR

## Include Files

ippi.h

## Parameters

*pSrc, pDst, pSrcDst* Pointers to the start of the array with the source image.

<i>srcStep</i> , <i>dstStep</i> , <i>srcDstStep</i>	Distances in bytes between starts of adjacent lines in the source image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function converts the value of each pixel from the two's complement representation to the binary representation. This function takes the data from *pSrc* and saves the result in *pDst*.

In-place functions convert the value of each pixel by rewriting the source data. These functions take the data from *pSrcDst* and saves the result in *pSrcDst*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when any of the specified pointers is <i>null</i> .
<i>ippStsSizeErr</i>	Indicates an error when the value for the data size is incorrect.
<i>ippStsStepErr</i>	Indicates an error when step value is less than, or equal to zero.

## Logical Operations

---

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

### And

*Performs a bitwise AND operation between corresponding pixels of two images.*

---

#### Syntax

##### Case 1: Not-in-place operation

```
IppStatusippiAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

## Case 2: In-place operation

```
IppStatusippiAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

```
8u_C1IR 16u_C1IR 32s_C1IR  
8u_C3IR 16u_C3IR 32s_C3IR  
8u_C4IR 16u_C4IR 32s_C4IR  
8u_AC4IR 16u_AC4IR 32s_AC4IR
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination imager for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channelss.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
---------------	--

## AndC

*Performs a bitwise AND operation of each pixel with a constant.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,  
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R      16u\_C1R      32s\_C1R

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>  
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      16u\_C3R      32s\_C3R  
8u\_AC4R      16u\_AC4R      32s\_AC4R

```
IppStatusippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>  
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R      16u\_C4R      32s\_C4R

#### Case 3: In-place operation on one-channel data

```
IppStatusippiAndC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,  
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      32s\_C1IR

#### Case 4: In-place operation on multi-channel data

```
IppStatusippiAndC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int  
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR      16u\_C3IR      32s\_C3IR  
8u\_AC4IR      16u\_AC4IR      32s\_AC4IR

---

```
IppStatusippiAndC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	32s_C4IR
---------	----------	----------

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>value</code>	The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Or

Performs bitwise inclusive OR operation between pixels of two source buffers.

---

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

### Case 2: In-place operation

```
IppStatusippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.

---

<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Example

The code example below show how to use the function `ippiOr_8u_C1R`.

```
void func_or()
{
    IppiSize Src1ROI = {8,4};
    IppiSize Src2ROI = {8,4};
    IppiSize DstROI = {5,4};
    Ipp8u src1[8*4];
    Ipp8u src2[8*4];
    Ipp8u dst[8*4];

   ippiSet_8u_C1R(0,dest,8,Src1ROI);
   ippiSet_8u_C1R(5,src1,8,Src1ROI);
   ippiSet_8u_C1R(6,src2,8,Src2ROI);
   ippiOr_8u_C1R(src1,8,src2,8,dst,8,DstROI);

}
```

Result:

src1	src2	dst
05 05 05 05 05 05 05	06 06 06 06 06 06 06	07 07 07 07 07 07 00
05 05 05 05 05 05 05	06 06 06 06 06 06 06	07 07 07 07 07 07 00
05 05 05 05 05 05 05	06 06 06 06 06 06 06	07 07 07 07 07 07 00
05 05 05 05 05 05 05	06 06 06 06 06 06 06	07 07 07 07 07 07 00

## OrC

Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.

## Syntax

### **Case 1: Not-in-place operation on one-channel data**

```
IppStatusippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R    16u\_C1R    32s\_C1R

### **Case 2: Not-in-place operation on multi-channel data**

```
IppStatusippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R    16u\_C3R    32s\_C3R

8u\_AC4R    16u\_AC4R    32s\_AC4R

```
IppStatusippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R    16u\_C4R    32s\_C4R

### **Case 3: In-place operation on one-channel data**

```
IppStatusippiOrC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR    16u\_C1IR    32s\_C1IR

### **Case 4: In-place operation on multi-channel data**

```
IppStatusippiOrC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR    16u\_C3IR    32s\_C3IR

8u\_AC4IR    16u\_AC4IR    32s\_AC4IR

```
IppStatusippiOrC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR                  16u\_C4IR                  32s\_C4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>value</code>	The constant value to perform the bitwise OR operation on each pixel of the source image (constant vector in case of multi-channel images).
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI for the in-place operation.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Xor

*Performs bitwise exclusive OR operation between pixels of two source buffers.*

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiXor_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

### Case 2: In-place operation

```
IppStatusippiXor_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## XorC

Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C1R`   `16u_C1R`   `32s_C1R`

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`   `16u_C3R`   `32s_C3R`

`8u_AC4R`   `16u_AC4R`   `32s_AC4R`

```
IppStatusippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C4R`   `16u_C4R`   `32s_C4R`

**Case 3: In-place operation on one-channel data**

```
IppStatusippiXorC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR 16u\_C1IR 32s\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiXorC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR 16u\_C3IR 32s\_C3IR

8u\_AC4IR 16u\_AC4IR 32s\_AC4IR

```
IppStatusippiXorC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR 16u\_C4IR 32s\_C4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

## Not

Performs a bitwise NOT operation on each pixel of a source buffer.

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiNot_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

- 8u\_C1R
- 8u\_C3R
- 8u\_C4R
- 8u\_AC4R

### Case 2: In-place operation

```
IppStatusippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u\_C1IR
- 8u\_C3IR
- 8u\_C4IR
- 8u\_AC4IR

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## RShiftC

Shifts bits in pixel values to the right.

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R

16u\_C1R

16s\_C1R

32s\_C1R

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatusippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R

16u\_C3R

16s\_C3R

32s\_C3R

```
IppStatusippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R

16u\_C4R

16s\_C4R

32s\_C4R

```
IppStatusippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4R

16u\_AC4R

16s\_AC4R

32s\_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatusippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR

16u\_C1IR

16s\_C1IR

32s\_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR

16u\_C3IR

16s\_C3IR

32s\_C3IR

8u\_AC4IR

16u\_AC4IR

16s\_AC4IR

32s\_AC4IR

```
IppStatusippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR

16u\_C4IR

16s\_C4IR

32s\_C4IR

**Include Files**

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## LShiftC

*Shifts bits in pixel values to the left.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C1R      16u\_C1R      32s\_C1R

### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      16u\_C3R      32s\_C3R      32s\_AC4R  
8u\_AC4R      16u\_AC4R

```
IppStatusippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4R      16u\_C4R      32s\_C4R

### Case 3: In-place operation on one-channel data

```
IppStatusippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      32s\_C1IR

### Case 4: In-place operation on multi-channel data

```
IppStatusippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3IR      16u\_C3IR      32s\_C3IR  
8u\_AC4IR      16u\_AC4IR      32s\_AC4IR

```
IppStatusippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C4IR      16u\_C4IR      32s\_C4IR

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.

Note that the functions with the `AC4` descriptor do not process alpha channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below illustrates the use of left shift function.

```
IppStatus lshift( void ) {
    Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
    IppiSize roi = { 8, 8 };
    IppStatus st =ippiLShiftC_8u_C1IR( 1, img, 8, roi );
    printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
    return st;
}
```

Output values:

```
02 fe fc
```

## Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer  $pDst$  is created by overlaying a pixel from the foreground image buffer  $pSrc1$  over a pixel from the background image buffer  $pSrc2$ . The supported types of images combining by using alpha values are listed in the table below.

### Types of Image Composing Operations

Type	Output Pixel		Description in Imaging Terms
	Color Components	Alpha value	
OVER	$\alpha_A * A + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A + (1 - \alpha_A) * \alpha_B$	$A$ occludes $B$
IN	$\alpha_A * A * \alpha_B$	$\alpha_A * \alpha_B$	$A$ within $B$ . $A$ acts as a matte for $B$ . $A$ shows only where $B$ is visible.
OUT	$\alpha_A * A * (1 - \alpha_B)$	$\alpha_A * (1 - \alpha_B)$	$A$ outside $B$ . NOT- $B$ acts as a matte for $A$ . $A$ shows only where $B$ is not visible.
ATOP	$\alpha_A * A * \alpha_B + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * \alpha_B + (1 - \alpha_A) * \alpha_B$	Combination of ( $A$ IN $B$ ) and ( $B$ OUT $A$ ). $B$ is both background and matte for $A$ .
XOR	$\alpha_A * A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B$	Combination of ( $A$ OUT $B$ ) and ( $B$ OUT $A$ ). $A$ and $B$ mutually exclude each other.
PLUS	$\alpha_A * A + \alpha_B * B$	$\alpha_A + \alpha_B$	Blend without precedence

In the formulas above, the input image buffers are denoted as  $A$  and  $B$  for simplicity. The Greek letter  $\alpha$  with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value  $alpha$  as:

$$\alpha = \text{alpha} / \text{max\_val}$$

where  $\text{max\_val}$  is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the `ippiAlphaComp` function that operates on 4-channel RGBA buffers only,  $\alpha_A$  and  $\alpha_B$  are the normalized alpha values of the two input image buffers, respectively.

For the `ippiAlphaCompC` function,  $\alpha_A$  and  $\alpha_B$  are the normalized constant alpha values that are passed as parameters to the function.

Note that in formulas for computing the resultant color channel values,  $A$  and  $B$  stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions `AlphaPremul`, `AlphaPremulC` for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for repeated compositing of an image.

The type of composition operation that is performed by the function `AlphaComp` and `AlphaCompC` is specified by the parameter  $alphaType$ , the table below lists its possible values.

## Possible Values of alphaType Parameter

Operation types	Parameter Value
OVER	ippAlphaOver
IN	ippAlphaIn
OUT	ippAlphaOut
ATOP	ippAlphaATop
XOR	ippAlphaXor
PLUS	ippAlphaPlus
	ippAlphaOverPremul
	ippAlphaInPremul
	ippAlphaOutPremul
	ippAlphaATopPremul
	ippAlphaXorPremul
	ippAlphaPlusPremul

## AlphaComp

Combines two images using alpha (opacity) values of both images.

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
IppiAlphaType alphaType);
```

Supported values for mod:

8u_AC1R	16u_AC1R	

8u_AC4R	16u_AC4R	

```
IppStatusippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step, const
Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>* const pDst[4], int dstStep,
IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for mod:

8u_AP4R	16u_AP4R
---------	----------

## Case 2: In-place operation

```
IppStatusippiAlphaComp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for mod:

8u_AC4IR	16u_AC4IR	

```
IppStatusippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,  
Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize, IppiAlphaType  
alphaType);
```

Supported values for mod:

8u_AP4IR	16u_AP4IR
----------	-----------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep, src1Step, src2Step</i>	Distances, in bytes, between the starting points of consecutive lines in the source images.
<i>pSrcDst</i>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.

*alphaType*

The composition type to perform. See [Table "Possible Values of the Parameter alphaType"](#) for the type value and description.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels  $(r_A, g_A, b_A, \alpha_A)$  from the foreground image  $pSrc1$  with pixels  $(r_B, g_B, b_B, \alpha_B)$  from the background image  $pSrc2$  to produce pixels  $(r_C, g_C, b_C, \alpha_C)$  in the resultant image  $pDst$ . The alpha values are assumed to be normalized to the range [0..1].

The type of the compositing operation is indicated by the *alphaType* parameter. Use [Table "Possible Values of the Parameter alphaType"](#) to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see [Table "Types of Image Composing Operations"](#)) are computed as follows:

$$r_C = \alpha_A * r_A + (1 - \alpha_A) * \alpha_B * r_B$$

$$g_C = \alpha_A * g_A + (1 - \alpha_A) * \alpha_B * g_B$$

$$b_C = \alpha_A * b_A + (1 - \alpha_A) * \alpha_B * b_B$$

The resulting (normalized) alpha value is computed as

$$\alpha_C = \alpha_A + (1 - \alpha_A) * \alpha_B$$

This function can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## AlphaCompC

*Combines two images using constant alpha values.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype> alpha2, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for `mod`:

8u\_C1R 8u\_C3R 8u\_C4R 8u\_AC4R

16u\_C1R 16u\_C3R 16u\_C4R 16u\_AC4R

16s\_C1R

32u\_C1R

32s\_C1R  
32f\_C1R

```
IppStatusippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step,
Ipp<datatype> alpha1, const Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>
alpha2, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

**Supported values for mod:**

8u\_AP4R 16u\_AP4R

### Case 2: In-place operation

```
IppStatusippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
alpha1, Ipp<datatype>* pSrcDst, int srcDstStep, Ipp<datatype> alpha2, IppiSize roiSize,
IppiAlphaType alphaType);
```

**Supported values for mod:**

8u\_C1IR 16u\_C1IR 16s\_C1IR 32s\_C1IR 32u\_C1IR 32f\_C1IR  
8u\_C3IR 16u\_C3IR  
8u\_C4IR 16u\_C4IR  
8u\_AC4IR 16u\_AC4IR

```
IppStatusippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype> alpha1, Ipp<datatype>* const pSrcDst[4], int srcDstStep, Ipp<datatype>
alpha2, IppiSize roiSize, IppiAlphaType alphaType);
```

**Supported values for mod:**

8u\_AP4IR 16u\_AP4IR

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

*pSrc1, pSrc2*

Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.

*src1Step, src2Step*

Distances, in bytes, between the starting points of consecutive lines in the source images.

<i>pSrcDst</i>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alpha1, alpha2</i>	Constant alpha values to use for the compositing operation.
<i>alphaType</i>	The composition type to perform. See <a href="#">Table "Possible Values of the Parameter alphaType"</a> for the type value and description.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on one-channel image buffers, three-channel RGB and four-channel RGBA image buffers and on planar images, using constant alpha values *alpha1* and *alpha2*. These values are passed to the function as parameters.

The compositing is done by overlaying pixels from the foreground image ROI *pSrc1* with pixels from the background image ROI *pSrc2* to produce pixels in the resultant image ROI *pDst*. The alpha values are normalized to the range [0..1].

The type of the compositing operation is indicated by the *alphaType* parameter. Use [Table "Possible Values of the Parameter alphaType"](#) to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see [Table "Types of Image Composing Operations"](#)) are computed as follows:

$$r_C = \alpha_1 * r_A + (1 - \alpha_1) * \alpha_2 * r_B$$

$$g_C = \alpha_1 * g_A + (1 - \alpha_1) * \alpha_2 * g_B$$

$$b_C = \alpha_1 * b_A + (1 - \alpha_1) * \alpha_2 * b_B$$

where  $\alpha_1, \alpha_2$  are the normalised alpha values *alpha1, alpha2*.

This function can be used for unsigned pixel data only.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below shows how to use alpha composition function.

```
IppStatus acomp( void ) {
    Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
   ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal );
   ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal );
    st =ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255, imgc, 8, roi, ippAlphaOver );
    printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d //
c=a*A+b*(1-A)*B\n",imga[6],imgb[6],imgc[6] );
    return st;
}
```

## Output

```
over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)
```

## AlphaPremul

*Pre-multiplies pixel values of an image by its alpha values.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4R    16u\_AC4R

```
IppStatusippiAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AP4R    16u\_AP4R

#### Case 2: In-place operation

```
IppStatusippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_AC4IR    16u\_AC4IR

```
IppStatusippiAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4], int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_AP4IR    16u\_AP4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If  $(r, g, b, a)$  are the red, green, blue, and alpha values of a pixel, then new pixel values are  $(r * \alpha, g * \alpha, b * \alpha, a)$  after execution of this function. Here  $\alpha$  is the pixel normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## AlphaPremulC

*Pre-multiplies pixel values of an image using constant alpha (opacity) values.*

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> alpha, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_C1R      16u\_C1R

8u\_C3R      16u\_C3R

8u\_C4R      16u\_C4R

8u\_AC4R    16u\_AC4R

```
IppStatusippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype> alpha, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_AP4R    16u\_AP4R

### Case 2: In-place operation

```
IppStatusippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_C1IR    16u\_C1IR

8u\_C3IR    16u\_C3IR

8u\_C4IR    16u\_C4IR

8u\_AC4IR   16u\_AC4IR

```
IppStatusippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

Supported values for mod :

8u\_AP4IR   16u\_AP4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alpha</i>	Global alpha value used for pre-multiplying pixel values.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts either a one-, three-, four-channel image or planar RGBA image to the pre-multiplied alpha form, using the global alpha value *alpha*. For one-, three-, four-channel image buffers, pixel values in each channel are multiplied by  $\alpha$ ; for RGBA (pixel order and planar) images with (r,g,b,a) pixel values, new pixel values are ( $r*\alpha$ ,  $g*\alpha$ ,  $b*\alpha$ ,  $\alpha$ ) after execution of this function. Here  $\alpha$  is the normalized *alpha* value in the range 0 to 1.

The function `ippiAlphaPremulC` can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

# Image Color Conversion

6

This chapter describes image processing functions that perform different types of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale and vice versa
- Different types of format conversion:
  - from pixel-order to planar format and vice versa
  - changing number of channels or planes
  - changing sampling format
  - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist
- Color keying

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest (ROI, see [Regions of Interest in Intel IPP](#)) used in function operations may extend to the size of the whole image.

This chapter starts with introductory material that discusses color space models essential for understanding of the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see [\[Jack01\]](#), [\[Rogers85\]](#), and [\[Foley90\]](#).

## Gamma Correction

Gamma correction of images is used to optimize the usage of data type depth when encoding an image by taking advantage of the non-linear manner in which humans perceive light and color. This non-linearity must be compensated to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a non-linear form using an inverse transformation. This process is called *gamma correction*.

The Intel IPP functions use the following basic equations to convert an RGB image to a gamma-corrected R'G'B' image:

```

for R,G,B < 0.018
  R' = 4.5R
  G' = 4.5G
  B' = 4.5B
for R,G,B ≥ 0.018
  R' = 1.099R0.45 - 0.099
  G' = 1.099G0.45 - 0.099
  B' = 1.099B0.45 - 0.099

```

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with ITU Rec.709 specification (see [\[ITU709\]](#)).

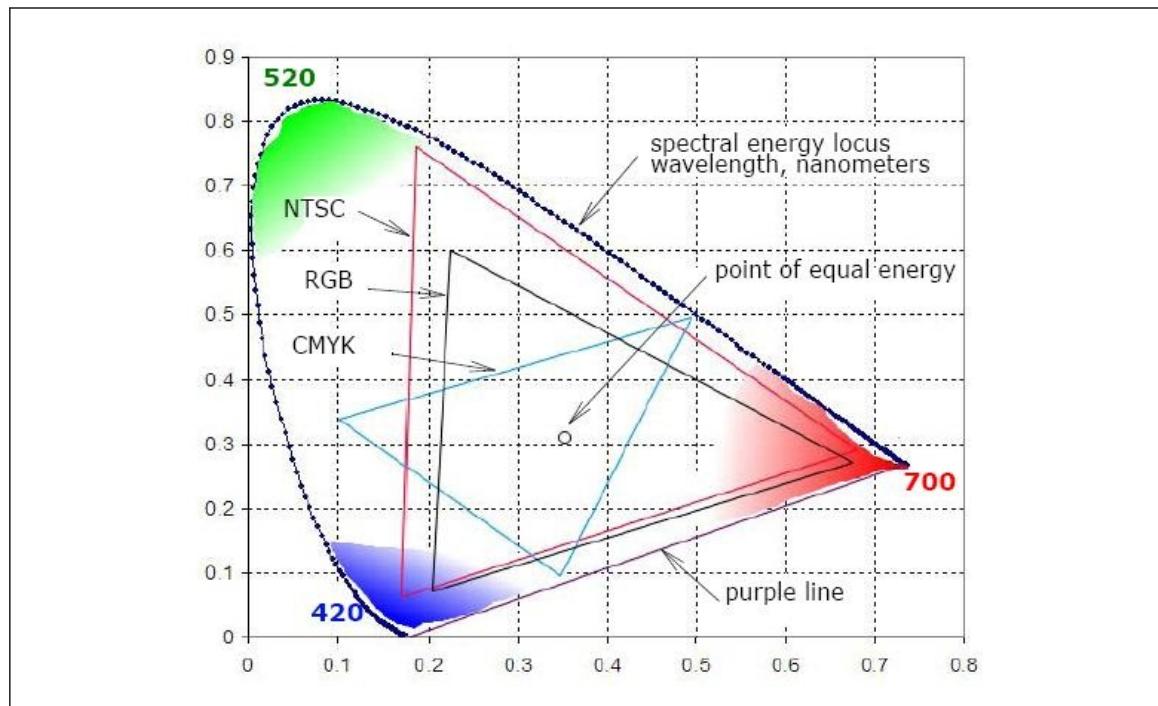
## CIE Chromaticity Diagram and Color Gamut

Figure CIE xyY Chromaticity Diagram and Color Gamut presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, <http://members.eunet.at/cie>). The diagram presents visible colors as a function of  $x$  (red) and  $y$  (green) components called *chromaticity coordinates*. Positions of various spectrum colors (from violet to red) are indicated as the points of a tongue-shaped curve called *spectrum locus*. The straight line connecting the ends of the curve is called the *purple line*. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. A straight-line segment joining any two points in the diagram defines all color variations that can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.

The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors cannot enclose all visible colors. For example, Figure CIE xyY Chromaticity Diagram and Color Gamut shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

### CIE xyY Chromaticity Diagram and Color Gamut



## Color Models

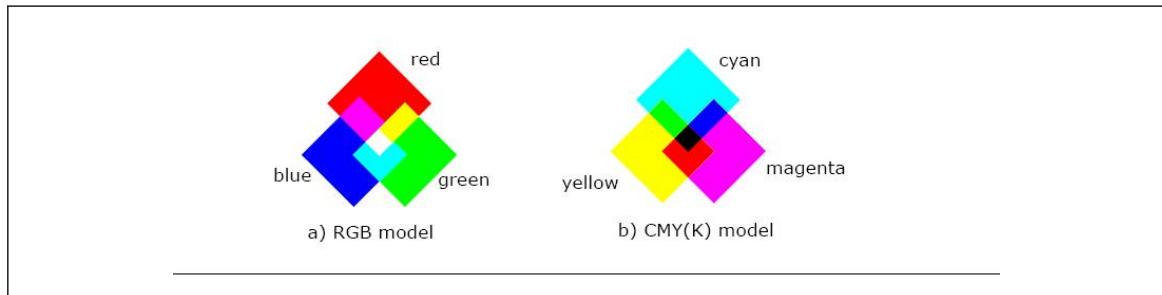
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC\* is used in PhotoCD\* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

## RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

### Primary and Secondary Colors for RGB and CMYK Models



The color subspace of interest is a cube shown in *Figure "RGB and CMY Color Models"* (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

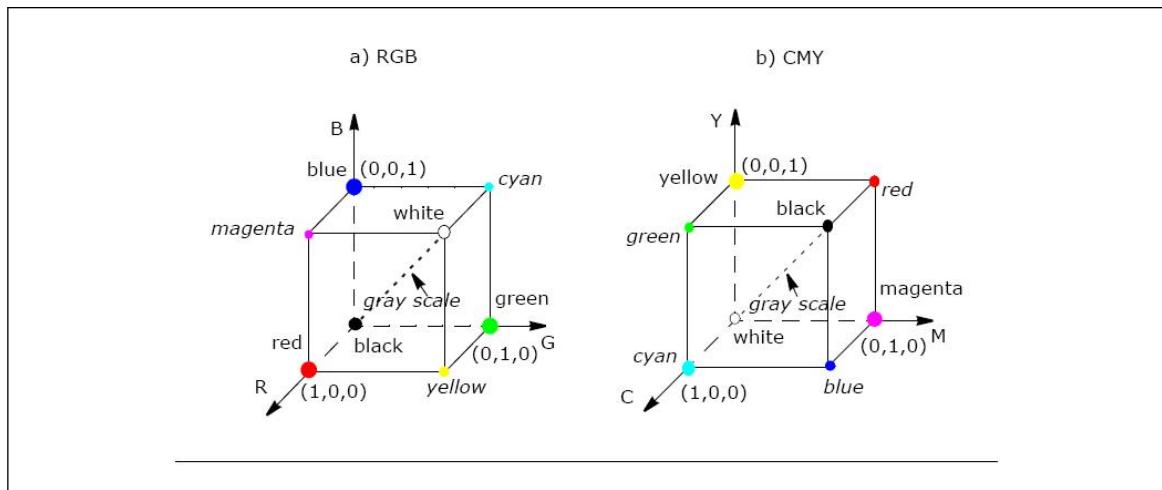
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, the Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

## RGB and CMY Color Models



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

## CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see [Figure "Primary and Secondary Colors for RGB and CMYK Models"](#)), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in [Figure "RGB and CMY Color Models"](#), in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

## YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is “derived” from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance*, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. [Figure "RGB Colors Cube in the YUV Color Space"](#) shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1].

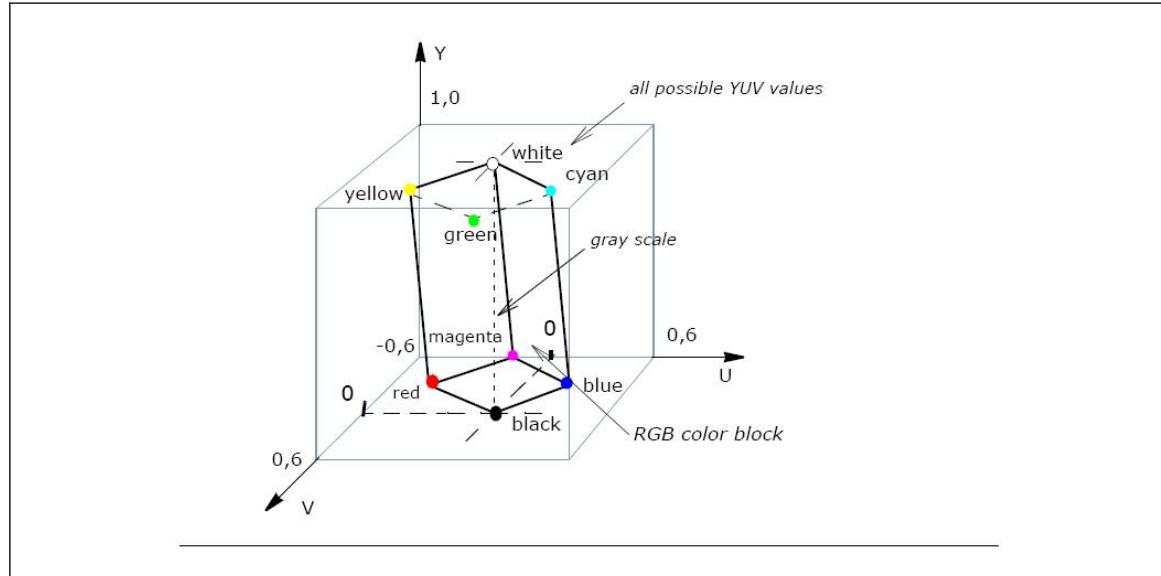
The  $Y'U'V'$  notation means that the components are derived from gamma-corrected  $R'G'B'$ . Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma $Y'$* . (*Luma* is often loosely referred to as *luminance*, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

The Intel IPP functions use the following basic equation ([\[Jack01\]](#)) to convert between gamma-corrected  $R'G'B'$  and  $Y'U'V'$  models:

$$\begin{aligned} Y' &= 0.299 * R' + 0.587 * G' + 0.114 * B' \\ U' &= -0.147 * R' - 0.289 * G' + 0.436 * B' = 0.492 * (B' - Y') \\ V' &= 0.615 * R' - 0.515 * G' - 0.100 * B' = 0.877 * (R' - Y') \\ R' &= Y' + 1.140 * V' \\ G' &= Y' - 0.394 * U' - 0.581 * V' \end{aligned}$$

$$B' = Y' + 2.032 \cdot U'$$

### RGB Colors Cube in the YUV Color Space



There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described later in this chapter in [Image Downsampling](#).

### YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [[Jack01](#)] to convert between  $R'G'B'$  in the range 0-255 and  $Y'C_b'C_r'$  (this notation means that all components are derived from gamma-corrected  $R'G'B'$ ):

$$Y' = 0.257 \cdot R' + 0.504 \cdot G' + 0.098 \cdot B' + 16$$

$$C_b' = -0.148 \cdot R' - 0.291 \cdot G' + 0.439 \cdot B' + 128$$

$$C_r' = 0.439 \cdot R' - 0.368 \cdot G' - 0.071 \cdot B' + 128$$

$$R' = 1.164 \cdot (Y' - 16) + 1.596 \cdot (C_r' - 128)$$

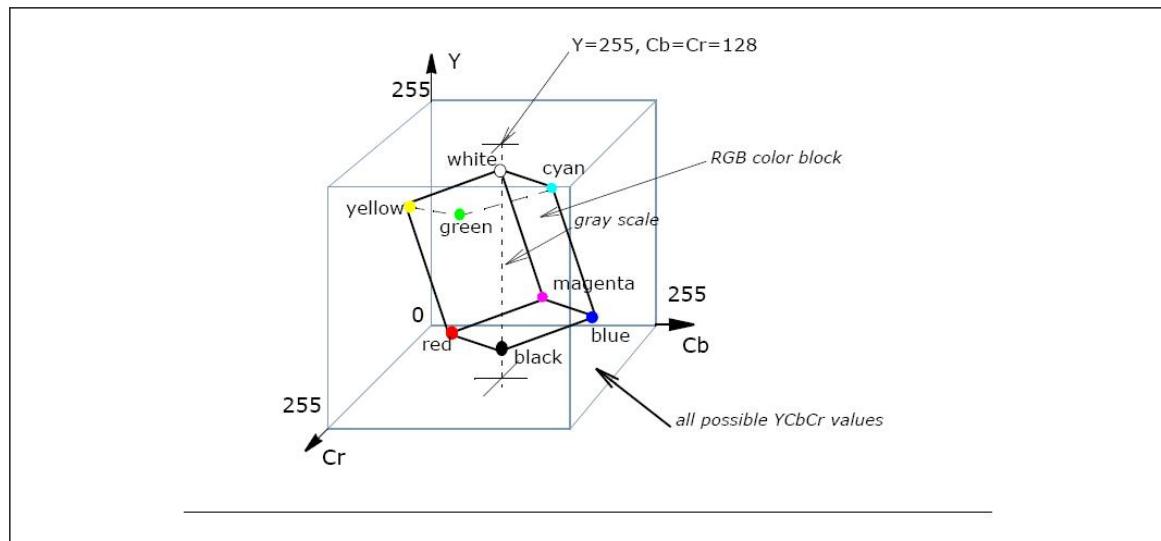
$$G' = 1.164 \cdot (Y' - 16) - 0.813 \cdot (C_r' - 128) - 0.392 \cdot (C_b' - 128)$$

$$B' = 1.164 \cdot (Y' - 16) + 2.017 \cdot (C_b' - 128)$$

Possible RGB colors occupy only part of the YCbCr color space (see [Figure "RGB Colors Cube in the YCbCr Space"](#)) limited by the nominal ranges, therefore there are many YCbCr combinations that result in invalid RGB values.

There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described later in this chapter in [Image Downsampling](#).

### RGB Colors Cube in the YCbCr Space



### PhotoYCC Color Model

The Kodak\* PhotoYCC\* was developed for encoding Photo CD\* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([\[ITU709\]](#)). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than the one that can currently be displayed.

The Intel IPP functions use the following basic equations [\[Jack01\]](#) to convert non-linear gamma-corrected R'G'B' to Y'C'C':

$$Y' = 0.213 * R' + 0.419 * G' + 0.081 * B'$$

$$C1' = -0.131 * R' - 0.256 * G' + 0.387 * B' + 0.612$$

$$C2' = 0.373 * R' - 0.312 * G' - 0.061 * B' + 0.537$$

The equations above are given on the assumption that R', G', and B' values are normalized to the range [0..1].

Since the PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [\[Jack01\]](#) are used in Intel IPP to generate R'G'B' values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

$$R' = 0.981 * Y + 1.315 * (C2 - 0.537)$$

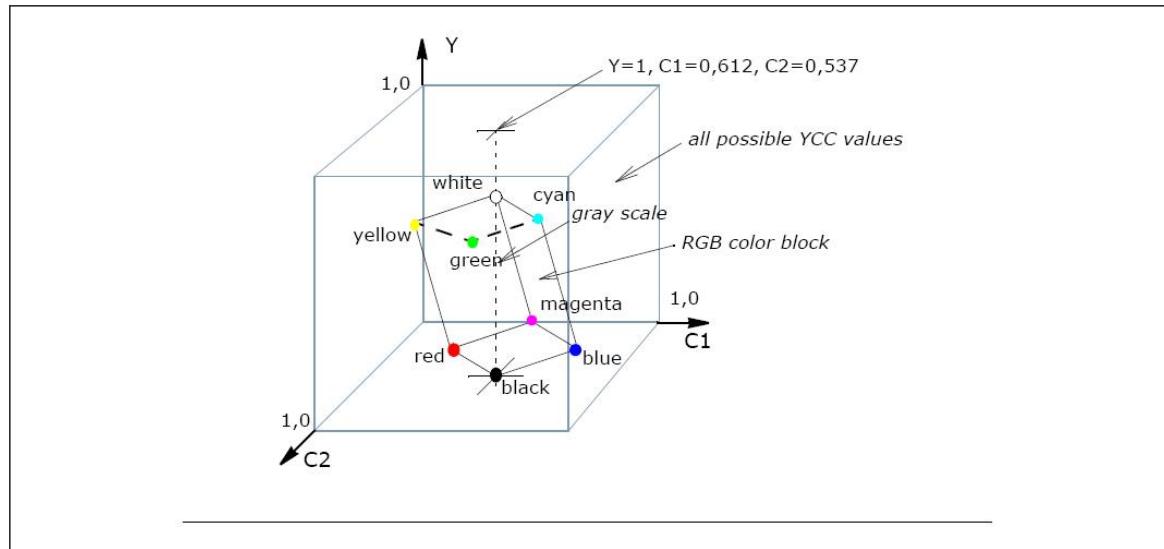
$$G' = 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537)$$

$$B' = 0.981 * Y + 1.601 * (C1 - 0.612)$$

The equations above are given on the assumption that source Y, C1 and C2 values are normalized to the range [0..1], and the display primaries have the chromaticity values in accordance with [\[ITU709\]](#) specifications.

The possible RGB colors occupy only part of the YCC color space (see Figure "RGB Colors in the YCC Color Space") limited by the nominal ranges, therefore there are many YCC combinations that result in invalid RGB values.

### RGB Colors in the YCC Color Space



### YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance ( $Y$ ) and two color difference components ( $C_o$  - offset orange,  $C_g$  - offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between RGB and YCoCg:

$$Y = R/4 + G/2 + B/4$$

$$C_o = R/2 - B/2$$

$$C_g = -R/4 + G/2 - B/4$$

$$R = Y + C_o - C_g$$

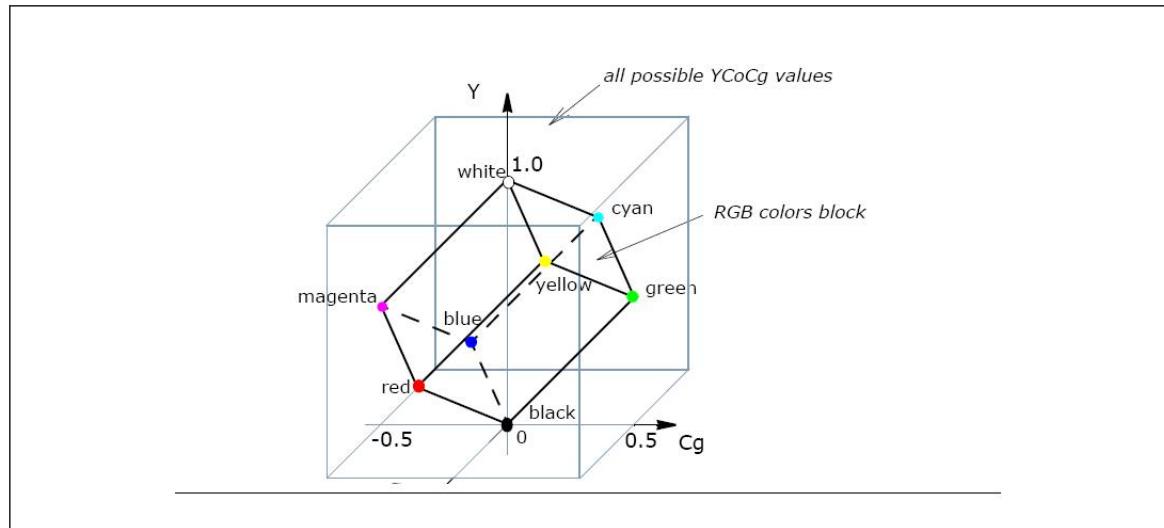
$$G = Y + C_g$$

$$B = Y - C_o - C_g$$

A variation of this color space which is called YCoCg-R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].

The possible RGB colors occupy only part of the YCoCg color space (see Figure "RGB Color Cube in the YCoCg Color Space") limited by the nominal ranges, therefore there are many YCoCg combinations that result in invalid RGB values.

### RGB Color Cube in the YCoCg Color Space



### HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more "intuitive" in manipulating with color and were designed to approximate the way humans perceive and interpret color.

*Hue* defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

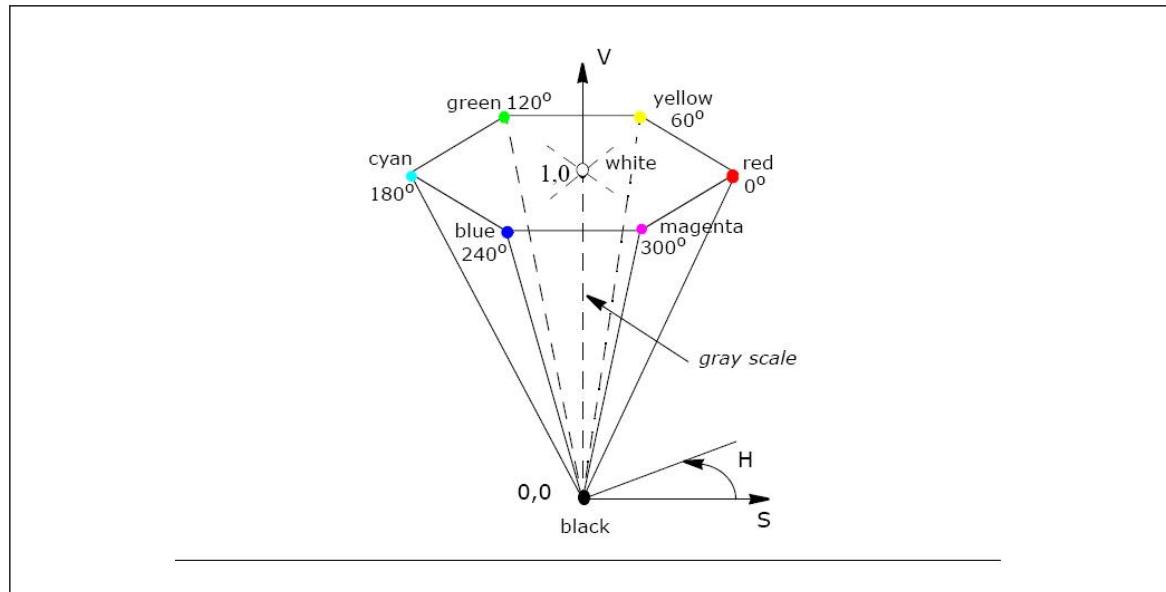
*Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue ( $S=1$ ) is at *value*  $V=1$  (full illumination) in the HSV color model, and at *lightness*  $L=0.5$  in the HLS color model.

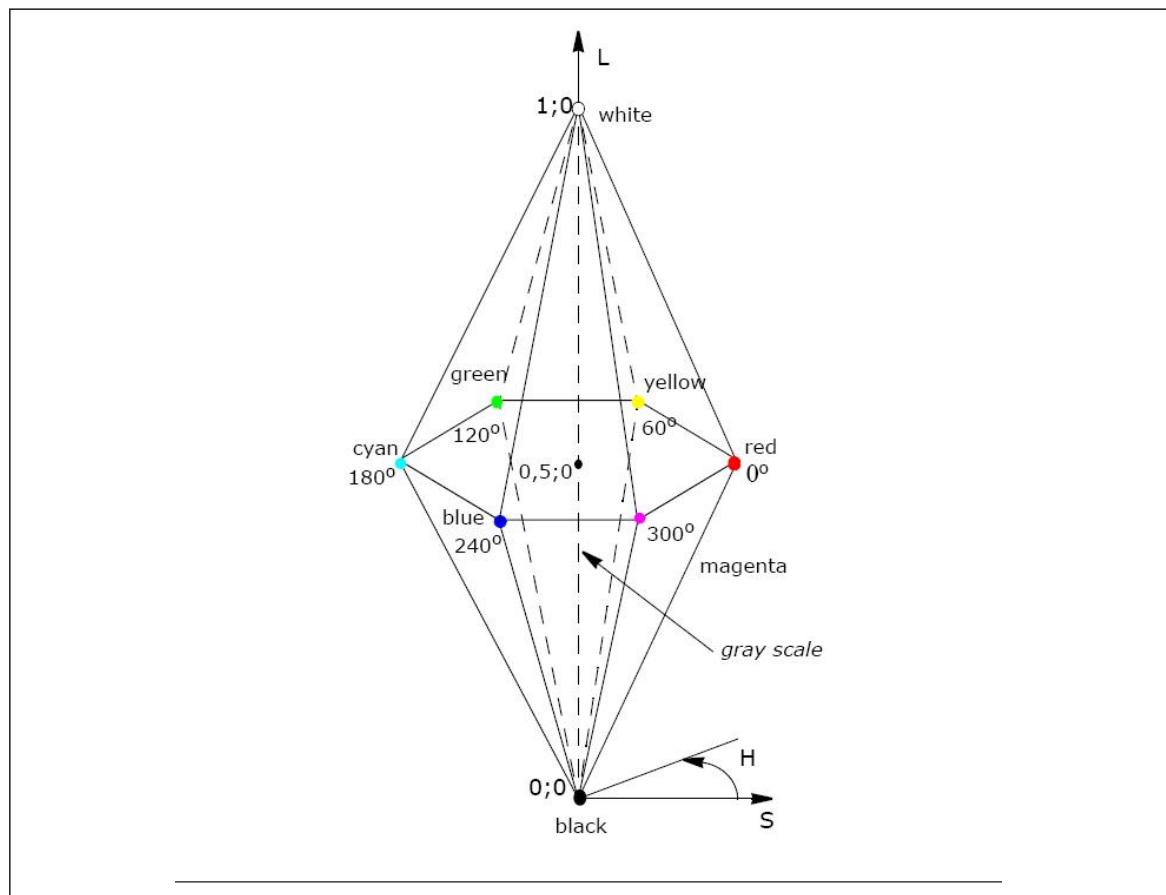
The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the [Figure "HSV Solid"](#), because the hexcone defines the subset of the HSV space with valid RGB values. The *value*  $V$  is the vertical axis, and the vertex  $V=0$  corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone ([Figure "HSV Solid"](#)) with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from [RGBToHSV](#)/[RGBToHSV](#) and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [Rogers85] given in the descriptions of corresponding conversion functions.

### HSV Solid



### HLS Solid

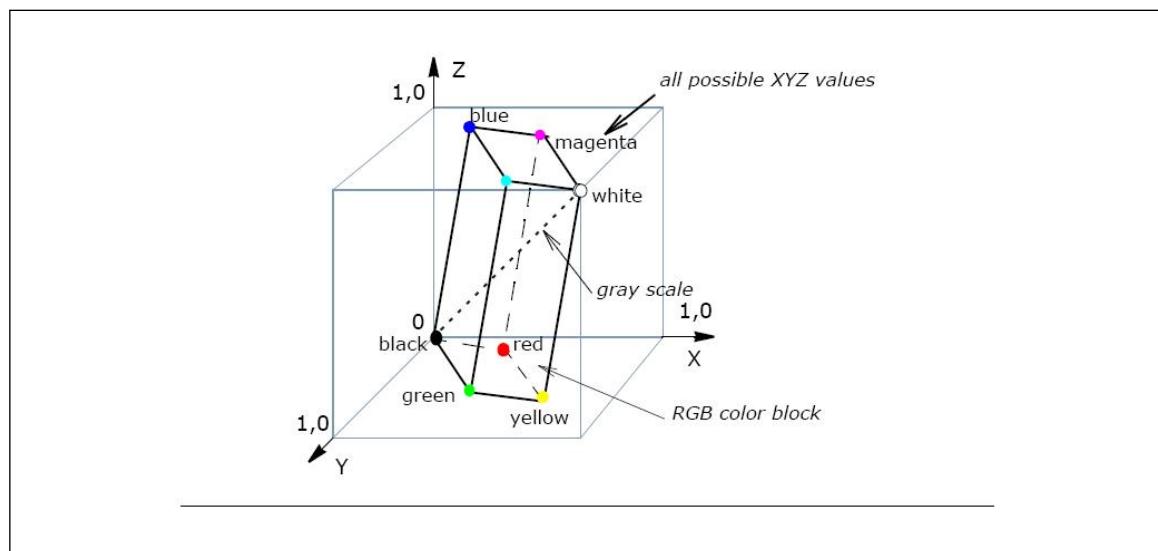


## CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it) is that this space is completely device-independent. The chromaticity diagram in [Figure "CIE xyY Chromaticity Diagram and Color Gamut"](#) is in fact a two-dimensional projection of the CIE XYZ sub-space. Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in Figure "RGB Colors Cube in the XYZ Color Space".

### RGB Colors Cube in the XYZ Color Space



Intel IPP functions use the following basic equations [\[Rogers85\]](#), to convert between gamma-corrected R'G'B' and CIE XYZ models:

$$X = 0.412453*R' + 0.35758 *G' + 0.180423*B'$$

$$Y = 0.212671*R' + 0.71516 *G' + 0.072169*B'$$

$$Z = 0.019334*R' + 0.119193*G' + 0.950227*B'$$

The equations for X, Y, Z calculation are given on the assumption that R', G', and B' values are normalized to the range [0..1].

$$R' = 3.240479 * X - 1.53715 * Y - 0.498535 * Z$$

$$G' = -0.969256 * X + 1.875991 * Y + 0.041556 * Z$$

$$B' = 0.055648 * X - 0.204043 * Y + 1.057311 * Z$$

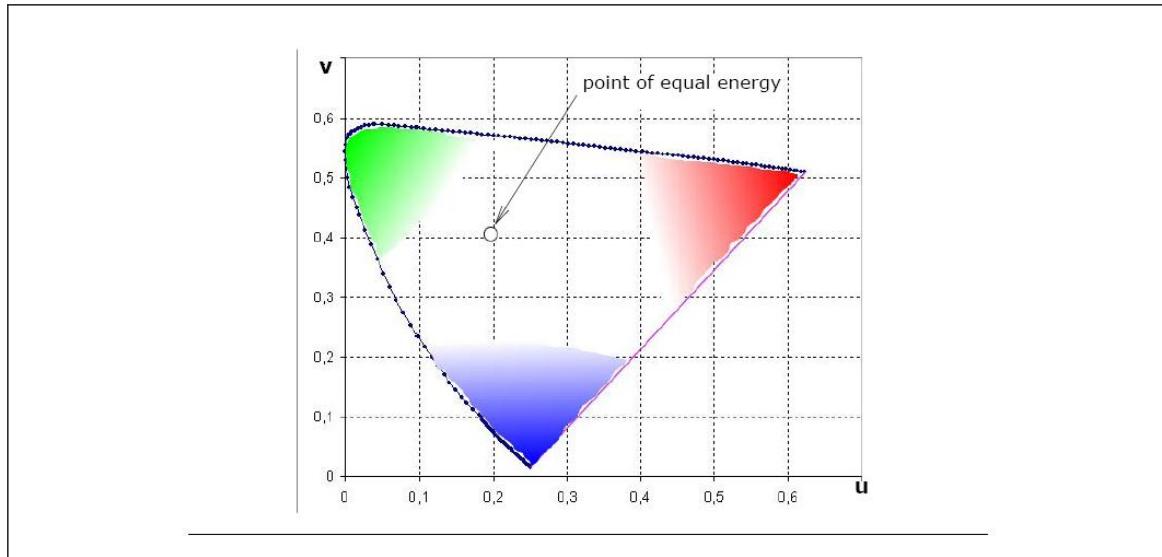
The equations for R', G', and B' calculation are given on the assumption that X, Y, and Z values are in the range [0..1].

## CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. "Perceptually uniform" means that two colors that are equally distant in the color space are equally distant perceptually. To

accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE ([Figure "CIE  \$u',v'\$  Uniform Chromaticity Scale Diagram"](#)). The UCS diagram uses a mathematical formula to transform the XYZ values or  $x, y$  coordinates ([Figure "CIE xyY Chromaticity Diagram and Color Gamut"](#)), to a new set of values that present a visually more accurate two-dimensional model. The Y lightness scale is replaced with a new scale called L that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are U and V for CIE LUV, and  $a$  and  $b$  (referred to also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

### CIE $u',v'$ Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows ([\[Rogers85\]](#)),

$$L = 116 \cdot (Y/Y_n)^{1/3} - 16.$$

$$U = 13 \cdot L \cdot (u - u_n)$$

$$V = 13 \cdot L \cdot (v - v_n)$$

where

$$u = 4 \cdot X / (X + 15 \cdot Y + 3 \cdot Z)$$

$$v = 9 \cdot Y / (X + 15 \cdot Y + 3 \cdot Z)$$

$$u_n = 4 \cdot x_n / (-2 \cdot x_n + 12 \cdot y_n + 3)$$

$$v_n = 9 \cdot y_n / (-2 \cdot x_n + 12 \cdot y_n + 3)$$

Inverse conversion is performed in accordance with equations:

$$Y = Y_n \cdot ((L + 16) / 116)^3$$

$$X = -9 \cdot Y \cdot u / ((u - 4) \cdot v - u \cdot v)$$

$$Z = (9 \cdot Y - 15 \cdot v \cdot Y - v \cdot X) / 3 \cdot v$$

where

$$u = U / (13 \cdot L) + u_n$$

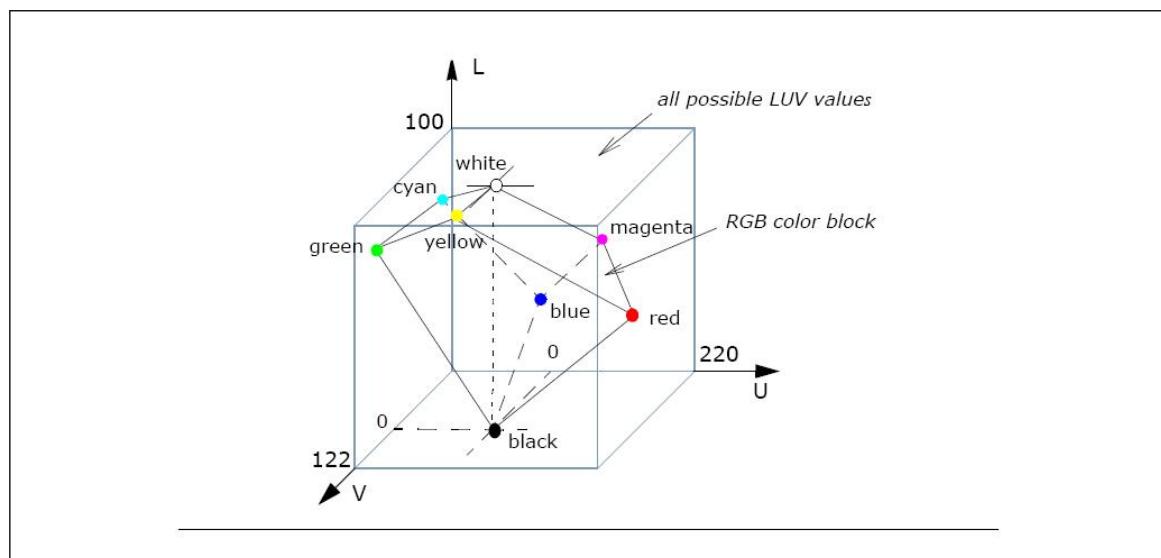
$$v = V / (13 \cdot L) + v_n$$

and  $u_n, v_n$  are defined above.

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and  $y_n = 1.0$  is the luminance of the D65 white point. The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see Figure 6-12) limited by the nominal ranges, therefore there are many LUV combinations that result in invalid RGB values.

### RGB Color Cube in the CIE LUV Color Space



The CIE Lab color space is derived from CIE XYZ as follows:

$$L = 116 \cdot (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$$

$$L = 903.3 \cdot (Y/Y_n)^{1/3} \text{ for } Y/Y_n \leq 0.008856$$

$$a = 500 \cdot [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200 \cdot [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$f(t) = t^{1/3} - 16 \text{ for } t > 0.008856$$

$$f(t) = 7.787 \cdot t + 16/116 \text{ for } t \leq 0.008856$$

Here  $Y_n = 1.0$  is the luminance, and  $X_n = 0.950455$ ,  $Z_n = 1.088753$  are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$$Y = Y_n \cdot P^3$$

$$X = X_n \cdot (P + a/500)^3$$

$$Z = Z_n \cdot (P - b/200)^3$$

where

$$P = (L + 16)/116$$

## Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow the number of bits required for acceptable color description of an image to be reduced. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance. The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called *macropixels*) in accordance with some specific rules. This process is called *downsampling* and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by the Intel IPP image processing functions (excluding the JPEG functions):

**4:4:4 YUV (YCbCr)** - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, than every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with the 4:4:4 descriptor omitted.

**4:2:2 YUV (YCbCr)** - uses the 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

**4:1:1 YCbCr** - uses the 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

**4:2:0 YUV (YCbCr)** - uses the 2:1 horizontal downsampling and the 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Therefore, the Intel IPP functions specific for a JPEG codec, support the following sampling formats:

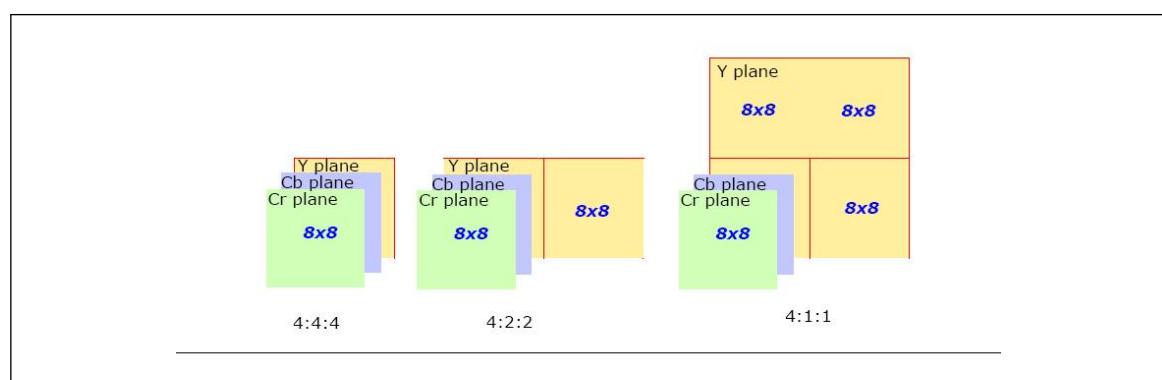
**4:4:4 YCbCr** - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:2:2 YCbCr** - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:1:1 YCbCr** - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in Figure MCU Structure for Different JPEG Sampling Formats.

### MCU Structure for Different JPEG Sampling Formats



## RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, the Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats, which include three RGB channels plus alpha channel. For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in [Table "Pixel-Order Image Formats"](#).

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. The figure below shows all the supported 16-bit-per-pixel formats and their memory layout (bit order):

### 16-bit pixel formats

16-bit pixel formats	
	High-order byte      Low-order byte
RGB565	B4 B3 B2 B1 B0 G5 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0      G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0      G3 G2 G1 G0 B3 B2 B1 B0

R - Red, G - Green, B - Blue, X - free bit

## Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

[Table "Pixel-Order Image Formats"](#) lists all pixel-order image formats that are supported by the Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol A denotes alpha-channel value). The last column of this table gives an example of an Intel IPP color conversion function that uses the respective image format.

### Pixel-Order Image Formats

Image Format	Number of Channel s	Channel Values Order	Example
RGB	3	<u>R0</u> <u>G0</u> <u>B0</u> <u>R1</u> <u>G1</u> <u>B1</u> <u>R2</u> <u>G2</u> <u>B2</u>	ippiRGBToYUV_8u_C3R
RGB444			ippiYCbCrToRGB444_8u16u_C3R
RGB555			ippiYCbCrToRGB555_8u16u_C3R
RGB565			ippiYCbCrToRGB565_8u16u_C3R
RGB	4	<u>R0</u> <u>G0</u> <u>B0</u> <u>A0</u> <u>R1</u> <u>G1</u> <u>B1</u> <u>A1</u>	ippiRGBToYUV_8u_Ac4R
BGR	3	<u>B0</u> <u>G0</u> <u>R0</u> <u>B1</u> <u>G1</u> <u>R1</u> <u>B2</u> <u>G2</u> <u>R2</u>	ippiYCbCrToBGR_8u_P3C3R
BGR444			ippiYCbCrToBGR444_8u16u_C3R

Image Format	Number of Channel s	Channel Values Order	Example
BGR555			ippiYCbCrToBGR555_8u16u_C3R
BGR565			ippiYCbCrToBGR565_8u16u_C3R
BGR	4	<u>B0</u> <u>G0</u> <u>R0</u> <u>A0</u> <u>B1</u> <u>G1</u> <u>R1</u> <u>A1</u>	ippiBGRToHLS_8u_A_C4R
YUV	3	<u>Y0</u> <u>U0</u> <u>V0</u> <u>Y1</u> <u>U1</u> <u>V1</u> <u>Y2</u> <u>U2</u> <u>V2</u>	ippiYUVToRGB_8u_C3R
YUV	4	<u>Y0</u> <u>U0</u> <u>V0</u> <u>A0</u> <u>Y1</u> <u>U1</u> <u>V1</u> <u>A1</u>	ippiYUVToRGB_8u_A_C4R
4:2:2 YUV	2	<u>Y0</u> <u>U0</u> <u>Y1</u> <u>V0</u> <u>Y2</u> <u>U1</u> <u>Y3</u> <u>V1</u>	ippiYUV422ToRGB_8u_C2C3R
YCbCr	3	<u>Y0</u> <u>Cb0</u> <u>Cr0</u> <u>Y1</u> <u>Cb1</u> <u>Cr1</u>	ippiYCbCrToRGB_8u_C3R
YCbCr	4	<u>Y0</u> <u>Cb0</u> <u>Cr0</u> <u>A0</u> <u>Y1</u> <u>Cb1</u> <u>Cr1</u> <u>A1</u>	ippiYCbCrToRGB_8u_A_C4R
4:2:2 YCbCr	2	<u>Y0</u> <u>Cb0</u> <u>Y1</u> <u>Cr0</u> <u>Y2</u> <u>Cb1</u> <u>Y3</u> <u>Cr1</u>	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 YCrCb	2	<u>Y0</u> <u>Cr0</u> <u>Y1</u> <u>Cb0</u> <u>Y2</u> <u>Cr1</u> <u>Y3</u> <u>Cb1</u>	ippiYCrCb422ToYCbCr422_8u_C2P3R
4:2:2 CbYCr	2	<u>Cb0</u> <u>Y0</u> <u>Cr0</u> <u>Y1</u> <u>Cb1</u> <u>Y2</u> <u>Cr1</u> <u>Y3</u>	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	<u>X0</u> <u>Y0</u> <u>Z0</u> <u>X1</u> <u>Y1</u> <u>Z1</u> <u>X2</u> <u>Y2</u> <u>Z2</u>	ippiXYZToRGB_8u_C3R
XYZ	4	<u>X0</u> <u>Y0</u> <u>Z0</u> <u>X1</u> <u>Y1</u> <u>Z1</u> <u>X2</u> <u>Y2</u> <u>Z2</u>	ippiXYZToRGB_16u_A_C4R
LUV	3	<u>L0</u> <u>U0</u> <u>V0</u> <u>L1</u> <u>U1</u> <u>V1</u> <u>L2</u> <u>U2</u> <u>V2</u>	ippiLUVToRGB_16s_C3R
LUV	4	<u>L0</u> <u>U0</u> <u>V0</u> <u>A0</u> <u>L1</u> <u>U1</u> <u>V1</u> <u>A1</u>	ippiLUVToRGB_32f_A_C4R
YCC	3	<u>Y0</u> <u>C0</u> <u>C0</u> <u>Y1</u> <u>C1</u> <u>C1</u> <u>Y2</u> <u>C2</u> <u>C2</u>	ippiYCCToRGB_8u_C3R
YCC	4	<u>Y0</u> <u>C0</u> <u>C0</u> <u>A0</u> <u>Y1</u> <u>C1</u> <u>C1</u> <u>A1</u>	ippiYCCToRGB_8u_A_C4R
HLS	3	<u>H0</u> <u>L0</u> <u>S0</u> <u>H1</u> <u>L1</u> <u>S1</u> <u>H2</u> <u>L2</u> <u>S2</u>	ippiHLSToRGB_16u_C3R
HLS	4	<u>H0</u> <u>L0</u> <u>S0</u> <u>A0</u> <u>H1</u> <u>L1</u> <u>S1</u> <u>A0</u>	ippiHLSToRGB_16u_AC4R
HSV	3	<u>H0</u> <u>S0</u> <u>V0</u> <u>H1</u> <u>S1</u> <u>V1</u> <u>H2</u> <u>S2</u> <u>V2</u>	ippiHSVToRGB_16s_C3R
HSV	4	<u>H0</u> <u>S0</u> <u>V0</u> <u>A0</u> <u>H1</u> <u>S1</u> <u>C1</u> <u>A1</u>	ippiHSVToRGB_16s_AC4R

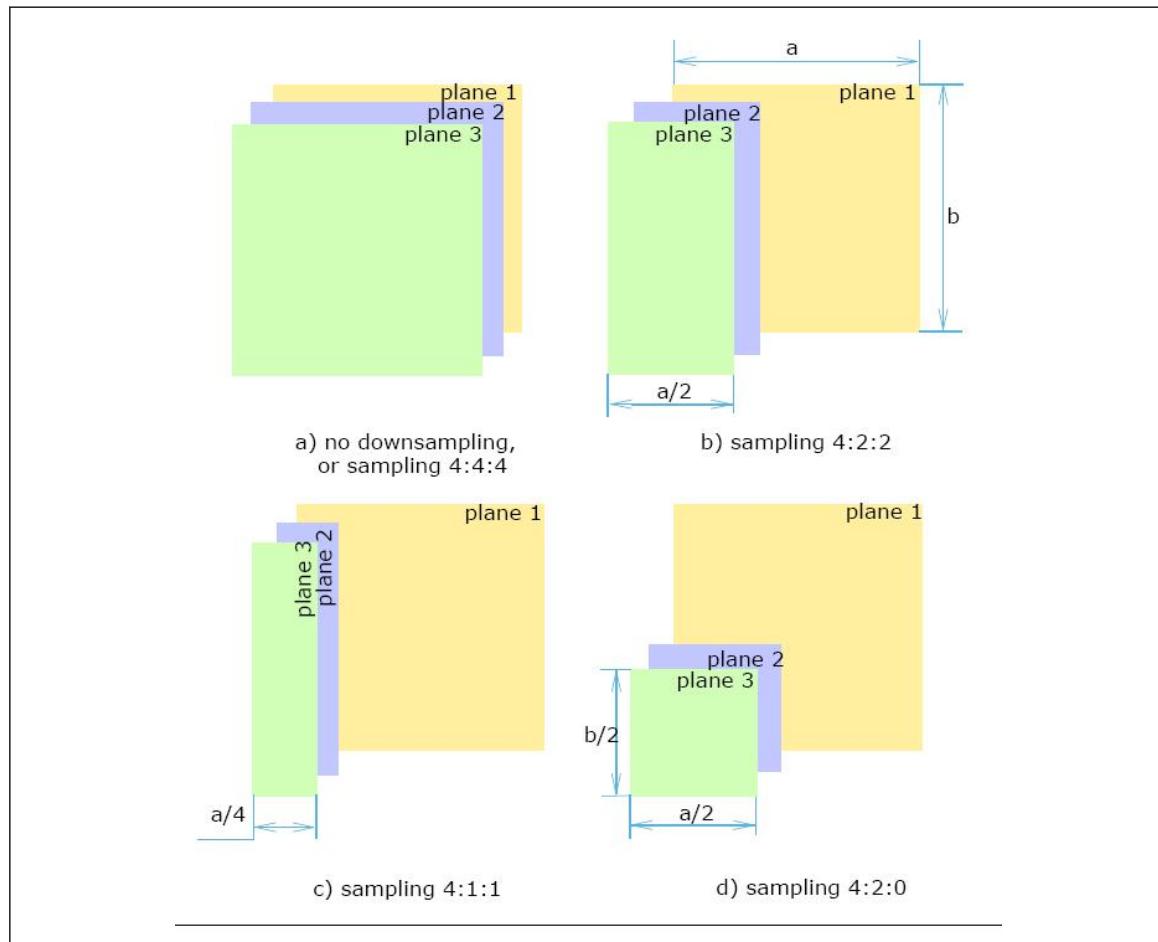
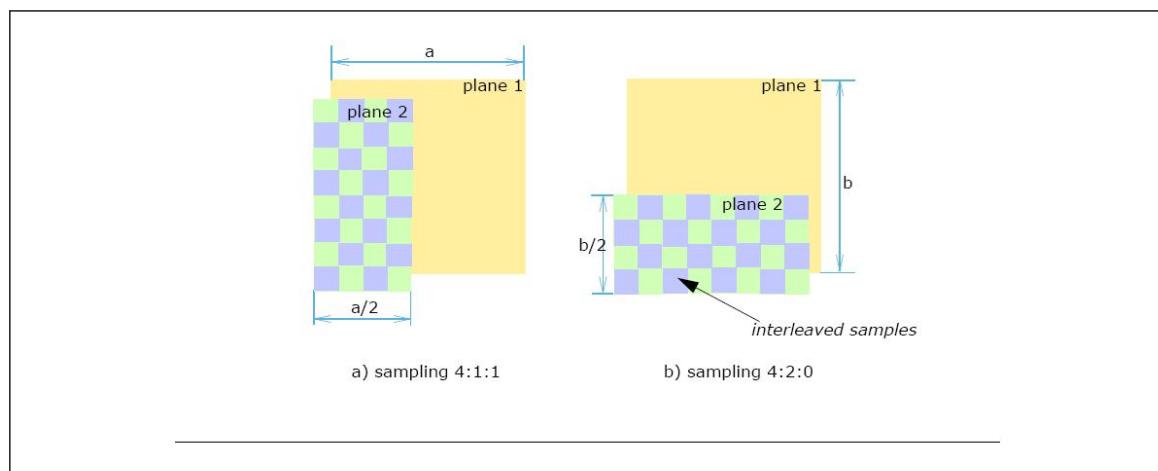
Planar image formats supported by the Intel IPP color conversion functions are listed in Table "Planar Image Formats" along with examples of the Intel IPP functions using that format. Planes layout and their relative sizes are shown in [Figure Plane Size and Layout: 3-planes Images](#) and [Figure Plane Size and Layout: 2-planes Images](#).

### Planar Image Formats

Image Format	Number of Planes	Planes Layout			Example	
		pl. 1	pl. 2	pl. 3		
RGB	3	R	G	B	see <a href="#">Figure below</a> , a	ippiRGBToYUV_8u_P3R
YUV	3	Y	U	V	see <a href="#">Figure below</a> , a	ippiYUVToRGB_8u_P3R
4:2:2 YUV	3	Y	U	V	see <a href="#">Figure below</a> , b	ippiYUV422ToRGB_8u_P3

Image Format	Number of Planes	Planes Layout			Example
		pl. 1	pl. 2	pl. 3	
4:2:0 YUV	3	Y	U	V	see Figure below, d ippiYUV420ToRGB_8u_P3C3R
YCbCr	3	Y	Cb	Cr	see Figure below, a ippiYCbCrToRGB_8u_P3R
4:2:2 YCbCr	3	Y	Cb	Cr	see Figure below, b ippiYCbCr422_8u_P3C2R
4:1:1 YCbCr	3	Y	Cb	Cr	see Figure below, c ippiYCbCr411_8u_P3P2R
4:1:1 YCbCr	2	Y	CbCr	-	see Figure below for 2-planes images, a ippiYCbCr411_8u_P2P3R
4:2:0 YCbCr	3	Y	Cb	Cr	see Figure below, d ippiRGBToYCbCr420_8u_C3P3R
4:2:0 YCbCr	2	Y	CbCr	-	see Figure below for 2-planes images, b ippiYCbCr420_8u_P2P3R

Image Format	Number of Planes	Planes Layout			Example
		pl. 1	pl. 2	pl. 3	
4:2:0 YCrCb	3	Y	Cr	Cb	ippiYCrCb420ToYCbCr422_8u_P3R see Figure below, d

**Plane Size and Layout - 3-planes Images****Plane Size and Layout - 2-planes Images**

## Color Model Conversion

---

### RGBToYUV

Converts an RGB image to the YUV color model.

---

#### Syntax

##### Case 1: Operation on pixel-order data

```
IppStatusippiRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C3R      8u\_AC4R

##### Case 2: Operation on planar data

```
IppStatusippiRGBToYUV_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);
```

##### Case 3: Conversion from pixel-order to planar data

```
IppStatusippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);
```

```
IppStatusippiRGBToYUV_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int
dstStep, IppiSize roiSize);
```

#### Include Files

ippcc.h

#### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image  $pSrc$  to the Y'U'V' image  $pDst$  (see [Figure Converting an RGB image to YUV](#)) according to the following formulas:

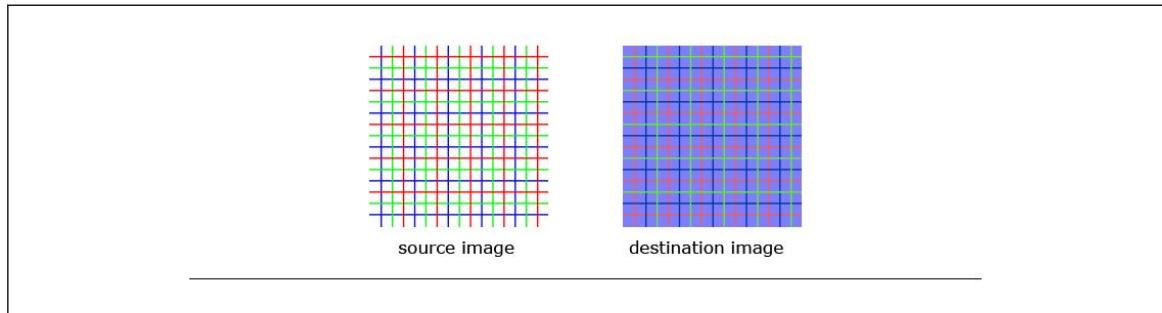
$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B'$$

$$U' = -0.147 \cdot R' - 0.289 \cdot G' + 0.436 \cdot B' = 0.492 \cdot (B' - Y')$$

$$V' = 0.615 \cdot R' - 0.515 \cdot G' - 0.100 \cdot B' = 0.877 \cdot (R' - Y')$$

For digital RGB values in the range [0..255],  $Y'$  has the range [0..255],  $U$  varies in the range [-112..+112], and  $V$  in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed  $U$  and  $V$  values, and  $V$  is then saturated.

### Converting an RGB image to YUV



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if $pSrc$ or $pDst$ pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if $roiSize$ has a field with a zero or negative value.

### Example

The code example below shows how to use the function `ippiRGBToYUV_8u_C3R`.

```
# define nChannels 3

int main () {
    Ipp 8 u src [3*3* nChannels ] = {
        255, 0, 0, 255, 0, 0, 255, 0, 0,
        0, 255, 0, 0, 255, 0, 0, 255, 0,
        0, 0, 255, 0, 0, 255, 0, 0, 255};
    Ipp 8 u dst [3*3* nChannels ];
    IppiSize roiSize = { 3, 3 };
    IppStatus st = ippStsNoErr ;
    int srcStep = 3* nChannels ;
    int dstStep = 3* nChannels ;
    st = ippiRGBToYUV _8 u _ C 3 R ( src , srcStep , dst , dstStep , roiSize );
    if ( st == ippStsNoErr){
        printf("\n ***** passed *****\n");
    }else{
        printf("\n ***** failed *****\t");
    }
    return 0;
}
```

**Result:**

```

255   0   0 255   0   0 255   0   0
  0 255   0   0 255   0   0 255   0     src
  0   0 255   0   0 255   0   0 255

76   90 255   76   90 255   76   90 255
149  54   0 149  54   0 149  54   0     dst
 29 239 102   29 239 102   29 239 102

```

**YUVToRGB***Converts a YUV image to the RGB color model.***Syntax****Case 1: Operation on pixel-order data**

```
IppStatusippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

8u\_C3R  
8u\_AC4R

```
IppStatusippiYUVToRGB_8u_C3C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize, Ipp8u aval);
```

**Case 2: Operation on planar data**

```
IppStatusippiYUVToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);
```

**Case 3: Conversion from planar to pixel-order data**

```
IppStatusippiYUVToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.

---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>aval</i>	Constant value to create the fourth channel.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the  $Y'U'V'$  image *pSrc* to the gamma-corrected  $R'G'B'$  image *pDst* according to the following formulas:

$$\begin{aligned} R' &= Y' + 1.140 \cdot V' \\ G' &= Y' - 0.394 \cdot U' - 0.581 \cdot V' \\ B' &= Y' + 2.032 \cdot U' \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYUV422

Converts an RGB image to the YUV color model; uses 4:2:2 sampling.

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data with ROI

```
IppStatusippiRGBToYUV422_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 3: Operation on planar data without ROI

```
IppStatusippiRGBToYUV422_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

### Case 4: Conversion from pixel-order to planar data with ROI

```
IppStatusippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 5: Conversion from pixel-order to planar data without ROI

```
IppStatusippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* with **4:2:2 sampling** format, according to the same formulas as the function `ippiRGBToYUV` does. For more details on this sampling format, see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#).

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], the constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operate with ROI (see [Regions of Interest in Intel IPP](#)). The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.

## YUV422ToRGB

*Converts a YUV image with the 4:2:2 sampling to the RGB color model.*

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Operation on planar data with ROI**

```
IppStatusippiYUV422ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

**Case 3: Operation on planar data without ROI**

```
IppStatusippiYUV422ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize
imgSize);
```

**Case 4: Conversion from planar to pixel-order data with ROI**

```
IppStatusippiYUV422ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_P3C3R 8u\_P3AC4R

**Case 5: Conversion from planar to pixel-order data without ROI**

```
IppStatusippiYUV422ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

**Description**

This function converts the **Y'U'V'** image *pSrc* to the gamma-corrected **R'G'B'** image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV422ToRGB4:2:0 sampling` the input data to be in `4:2:2 sampling` format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

The function `ippiYUV422ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operate with ROI (see [Regions of Interest in Intel IPP](#)). The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.

## RGBToYUV420

*Converts an RGB image to the 4:2:0 YUV image.*

### Syntax

#### Case 1: Operation on planar data with ROI

```
IppStatusippiRGBToYUV420_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 2: Operation on planar data without ROI

```
IppStatusippiRGBToYUV420_8u_P3(const Ipp8u*pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

#### Case 3: Conversion from pixel-order to planar data with ROI

```
IppStatusippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 4: Conversion from pixel-order to planar data without ROI

```
IppStatusippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffers in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
<i>pDst</i>	An array of pointers to the destination image buffers in each color plane.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the destination image for operations with ROI.

---

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* with the 4:2:0 sampling (see Table “Planar Image Formats” for more details). The conversion is performed in the accordance with the same formulas as the function `ippiRGBToYUV` does.

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI see [Regions of Interest in Intel IPP](#).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

*roiSize.width (imgSize.width)* and *roiSize.height (imgSize.height)* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> or <i>imgSize</i> has a field that is not a multiple of 2.

## YUV420ToRGB

Converts a YUV image that has 4:2:0 sampling format to the RGB image.

## Syntax

### Case 1: Operation on planar data with ROI

```
IppStatusippiYUV420ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3],  
int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data without ROI

```
IppStatusippiYUV420ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize  
imgSize);
```

### Case 3: Conversion from planar to pixel-order data with ROI

```
IppStatusippiYUV420ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int  
dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_P3C3R 8u\_P3AC4R

**Case 4: Conversion from planar to pixel-order data without ROI**

```
IppStatusippiYUV420ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	An array of pointers to the source image buffers in each color plane.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

**Description**

This function converts the **Y'U'V'** image *pSrc* to the gamma-corrected **R'G'B'** image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV420ToRGB4:2:0 sampling` the input data to be in **4:2:2 sampling** format (see [Table “Planar Image Formats”](#) for more details).

The function `ippiYUV420ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operate with ROI see [Regions of Interest in Intel IPP](#) ).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

*roiSize.width* (*imgSize.width*) and *roiSize.height* (*imgSize.height*) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.

---

ippStsDoubleSize	Indicates a warning if <i>roiSize</i> or <i>imgSize</i> has a field that is not a multiple of 2.
------------------	--

## BGRToYUV420

Converts an BGR image to the YUV color model; uses 4:2:0 sampling

---

### Syntax

```
IppStatusippiBGRToYUV420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to the destination image buffers in each color plane.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected **'G' 'R'** image *pSrc* to the **'Y' 'U' 'V'** image *pDst* with the [4:2:0 sampling](#) (see [Table "Planar Image Formats"](#) for more details). The function uses the same formulas as the function [ippiRGBToYUV](#) does.

For digital BGR values in the range [0..255], Y' varies in the range [0..255], U - in the range [-112..+112], and V - in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to the computed U and V values, and V is then saturated.

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with a zero or negative value.

ippStsDoubleSize	Indicates a warning if <i>roiSize</i> or <i>imgSize</i> has a field that is not a multiple of 2.
------------------	--

## YUV420ToBGR

Converts a YUV image that has 4:2:0 sampling to the BGR image.

---

### Syntax

```
IppStatusippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Y'U'V'** image *pSrc* to the gamma-corrected **B'G'R'** image *pDst* according to the same formulas as the function [ippiYUVToRGB](#) does. The input data must be presented in the **4:2:0 sampling** format (see [Table “Planar Image Formats”](#) for more details).

*roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## **YUV422v210ToRGB, YUV422v210ToBGR**

Converts a YUV422 (v210) image to a RGB/BGR image for ITU-R BT.709 HDTV signal.

### Syntax

```
IppStatusippiYUV422v210ToRGB_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep,
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiYUV422v210ToBGR_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep,
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the YUV image *pSrc*, packed in the 4:2:2 sampling format, to the gamma-corrected RGB/BGR image *pDst* for digital component video signals in compliance with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UYV|YUY|VYU|YYV, ... . The conversion is performed according to the following formulas:

$$R = Y + 1.540 * (V - 512)$$

$$G = Y - 0.459 * (V - 512) - 0.183 * (U - 512)$$

$$B = Y + 1.816 * (U - 512)$$

The output RGB/BGR values are saturated to the range R [0..31], G [0..63], B [0..31].

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YUV422v210ToGray

*Converts a YUV422 (v210) image to a grayscale image for ITU-R BT.709 HDTV signal*

---

### Syntax

```
IppStatusippiYUV422v210ToGray_709HDTV_32u16u_C3C1(const Ipp32u* pSrc, int srcStep,  
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the YUV image *pSrc*, packed in the 4:2:2 sampling format, to the grayscale 16U\_C1 image *pDst* for digital component video signals in compliance with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UYV|YUY|VYU|YVY, ... .

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCbCr

*Converts an RGB image to the YCbCr color model.*

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3R    8u\_AC4R

### Case 2: Operation on planar data

```
IppStatusippiRGBToYCbCr_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Conversion from pixel-order to planar data

```
IppStatusippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u\_C3P3R    8u\_AC4P3R

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for a pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination immage ROI. Array of pointers to ROI in the separate destination color planes for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* with values in the range [0..255] to the Y'Cb'Cr' image *pDst* according to the following formulas:

$$Y' = 0.257 \cdot R' + 0.504 \cdot G' + 0.098 \cdot B' + 16$$

$$C_b' = -0.148 \cdot R' - 0.291 \cdot G' + 0.439 \cdot B' + 128$$

$$C_r' = 0.439 \cdot R' - 0.368 \cdot G' - 0.071 \cdot B' + 128$$

In the YCbCr model, Y is defined to have a nominal range [16..235], while Cb and Cr are defined to have a range [16..240], with the value of 128 as corresponding to zero.

Both the source and destination images have the same bit depth.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToRGB

*Converts a YCbCr image to the RGB color model.*

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatusippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3R      8u\_AC4R

#### Case 2: Operation on planar data

```
IppStatusippiYCbCrToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatusippiYCbCrToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 3: Conversion from planar to pixel-order data

```
IppStatusippiYCbCrToRGB_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. Array of pointers to the ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. Array of pointers to the ROI in the separate destination color planes for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

---

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Y'Cb'Cr'** image *pSrc* to the 24-bit gamma-corrected **R'G'B'** image *pDst*. The following formulas are used for conversion:

$$R' = 1.164 * (Y' - 16) + 1.596 * (Cr' - 128)$$

$$G' = 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128)$$

$$B' = 1.164 * (Y' - 16) + 2.017 * (Cb' - 128)$$

The output **R'G'B'** values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCrToBGR

*Converts a YCbCr image to the BGR color model.*

### Syntax

```
IppStatusippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

`ippcc.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* to the 24-bit gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does. The output `B'G'R'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

## YCbCrToBGR\_709CSC

Converts a YCbCr image to the BGR image for ITU-R BT.709 CSC signal.

---

## Syntax

```
IppStatusippiYCbCrToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiYCbCrToBGR_709CSC_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

---

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar `Y'Cb'Cr'` image *pSrc* to the three- or four-channel gamma-corrected `B'G'R'` image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = 1.164 * (Y' - 16) + 1.793 * (Cr' - 128)$$

$$G' = 1.164 * (Y' - 16) - 0.534 * (Cr' - 128) - 0.213 * (Cb' - 128)$$

$$B' = 1.164 * (Y' - 16) + 2.115 * (Cb' - 128)$$

The output `R'G'B'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCbCr422

Converts an RGB image to the YCbCr image with 4:2:2 sampling.

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data

```
IppStatusippiRGBToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 2: Conversion from planar to pixel-order data

```
IppStatusippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order image. An array of pointer to ROI in each separate planes for the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* with [4:2:2 sampling](#) (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details). The conversion is performed according to the same formulas as the function [`ippiRGBToYCbCr`](#) does.

The converted buffer for pixel-order image has the reduced bit depth of a 16 bits per pixel, whereas the source buffer has 24 bit depth.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCbCr422ToRGB

*Converts an YCbCr image with the 4:2:2 sampling to the RGB image.*

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiYCbCr422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Case 2: Conversion from pixel-order to planar data

```
IppStatusippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Conversion from planar to pixel-order data

```
IppStatusippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the Y'Cb'Cr' image *pSrc* with the 4:2:2 sampling (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function [ippiYCbCrToRGB](#) does. The output R'G'B' values are saturated to the range [0..255].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the `ippiYCbCr422ToRGB_8u_C2C4R` function.

```
const int WIDTH = 2;
const int HEIGHT = 2;

Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{
    236,50,236,80,
    236,50,236,80,
};

Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
```

```
Ipp8u alphaValue = 0xFF;
IppStatus status =ippiYCbCr422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize,
alphaValue);
if (status ==ippStsNoErr)
    printf("\n ***** passed *****\n");
else
    printf("\n ***** failed *****\n");
```

## RGBToYCrCb422

*Converts 24-bit per pixel RGB image to 16-bit per pixel YCrCb image*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatusippiRGBToYCrCb422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar to pixel-order data

```
IppStatusippiRGBToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* according to the same formulas as the function [ippiRGBToYCbCr](#) does. The difference is that [ippiRGBToYCrCb422](#) uses 4:2:2 sampling format for the converted image (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#) for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## YCrCb422ToRGB, YCrCb422ToBGR

Convert 16-bit per pixel YCrCb image to 24 or 32-bit per pixel RGB or BGR image.

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatusippiYCrCb422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCrCb422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatusippiYCrCb422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCrCb422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatusippiYCrCb422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source plane for planar images.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each plane of the destination planar image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

`aval` Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cr'Cb'` image `pSrc`, packed in `4:2:2 sampling` format (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#) for more details) to the 24-bit gamma-corrected `R'G'B'` or `B'G'R'` image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does. The output `R'G'B'` values are saturated to the range [0..255]. `Y'Cr'Cb'` image with `4:2:2` sampling is also known as YVYU format.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the `ippiYCrCb422ToRGB_8u_C2C4R` function.

```
#define WIDTH 2
#define HEIGHT 2

Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{
    236,50,236,80,
    236,50,236,80,
};

Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCrCb422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize,
alphaValue);
if (status == ippStsNoErr)
    printf("\n ***** passed *****\n");
else
    printf("\n ***** failed *****\n");
```

## BGRToYCbCr422

Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

## Case 2: Conversion from pixel-order to planar data

```
IppStatusippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);  
  
IppStatusippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source mage ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected **'G'-'R'** image *pSrc* to the two-channel or three-planes **'Y'-'Cb'-'Cr'** image *pDst* according to the same formulas as the function [ippiRGBToYCbCr](#) does. The difference is that [ippiBGRToYCbCr422](#) uses the [4:2:2 sampling](#) format (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

## YCbCr422ToBGR

Converts a YCbCr image with 4:2:2 sampling to the BGR image.

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Case 2: Conversion from planar to pixel-order data

```
IppStatusippiYCbCr422ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI. An array of pointers to the ROI in each separate plane of the source planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of such distances in bytes for each plane of the source planar image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **'Y'Cb'Cr'** image *pSrc* with **4:2:2 sampling** (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the gamma-corrected **'B'G'R'** image *pDst* according to the same formulas as the function [ippiYCbCrToRGB](#) does.

The output **'B'G'R'** values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

## YCbCr422ToGray

Converts an interlaced 4:2:2 YCbCr or YCrCb image to gray-scale extracting luminance (Y) component.

### Syntax

```
IppStatusippiYCbCr422ToGray_8u_C2C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an interlaced **Y'Cb'Cr'** or **Y'Cr'Cb'** image *pSrc* with the **4:2:2 sampling** (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details) to the gray-scale image *pDst* extracting luminance (Y) component.

**Y'Cb'Cr'** image with 4:2:2 sampling is also known as YUY2 format, and **Y'Cr'Cb'** as YVYU.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

### Example

The code example below demonstrates how to use the `ippiYCbCr422ToGray_8u_C2C1R` function.

```
const int WIDTH = 2;
const int HEIGHT = 2;

Ipp8u pSrc[WIDTH * HEIGHT * 2] = {
    190, 70, 191, 80,
```

```

    200,71,201,81,
};

Ipp8u pDst[WIDTH * HEIGHT];
int srcStep = WIDTH * 2, dstStep = WIDTH;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status =ippiYCbCr422ToGray_8u_C2C1R(pSrc, srcStep, pDst, dstStep, roiSize);
if (status == ippStsNoErr)
    printf("PASS:\n%3d %3d\n%3d %3d\n", pDst[0], pDst[1], pDst[2], pDst[3]);
else
    printf("FAIL: status = %d\n", status);

```

**Result:**

```
PASS:
190 191
200 201
```

## RGBToCbYCr422, RGBToCbYCr422Gamma

*Convert 24-bit per pixel RGB image to 16-bit per pixel Cb'Y'Cr' image.*

---

### Syntax

```
IppStatusippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiRGBToCbYCr422` converts the gamma-corrected R'G'B' image *pSrc* to the Cb'Y'Cr' image *pDst* according to the same formulas as the function [ippiRGBToYCbCr](#) does.

The function `ippiRGBToCbYCr422Gamma` performs gamma-correction of the source RGB image `pSrc` according to the same formula as the function `ippiGammaFwd` does, and then converts it to the `Cb'Y'Cr'` image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` does.

The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` use `4:2:2 sampling` format for the converted image.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...`

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## CbYCr422ToRGB

*Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.*

### Syntax

```
IppStatusippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Cb'Y'Cr'` image `pSrc`, packed in the `4:2:2 sampling` format, to the 24-bit gamma-corrected `R'G'B'` image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...`.

The output `R'G'B'` values are saturated to the range [0..255].

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGRTToCbYCr422

Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.

---

### Syntax

```
IppStatusippiBGRTToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* according to the same formulas as the function [ippiRGBToYCbCr](#) does. The function [ippiBGRTToCbYCr422](#) uses [4:2:2 sampling](#) format for the converted image. An alpha-channel information is lost.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.

---

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
---------------	---

## BGRTToCbYCr422\_709HDTV

Converts *BGR* image to 16-bit per pixel *CbYCr* image for ITU-R BT.709 HDTV signal.

---

### Syntax

```
IppStatusippiBGRTToCbYCr422_709HDTV_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiBGRTToCbYCr422_709HDTV_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the three- or four-channel gamma-corrected *B'G'R'* image *pSrc* to the two-channel *Cb'Y'Cr'* image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [[ITU709](#)] for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas [[Jack01](#)]:

$$\begin{aligned} Y' &= 0.213 \cdot R' + 0.715 \cdot G' + 0.072 \cdot B' \\ Cb' &= -0.117 \cdot R' - 0.394 \cdot G' + 0.511 \cdot B' + 128 \\ Cr' &= 0.511 \cdot R' - 0.464 \cdot G' - 0.047 \cdot B' + 128 \end{aligned}$$

The values of *Y'* of the destination image are in the range [16..235], the values of *Cb'*, *Cr'* are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The function `ippiBGRTToCbYCr422_709HDTV` uses the [4:2:2 sampling](#) format for the converted image. The alpha-channel information is lost.

A *CbYCr* image has the following sequence of bytes: *Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## CbYCr422ToBGR

Converts 16-bit per pixel CbYCr image to four channel BGR image.

---

### Syntax

```
IppStatusippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Cb'Y'Cr'** image *pSrc*, packed in [4:2:2 sampling](#) format, to the four channel gamma-corrected **B'G'R'** image *pDst* according to the same formulas as the function [ippiYCbCrToRGB](#) does.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

The output **B'G'R'** values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## CbYCr422ToBGR\_709HDTV

Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.

### Syntax

```
IppStatusippiCbYCr422ToBGR_709HDTV_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatusippiCbYCr422ToBGR_709HDTV_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Cb'Y'Cr'** image *pSrc*, packed in [4:2:2 sampling](#) format, to the three- or four-channel gamma-corrected **B'G'R'** image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for high-definition TV (HDTV). A source CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... . The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$R' = Y' + 1.540 * (Cr' - 128)$$

$$G' = Y' - 0.459 * (Cr' - 128) - 0.183 * (Cb' - 128)$$

```
B' = Y' + 1.816*(Cb' - 128)
```

The destination image pixel values have a nominal range [16..235]. The resulting R'G'B' values should be saturated at the 0 and 255 levels.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToYCbCr420

Converts an RGB image to the YCbCr color model;  
uses 4:2:0 sampling.

---

### Syntax

```
IppStatusippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);  
  
IppStatusippiRGBToYCbCr420_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int  
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);  
  
IppStatusippiRGBToYCbCr420_8u_C4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int  
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in the separate planes of the destination image for three-plane image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image for three-plane image.
<code>pDstY</code>	Pointer to ROI in the luminance plane of the destination image for two-plane image.
<code>dstStep</code> , <code>dstYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane for two-plane image.

---

<i>pDstCbCr</i>	Pointer to ROI in the interleaved chrominance plane of the destination image for two-plane image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane for two-plane image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image according to the same formulas as the function [ippiRGBToYCbCr](#) does. The difference is that [ippiRGBToYCbCr420](#) uses 4:2:0 sampling format for the converted image (see [Table "Planar Image Formats"](#) for more details).

*roiSize.width* should be multiple of 2, and *roiSize.height* should be multiple of 2 (for three-plane image) or 4 (for two-plane image). If not the function reduces their original values to the nearest multiples of 2 or 4 correspondingly, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with value less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize.width</i> is not a multiple of 2, or if <i>roiSize.height</i> is not a multiple of 2 (for three-plane image) or 4 (for two-plane image).

## YCbCr420ToRGB, YCbCr420ToBGR

Convert a YCbCr image that has 4:2:0 sampling format to the RGB or BGR color model.

---

## Syntax

```
IppStatusippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiYCbCr420ToRGB_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCbCr420ToRGB_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatusippiYCbCr420ToBGR_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCbCr420ToBGR_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the three-plane source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the three-plane source image.
<i>pSrcY</i>	Pointer to ROI in the luminance plane of the two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the two-plane source image.
<i>pSrcCbCr</i>	Pointer to ROI in the interleaved chrominance plane of the two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the two-plane source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `R'G'B'` or `B'G'R'` image *pDst* according to the same formulas as the function [ippiYCbCrToRGB](#) does. The difference is that the [ippiYCbCr420ToRGB](#) and [ippiYCbCr420ToBGR](#) functions use the input data in the `4:2:0 sampling` format, in which the number of `Cb` and `Cr` samples is reduced by half in both vertical and horizontal directions (see [Table "Planar Image Formats"](#) for more details). Two-plane `Y'Cb'Cr'` image with `4:2:0` sampling is also known as NV12 format.

The value of `roiSize.width` and `roiSize.height` must be a multiple of 2. Otherwise, the function reduces original values to the nearest multiples of 2, performs operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with value less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## Example

The code example below demonstrates how to use the `ippiYCbCr420ToRGB_8u_P2C4R` function.

```
#define WIDTH 4
#define HEIGHT 4

Ipp8u pSrcY[WIDTH * HEIGHT] =
```

```

{
    236,236,236,236,
    236,236,236,236,
    236,236,236,236,
    236,236,236,236
};

Ipp8u pSrcCbCr[WIDTH * HEIGHT / 2] =
{
    128,128,128,128,
    128,128,128,128
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcYStep = WIDTH, srcCbCrStep = WIDTH, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status =ippiYCbCr420ToRGB_8u_P2C4R(pSrcY, srcYStep, pSrcCbCr, srcCbCrStep,
pDstRGB, dstStep, roiSize, alphaValue);
if (status == ippStsNoErr)
    printf("\n ***** passed *****\n");
else
    printf("\n ***** failed *****\n");

```

## RGBToYCrCb420

Converts an RGB image to the YCrCb image with 4:2:0 sampling format.

### Syntax

```
IppStatusippiRGBToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h, ipp.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a four-channel gamma-corrected R'G'B' image *pSrc* to the planar Y'Cr'Cb' image *pDst* with the 4:2:0 sampling (see [Table “Planar Image Formats”](#) for more details). The conversion is performed according to the same formulas as the function [ippiRGBToYCbCr](#) does.

*roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCrCb420ToRGB, YCrCb420ToBGR

Convert a YCrCb image with the 4:2:0 sampling to the RGB or BGR image.

---

## Syntax

```
IppStatusippiYCrCb420ToRGB_8u_P3C4R (const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatusippiYCrCb420ToRGB_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb,
int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatusippiYCrCb420ToRGB_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb,
int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCrCb420ToBGR_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb,
int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCrCb420ToBGR_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb,
int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatusippiYCrCb420ToBGR_Filter_8u_P3C4R (const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

*pSrc* An array of pointers to ROI in separate planes of the three-plane source image.

<i>srcStep</i>	An array of distances, in bytes, between the starting points of consecutive lines in each plane of the three-plane source image.
<i>pSrcY</i>	Pointer to ROI in the luminance plane of the two-plane source image.
<i>srcYStep</i>	Distance, in bytes, between the starting points of consecutive lines in the luminance plane of the two-plane source image.
<i>pSrcCrCb</i>	Pointer to ROI in the interleaved plane chrominance plane of the two-plane source image.
<i>srcCrCbStep</i>	Distance, in bytes, between the starting points of consecutive lines in the interleaved chrominance plane of the two-plane source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI.

This function converts the `Y'Cr'Cb'` image *pSrc* to the gamma-corrected `R'G'B'` or `B'G'R'` image *pDst* according to the same formulas as the `ippiippiYCbCrToRGB` function does. The difference is that the `ippiYCrCb420ToRGB` and `ippiYCrCb420ToBGR` functions use the source data in the `4:2:0 sampling` format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see [Table "Planar Image Formats"](#) for more details). Two-plane `Y'Cr'Cb` image with `4:2:0` sampling is also known as NV21 format.

The value of `roiSize.width` and `roiSize.height` must be a multiple of 2. Otherwise, the function reduces original values of `roiSize.width` and `roiSize.height` to the nearest multiples of 2, performs operation, and returns a warning.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a value less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## Example

The code example below demonstrates how to use the `ippiYCrCb420ToRGB_8u_P2C4R` function.

```
static void sampleNV21ToRGBA()
{
    Ipp8u pY[4*4]=
    {
        236,236,236,236,
        236,236,236,236,
```

```

236,236,236,236,
236,236,236,236
};

Ipp8u pCbCr[4*2]=
{
    128,128,128,128,
    128,128,128,128
};

Ipp8u pRGB[(4*4)*4];
int YStep = 4, CbCrStep = 4, rgbStep = 4*4;
IppiSize roiSize = {4,4};
Ipp8u alpha = 0xFF;
IppStatus status =ippiYCrCb420ToRGB_8u_P2C4R(pY, YStep, pCbCr, CbCrStep, pRGB, rgbStep,
roiSize, alpha);
if (status == ippStsNoErr)
    printf("\n ***** passed *****\n");
else
    printf("\n ***** failed *****\t");
}

```

## See Also

[Regions of Interest in Intel IPP](#)

## BGRToYCbCr420

*Converts a BGR image to the YCbCr image with 4:2:0 sampling format.*

---

### Syntax

```

IppStatusippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);

IppStatusippiBGRToYCbCr420_8u_C3P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int
yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);

IppStatusippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);

IppStatusippiBGRToYCbCr420_8u_AC4P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int
yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);

```

### Include Files

ippcc.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc, pRGB</i>	Pointer to the source image ROI.
<i>pY</i>	Pointer to the image Y plane.
<i>srcStep, rgbStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.

<i>yStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image Y plane.
<i>pDst, pCbCr</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep, cbCrStep</i>	An array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image to the planar `Y'Cb'Cr'` image according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr420` uses `4:2:0` sampling format (see [Table "Planar Image Formats"](#) for more details).

`roiSize.width` and `roiSize.height` should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> ( <code>pRGB</code> ) or <code>pDst</code> ( <code>pCbCr</code> ) is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize</code> has a field that is not a multiple of 2.

## BGRToYCbCr420\_709CSC

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

## Syntax

```
IppStatusippiBGRToYCbCr420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiBGRToYCbCr420_709CSC_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatusippiBGRToYCbCr420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiBGRToYCbCr420_709CSC_8u_AC4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i> , <i>pDstY</i> , <i>pDstCbCr</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i> , <i>dstYStep</i> , <i>dstCbCrStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16$$

$$C_b' = -0.101*R' - 0.338*G' + 0.439*B' + 128$$

$$C_r' = 0.439*R' - 0.399*G' - 0.040*B' + 128$$

The destination image *pDst* has [4:2:0 sampling](#) format (see [Table “Planar Image Formats”](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGRToYCbCr420\_709HDTV

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.

## Syntax

```
IppStatusippiBGRToYCbCr420_709HDTV_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in separate planes of the destination image.
<code>dstStep</code>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a four-channel gamma-corrected B'G'R' image `pSrc` to the planar `Y'Cb'Cr'` image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$Y' = 0.213*R' + 0.715*G' + 0.072*B'$$

$$C_{b'} = -0.117*R' - 0.394*G' + 0.511*B' + 128$$

$$C_{r'} = 0.511*R' - 0.464*G' - 0.047*B' + 128$$

The values of `Y'` of the destination image are in the range [16..235], the values of `Cb'`, `Cr'` are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The destination image `pDst` has the `4:2:0` sampling format (see [Table “Planar Image Formats”](#) for more details).

The values of `roiSize.width` and `roiSize.height` should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 2.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize</code> has a field that is not a multiple of 2.

## BGRToYCrCb420\_709CSC

Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

## Syntax

```
IppStatusippiBGRToYCrCb420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatusippiBGRToYCrCb420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [\[Jack01\]](#):

$$Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16$$

$$Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128$$

$$Cr' = 0.439*R' - 0.399*G' - 0.040*B' + 128$$

The destination image *pDst* has [4:2:0 sampling](#) format (see [Table “Planar Image Formats”](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

---

ippstsDoubleSize	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.
------------------	--

## YCbCr420ToBGR

Converts a YCbCr image with the 4:2:0 sampling to the BGR image.

---

### Syntax

```
IppStatusippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiYCbCr420ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the Y'Cb'Cr' image *pSrc* with the 4:2:0 sampling (see [Table "Planar Image Formats](#) for more details) to the gamma-corrected three- or four-channel B'G'R' image *pDst*. The conversion is performed according to the same formulas as the function [ippiYCbCrToRGB](#) does.

Fourth channel is created by setting channel values to the constant value *aval*.

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

### Return Values

ippstsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippstsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCbCr420ToBGR\_709CSC

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.

---

### Syntax

```
IppStatusippiYCbCr420ToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar `Y'Cb'Cr'` image *pSrc* to the three-channel gamma-corrected `B'G'R'` image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [\[ITU709\]](#) for computer systems consideration (CSC). The conversion is performed according to the following formulas [\[Jack01\]](#):

$$\begin{aligned} R' &= 1.164 * (Y' - 16) + 1.793 * (Cr' - 128) \\ G' &= 1.164 * (Y' - 16) - 0.534 * (Cr' - 128) - 0.213 * (Cb' - 128) \\ B' &= 1.164 * (Y' - 16) + 2.115 * (Cb' - 128) \end{aligned}$$

The output `R'G'B'` values are saturated to the range [0..255]. The source image *pDst* has the [4:2:0 sampling](#) format (see [Table "Planar Image Formats"](#) for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## YCbCr420ToBGR\_709HDTV

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.

### Syntax

```
IppStatusippiYCbCr420ToBGR_709HDTV_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a planar Y'Cb'Cr' image *pSrc* to the four-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [[ITU709](#)] for high-definition TV (HDTV). The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas [[Jack01](#)]:

$$R' = Y' + 1.540 * (Cr' - 128)$$

$$G' = Y' - 0.459 * (Cr' - 128) - 0.183 * (Cb' - 128)$$

$$B' = Y' + 1.816 * (Cb' - 128)$$

The destination image pixel values have a nominal range [16..235]. The resulting R'G'B' values should be saturated at the 0 and 255 levels.

The source image *pDst* has the 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize</i> has a field that is not a multiple of 2.

## BGRToYCrCb420

Converts a BGR image to the YCrCb image with 4:2:0 sampling format.

---

### Syntax

```
IppStatusippiBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);  
  
IppStatusippiBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected **B'G'R'** image *pSrc* to the planar **Y'Cr'Cb'** image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image *pDst* has the `4:2:0 sampling` format and the following order of pointers: *Y*-plane, *Cr*-plane, *Cb*-plane (see [Table "Planar Image Formats"](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

## BGRToYCbCr411

*Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.*

### Syntax

```
IppStatusippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);  
  
IppStatusippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],  
int dstStep[3], IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channel gamma-corrected **B'G'R'** image *pSrc* to the planar **Y'Cb'Cr'** image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr411` uses the `4:1:1 sampling` format (see [Table "Planar Image Formats"](#) for more details).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

## YCbCr411ToBGR

Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.

---

## Syntax

```
IppStatusippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the planar 'Y'Cb'Cr' image *pSrc* to the three- or four-channel image *pDst*. To compute gamma-corrected R'G'B' (B'G'R') channel values the above formulas are used. The difference is that `ippiYCbCr411ToBGR` uses the input data in the 4:1:1 sampling format (see [Table "Planar Image Formats"](#) for more details). Fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
-------------	--

---

ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToXYZ

Converts an RGB image to the XYZ color model.

### Syntax

```
IppStatusippiRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image *pSrc* to the CIEXYZ image *pDst* according to the following basic equations:

$$X = 0.412453*R + 0.35758 *G + 0.180423*B$$

$$Y = 0.212671*R + 0.71516 *G + 0.072169*B$$

$$Z = 0.019334*R + 0.119193*G + 0.950227*B$$

The equations above are given on the assumption that R,G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed XYZ values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges" in Chapter 2](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## XYZToRGB

*Converts an XYZ image to the RGB color model.*

### Syntax

```
IppStatusippiXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the CIEXYZ image *pSrc* to the RGB image *pDst* according to the following basic equations:

$$\begin{aligned} R &= 3.240479 * X - 1.53715 * Y - 0.498535 * Z \\ G &= -0.969256 * X + 1.875991 * Y + 0.041556 * Z \\ B &= 0.055648 * X - 0.204043 * Y + 1.057311 * Z \end{aligned}$$

The equations above are given on the assumption that *X*, *Y*, and *Z* values are in the range [0..1]. In case of the floating-point data type, the input XYZ values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed `RGB` values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges" in Chapter 2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToLUV, BGRTToLUV

*Converts an `RGB` or `BGR` image to the `LUV` color model.*

### Syntax

#### Case 1: RGB to LUV

```
IppStatusippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

#### Case 2: BGR to LUV

```
IppStatusippiBGRTToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>32f_C3R</code>
---------------------	----------------------

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.
---------	---

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB or BGR image  $pSrc$  to the CIE LUV [CIE LUV](#) image  $pDst$  in two steps. First, the conversion is done into [CIE XYZ](#) format, using equations defined for the [ippiRGBToXYZ](#) function. After that, conversion to LUV image is performed in accordance with the following equations:

$$L = 116. * (Y/Y_n)^{1/3.} - 16.$$

$$U = 13. * L * (u - u_n)$$

$$V = 13. * L * (v - v_n)$$

where

$$u = 4.*X / (X + 15.*Y + 3.*Z)$$

$$v = 9.*Y / (X + 15.*Y + 3.*Z)$$

$$u_n = 0.197839$$

$$v_n = 0.468342$$

The computed values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The equations above are given on the assumption that R, G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

In case of 8u data type, the computed L, U, and V values are quantized and converted to fit in the range [0..IPP\_MAX\_8U] as follows:

$$L = L * IPP\_MAX\_8U / 100.$$

$$U = (U + 134.) * IPP\_MAX\_8U / 354.$$

$$V = (V + 140.) * IPP\_MAX\_8U / 262.$$

In case of 16u data type, the computed L, U, and V values are quantized and converted to fit in the range [0..IPP\_MAX\_16U] as follows:

$$L = L * IPP\_MAX\_16U / 100.$$

$$U = (U + 134.) * IPP\_MAX\_16U / 354.$$

$$V = (V + 140.) * IPP\_MAX\_16U / 262.$$

In case of 16s data type, the computed L, U, and V values are quantized and converted to fit in the range [IPP\_MIN\_16S..IPP\_MAX\_16S] as follows:

$$L = L * IPP\_MAX\_16U / 100. + IPP\_MIN\_16S$$

$$U = (U + 134.) * IPP\_MAX\_16U / 354. + IPP\_MIN\_16S$$

$$V = (V + 140.) * IPP\_MAX\_16U / 262. + IPP\_MIN\_16S$$

For 32f data type, no further conversion is done and L, U, and V components remain in the ranges [0..100], [-134..220], and [-140..122], respectively.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
-------------	---

---

ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## LUVToRGB, LUVToBGR

Converts a LUV image to the RGB or BGR color model.

### Syntax

#### Case 1: LUV to RGB

```
IppStatusippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

#### Case 2: LUV to BGR

```
IppStatusippiLUVToBGR_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	32f_C3R
--------	---------

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function converts the [CIE LUV](#) image *pSrc* to the [RGB](#) or [BGR](#) image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, LUV components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

```
L = L * 100./ IPP_MAX_8U
U = (U * 354./ IPP_MAX_8U) - 134.
V = (V * 262./ IPP_MAX_8U) - 140.
```

For 16u data type:

```
L = L * 100./ IPP_MAX_16U
U = (U * 354./ IPP_MAX_16U) - 134.
V = (V * 262./ IPP_MAX_16U) - 140.
```

For 16s data type:

```
L = (L - IPP_MIN_16S)* 100./ IPP_MAX_16U
U = ((U - IPP_MIN_16S)* 354./ IPP_MAX_16U) - 134.
V = ((V - IPP_MIN_16S) * 262./ IPP_MAX_16U) - 140.
```

After that, conversion to XYZ format takes place as follows:

```
Y = Yn * ((L + 16.) / 116.)**3.
X = -9.* Y * u / ((u - 4.)* v - u* v )
Z = (9.* Y - 15*v*Y - v*X) / 3. * v
```

where

```
u = U / (13.* L) + un
v = V / (13.* L) + vn
```

and

```
un = 4.*xn / (-2.*xn + 12.*yn + 3.)
vn = 9.*yn / (-2.*xn + 12.*yn + 3.)
```

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point, and  $Y_n = 1.0$  is the luminance of the D65 white point.

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination RGB or BGR format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## BGRTolab, RGBtolab

Converts a BGR or RGB image to the Lab color model.

## Syntax

### Case 1: BGR to Lab

```
IppStatusippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiBGRToLab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

### Case 2: RGB to Lab

```
IppStatusippiRGBToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiRGBToLab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiRGBToLab_32f_P3R(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3],
int dstStep[3], IppiSize roiSize);

IppStatusippiRGBToLab_64f_P3R(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3],
int dstStep[3], IppiSize roiSize);
```

### Case 3: RGB to Lab with platform-aware functions

```
IppStatusippiRGBToLab_32f_P3R_L(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiRGBToLab_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

### Case 4: RGB to Lab with TL functions based on the Platform Aware API

```
IppStatusippiRGBToLab_32f_P3R_LT(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiRGBToLab_64f_P3R_LT(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

### Case 5: RGB to Lab with TL functions based on the Classic API

```
IppStatusippiRGBToLab_32f_P3R_T(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f*
pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiRGBToLab_64f_P3R_T(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f*
pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h  
ippcc\_l.h  
ippcc\_t1.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc, pSrc[3]</i>	Pointer to the source image ROI.
<i>srcStep, srcStep[3]</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst, pDst[3]</i>	Pointer to the destination image ROI.
<i>dstStep, dstStep[3]</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the BGR or RGB image *pSrc* to the [CIE Lab](#) image *pDst*, and vice versa. Conversion to Lab consists of two steps. First, the conversion is done into [CIE XYZ](#) format, using equations defined for the function [ippiRGBToXYZ](#). After that, conversion to the Lab image is performed in accordance with the following equations:

$$L = 116. * (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$$

$$L = 903.3 * (Y/Y_n)^{1/3} \text{ for } Y/Y_n \leq 0.008856$$

$$a = 500. * [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. * [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$f(t) = \begin{cases} t^{1/3}, & t > (6/29)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} & \end{cases}$$

Here  $Y_n = 1.0$ ,  $X_n = 0.950455$ ,  $Z_n = 1.088753$  for the D65 white point with the CIE chromaticity coordinates  $x_n = 0.312713$ ,  $y_n = 0.329016$ .

The equations above are given on the assumption that initial B, G, R values are normalized to the range [0..1]. The computed values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

These values are quantized and scaled to the 8-bit range of 0 to 255 for `8u_C3` flavors:

$$L = L * 255./100.$$

$$a = (a + 128.)$$

$$b = (b + 128.)$$

or to the 16-bit range of 0 to 65535 for `ippiBGRTolab_8u16u_C3R`:

$$L = L * 65535./100.$$

$$a = (a + 128.) * 255$$

$$b = (b + 128.) * 255$$

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> , <i>pSrc[3]</i> , <i>pDst</i> , or <i>pDst[3]</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## LabToBGR, LabToRGB

*Converts a Lab image to the BGR or RGB color model.*

### Syntax

#### Case 1: Lab to BGR

```
IppStatusippiLabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiLabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiLabToBGR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

#### Case 2: Lab to RGB

```
IppStatusippiLabToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiLabToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);

IppStatusippiLabToRGB_32f_P3R(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3],
int dstStep[3], IppiSize roiSize);

IppStatusippiLabToRGB_64f_P3R(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3],
int dstStep[3], IppiSize roiSize);
```

#### Case 3: Lab to RGB with platform-aware functions

```
IppStatusippiLabToRGB_32f_P3R_L(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiLabToRGB_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

#### Case 4: Lab to RGB with TL functions based on the Platform Aware API

```
IppStatusippiLabToRGB_32f_P3R_LT(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiLabToRGB_64f_P3R_LT(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

#### Case 5: Lab to RGB with TL functions based on the Classic API

```
IppStatusippiLabToRGB_32f_P3R_T(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f*
pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiLabToRGB_64f_P3R_T(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f*
pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h  
ippcc\_1.h  
ippcc\_t1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc, pSrc[3]</i>	Pointer to the source image ROI.
<i>srcStep, srcStep[3]</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst, pDst[3]</i>	Pointer to the destination image ROI.
<i>dstStep, dstStep[3]</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [CIE Lab](#) image *pSrc* to the BGR or RGB image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, Lab components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

```
L = L * 100./255.  
a = a - 128.  
b = b - 128.
```

For 16u data type:

```
L = L * 100./65535.  
a = (a/255. - 128.)  
b = (b/255.) - 128.)
```

After that, conversion to XYZ format takes place as follows:

$$\begin{aligned} Y &= Y_n * P^3. \\ X &= X_n * (P + a/500.)^3. \\ Z &= Z_n * (P - b/200.)^3. \end{aligned}$$

where

$$P = (L + 16)/116.$$

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination BGR or RGB format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> , <i>pSrc[3]</i> , <i>pDst</i> , or <i>pDst[3]</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCC

Converts an RGB image to the YCC color model.

### Syntax

```
IppStatusippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'B'G' image *pSrc* to the PhotoY'C'C' image *pDst* according to the following basic equations:

$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B'$$

$$C1' = -0.299 \cdot R' - 0.587 \cdot G' + 0.886 \cdot B' = B' - Y$$

$$C2' = 0.701 \cdot R' - 0.587 \cdot G' - 0.114 \cdot B' = R' - Y$$

The equations above are given on the assumption that R', G', and B' values are normalized to the range [0..1]. In case of the floating-point data type, the input R'G'B' values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed  $Y'$ ,  $C1'$ ,  $C2'$  values are then quantized and converted to fit in the range [0..1] as follows:

```
Y' = 1. / 1.402 * Y'  
C1' = 111.4 / 255. * C1' + 156. / 255.  
C2' = 135.64 / 255. * C2' + 137. / 255.
```

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges" in Chapter 2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## YCCToRGB

*Converts a YCC image to the RGB color model.*

---

### Syntax

```
IppStatusippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `PhotoY'C'C'` image `pSrc` to the `R'B'G'` image `pDst`. The function `ippiYCCToRGB` first restores normal luminance and chrominance data as:

```

Y' = 1.3584 * Y'
C1' = 2.2179 * (C1' - 156./255.)
C2' = 1.8215 * (C2' - 137./255.)

```

The equations above are given on the assumption that source `Y`, `C1`, and `C2` values are normalized to the range [0..1]. In case of the floating-point data type, the input YCC values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

```

R' = Y' + C2'
G' = Y' - 0.194*C1' - 0.509*C2'
B' = Y' + C1'

```

In case of integer function flavors, the computed `R'B'G'` values are then scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges"](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## RGBToHLS

Converts an RGB image to the HLS color model.

### Syntax

```
IppStatusippiRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'B'G' image *pSrc* to the HLS image *pDst*. For function flavors operating on the floating point data, source RGB values must be in the range [0..1].

The conversion algorithm from RGB to HLS can be represented in pseudocode as follows:

```
// Lightness:
M1 = max(R,G,B); M2 = min(R,G,B); L = (M1+M2)/2
// Saturation:
if M1 = M2 then // achromatics case
    S = 0
    H = 0
else // chromatics case
    if L <= 0.5 then
        S = (M1-M2) / (M1+M2)
    else
        S = (M1-M2) / (2-M1-M2)
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M1 then H = Cb - Cg           //change R=M2 to R=M1
if G = M1 then H = 2 + Cr - Cb       //change G=M2 to G=M1
if B = M1 then H = 4 + Cg - Cr       //change B=M2 to B=M1
H = 60*H
if H < 0 then H = H + 360
```

For floating point function flavors, the computed *H*, *L*, *S* values are scaled to the range [0..1]. In case of integer function flavors, these values are scaled to the full range of the destination data type ([Table "Image Data Types and Ranges"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## HLSToRGB

Converts an HLS image to the RGB color model.

### Syntax

```
IppStatusippiHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

---

8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R
---------	----------	----------	----------

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `HLS` image *pSrc* to the `R'B'G'` image *pDst*. For function flavors operating on the floating point data, source HLS values must be in the range [0..1]. The conversion algorithm from `HLS` to `RGB` can be represented in pseudocode as follows:

```

if L <= 0.5 then      M2 = L * (1 + S) else      M2 = L + S - L * S
M1 = 2 * L - M2 if S = 0
then      R = G = B = L else      h = H + 120      if h > 360 then      h = h - 360      if
h < 60 then      R = ( M1 + ( M2 - M1 ) * h / 60)      else if h < 180 then      R =
M2      else if h < 240 then      R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
else      R = M1      h = H      if h < 60 then      G = ( M1 + ( M2 - M1 ) * h /
60      else if h < 180 then      G = M2      else if h < 240 then      G = M1 + ( M2 -
M1 ) * ( 240 - h ) / 60      else      G = M1      h = H - 120      if h < 0
then      h += 360      if h < 60 then      B = ( M1 + ( M2 - M1 ) * h / 60
else if h < 180 then      B = M2      else if h < 240 then      B = M1 + ( M2 - M1 ) *
( 240 - h ) / 60      else      B = M1

```

For floating point function flavors, the computed `R', G', B'` values are scaled to the range [0..1]. In case of integer function flavors, these values are scaled to the full range of the destination data type (see [Table "Image Data Types and Ranges" in Chapter 2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## BGRToHLS

*Converts a BGR image to the HLS color model.*

---

## Syntax

```
IppStatusippiBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatusippiBGRToHLS_8u_AP4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `B'G'R'` image *pSrc* to the `HLS` image *pDst* according to the same formula as the function `ippiRGBToHLS` does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## HLSToBGR

*Converts an HLS image to the RGB color model.*

### Syntax

```
IppStatusippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatusippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatusippiHLSToBGR_8u_AP4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatusippiHLSToBGR_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatusippiHLSToBGR_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiHLSToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [HLS](#) image *pSrc* to the 'B'G'R' image *pDst* according to the same formula as the function [ippiHLSToRGB](#) does.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
---------------	---

## RGBToHSV

*Converts an RGB image to the HSV color model.*

### Syntax

```
IppStatusippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'G'B' image *pSrc* to the HSV image *pDst*.

The conversion algorithm from RGB to HSV can be represented in pseudocode as follows:

```
// Value: V = max(R,G,B); // Saturation: temp = min(R,G,B); if V = 0 then // achromatics
case      S = 0 // H = 0 else // chromatics case      S = (V - temp)/V // Hue: Cr = (V -
R) / (V - temp) Cg = (V - G) / (V - temp) Cb = (V - B) / (V - temp) if R = V then H = Cb - Cg if
G = V then H = 2 + Cr - Cb if B = V then H = 4 + Cg - Cr H = 60*H if H < 0 then H = H + 360
```

The computed *H,S,V* values are scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges” in Chapter 2](#)).

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.

`ippStsSizeErr` Indicates an error condition if `roiSize` has a field with a zero or negative value.

## HSVToRGB

*Converts an HSV image to the RGB color model.*

### Syntax

```
IppStatusippiHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a `HSV` image `pSrc` to the image `R'G'B'` `pDst`.

The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```
if S = 0 then R = G = B = V else if H = 360 then H = 0 else H = H/60
I = floor(H) F = H - I; M = V * (1 - S); N = V * (1 - S * F); K =
V * (1 - S * (1 - F)); if(I == 0)then{ R = V;G = K;B = M;} if(I == 1)then{ R = N;G =
V;B = M;} if(I == 2)then{ R = M;G = V;B = K;} if(I == 3)then{ R = M;G = N;B = V;}
if(I == 4)then{ R = K;G = M;B = V;} if(I == 5)then{ R = V;G = M;B = N;}
```

The computed `R'`, `G'`, `B'` values are scaled to the full range of the destination data type (see [Table “Image Data Types and Ranges” in Chapter 2](#)).

### Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## RGBToYCoCg

*Converts a RGB image to the YCoCg color model.*

---

### Syntax

```
IppStatusippiRGBToYCoCg_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roi);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the destination image ROI in each plane.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roi</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a RGB image *pSrc* to the [YCoCg](#) image *pDst* according to the following formulas:

$$Y = ((R + 2*G + B) + 2)/4$$

$$Co = ((R - B) + 1)/2$$

$$Cg = (( - R + 2*G - B) + 2)/4$$

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

## YCoCgToRGB

*Converts a YCoCg image to the RGB image.*

---

## Syntax

```
IppStatusippiYCoCgToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roi);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Array of pointers to the source image ROI in each plane.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roi</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg](#) image *pSrc* to the RGB image *pDst* according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## BGRToYCoCg

Converts a 24-bit BGR image to the YCoCg color model.

## Syntax

```
IppStatusippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

```
IppStatusippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image *pBGR* to the YCoCg image *pYCC* according to the following formulas:

$$Y = ((R + 2*G + B) + 2)/4$$

$$Co = ((R - B) + 1)/2$$

$$Cg = (( - R + 2*G - B) + 2)/4$$

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.

## SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

---

## Syntax

```
IppStatusippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_C3P3R	16s32s_C3P3R
16s_C4P3R	16s32s_C4P3R

## Include Files

ippcc.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit `BGR` image *pBGR* to the `YCoCg` image *pYCC* according to the following formulas:

$$\begin{aligned} Y &= ((R + 2*G + B) + 2)/4 \\ Co &= ((R - B) + 1)/2 \\ Cg &= (( - R + 2*G - B) + 2)/4 \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToBGR

Converts a `YCoCg` image to the 24-bit `BGR` image.

## Syntax

```
IppStatusippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatusippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.

<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `YCoCg` image *pYCC* to the 24-bit BGR image *pBGR* according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## YCoCgToSBGR

*Converts a YCoCg image to the 48-bit BGR image.*

---

## Syntax

### Case 1: Conversion to 3-channel image

```
IppStatusippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize);
```

```
IppStatusippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize);
```

### Case 2: Conversion to 4-channel image

```
IppStatusippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize, Ipp16s aval);
```

```
IppStatusippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize, Ipp16s aval);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
-------------	--

---

<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg](#) image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## BGRToYCoCg\_Rev

Converts a 24-bit BGR image to the YCoCg-R color model.

---

### Syntax

```
IppStatusippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

```
IppStatusippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image *pSrc* to the YCoCg-R image *pDst* according to the following formulas:

$$\begin{aligned} Co &= R - B \\ t &= B + (Co \gg 1) \\ Cg &= G - t \\ Y &= t + (Cg \gg 1) \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## SBGRToYCoCg\_Rev

Converts a 48-bit BGR image to the YCoCg-R color model.

---

## Syntax

```
IppStatusippiSBGRToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>16s_C3P3R</code>	<code>16s32s_C3P3R</code>
<code>16s_C4P3R</code>	<code>16s32s_C4P3R</code>

## Include Files

`ippcc.h`

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit *BGR* image *pBGR* to the *YCoCg-R* image *pYCC* according to the following formulas:

$$\begin{aligned} Co &= R - B \\ t &= B + (Co \gg 1) \\ Cg &= G - t \\ Y &= t + (Cg \gg 1) \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## **YCoCgToBGR\_Rev**

*Converts a YCoCg-R image to the 24-bit BGR image.*

### Syntax

```
IppStatusippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatusippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg-R](#) image `pYCC` to the 24-bit BGR image `pBGR` according to the following formulas:

$$t = Y - (Cg \gg 1)$$

$$G = Cg + t$$

$$B = t - (Co \gg 1)$$

$$R = B + Co$$

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## [YCoCgToSBGR\\_Rev](#)

Converts a YCoCg-R image to the 48-bit BGR image.

### Syntax

#### Case 1: Conversion to 3-channel image.

```
IppStatusippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
IppStatusippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

#### Case 2: Conversion to 4-channel image

```
IppStatusippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatusippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pYCC</code>	Array of pointers to the source image ROI in each plane.
<code>yccStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pBGR</code>	Pointer to the destination image ROI.
<code>bgrStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

`aval` Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `YCoCg-R` image  $pYCC$  to the 48-bit BGR image  $pBGR$  according to the following formulas:

$$t = Y - (Cg \gg 1)$$

$$G = Cg + t$$

$$B = t - (Co \gg 1)$$

$$R = B + Co$$

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

# Color - Gray Scale Conversions

## GrayToRGB

*Covers a gray scale image to RGB/BGR by copying luminance component to color components.*

### Syntax

```
IppStatusippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C1C3R</code>	<code>16u_C1C3R</code>	<code>32f_C1C3R</code>
-----------------------	------------------------	------------------------

```
IppStatusippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> aval);
```

Supported values for `mod`:

<code>8u_C1C4R</code>	<code>16u_C1C4R</code>	<code>32f_C1C4R</code>
-----------------------	------------------------	------------------------

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function converts a gray scale image to an RGB/BGR image by copying luminance component to color components.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error when <i>pSrc</i> or <i>pDst</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the `ippiGrayToRGB_8u_C1C4R` function.

```
const int WIDTH = 2;
const int HEIGHT = 1;

Ipp8u pSrc[WIDTH * HEIGHT] = {
    113,113,
};

Ipp8u pDst[WIDTH * HEIGHT * 4];
int srcStep = WIDTH, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status =ippiGrayToRGB_8u_C1C4R(pSrc, srcStep, pDst, dstStep, roiSize, 0xFF);
if (status == ippStsNoErr) {
    printf("PASS:\n(%3d %3d %3d %3d), (%3d %3d %3d %3d)\n", pDst[0], pDst[1], pDst[2],
pDst[3], pDst[4], pDst[5], pDst[6], pDst[7]);
}
else
    printf("FAIL: status = %d\n", status);
```

Result:

```
PASS:
(113 113 113 255), (113 113 113 255)
```

## See Also

[Regions of Interest in Intel IPP](#)

## RGBToGray

Converts an RGB image to gray scale using fixed transform coefficients.

### Syntax

```
IppStatusippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

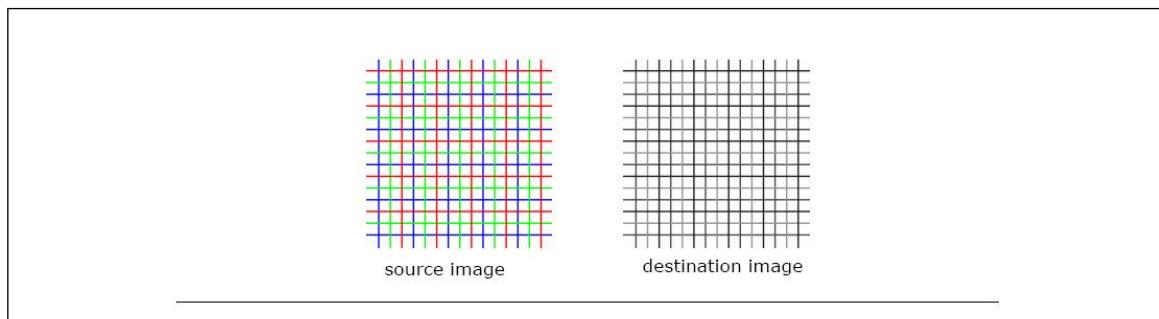
<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI.

Conversion from RGB image to gray scale (see figure *Converting an RGB Image to Gray Scale*) uses the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values:  $Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$ . Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

### Converting an RGB Image to Gray Scale



## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiRGBToGray_8u_C3C1R`.

```
Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                     0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                     0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };

ippiRGBToGray_8u_C3C1R ( src, 12, dst, 4, srcRoi );
```

### Result:

255 0 0 255 0 0 255 0 0 255 0 0	src
0 255 0 0 255 0 0 255 0 0 255 0	
0 0 255 0 0 255 0 0 255 0 0 255	
76 76 76 76	
149 149 149 149	dst
29 29 29 29	

## ColorToGray

Converts an RGB image to gray scale using custom transform coefficients.

---

### Syntax

```
IppStatusippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
```

Supported values for *mod*:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

```
IppStatusippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp64f coeffs[3]);
```

Supported values for *mod*:

64f_C3C1R
64f_AC4C1R

### Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h,ippi.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib,ippi.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>coeffs</code>	Transform coefficients.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function uses the following equation to convert an RGB image to gray scale:

$$Y = coeffs[0] * R + coeffs[1] * G + coeffs[2] * B,$$

where the `coeffs` array contains user-defined transform coefficients which must be non-negative and satisfy the condition

$$coeffs[0] + coeffs[1] + coeffs[2] \leq 1.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## CFAToBGRA

*Restores the RGB image from the gray-scale CFA image using the VNG algorithm.*

## Syntax

```
IppStatusippiCFAToBGRA_VNG_8u_C1C4R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp32f scale[4], Ipp8u* pDst, int dstStep, IppiBayerGrid grid);
IppStatusippiCFAToBGRA_VNG_16u_C1C4R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp32f scale[4], Ipp16u* pDst, int dstStep, IppiBayerGrid grid);
```

## Platform-aware functions

```
IppStatusippiCFAToBGRA_VNG_8u_C1C4R_L(const Ipp8u* pSrc, IppiRectL srcRoiL, IppiSizeL srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp8u* pDst, IppSizeL dstStepL, IppiBayerGrid grid);
```

```
IppStatusippiCFAToBGRA_VNG_16u_C1C4R_L(const Ipp16u* pSrc, IppiRectL srcRoiL,
IppiSizeL srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp16u* pDst, IppSizeL
dstStepL, IppiBayerGrid grid);
```

## Include Files

ippcc.h

ippcc\_l.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h, ipp.i.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ipp.i.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcRoi, srcRoiL</i>	Region of interest in the source image (of the IppiRect or IppiRectL type).
<i>srcSize, srcSizeL</i>	Size of the source image.
<i>srcStep, srcStepL</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>scale[4]</i>	Coefficients by which the resulting RGB channels are multiplied after interpolation. By default, equal to 1.0.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep, dstStepL</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>grid</i>	Specifies the configuration of the Bayer grid in the source image. The function copies 2-pixel width border pixels from the internal neighborhood pixels. The following values are possible: ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the Variable Number of Gradients (VNG) demosaicing algorithm.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

---

ippStsSizeErr	Indicates an error condition if the <i>srcSize</i> or <i>srcSizeL</i> has a field that is less than 2, or if the <i>srcRoi</i> or <i>srcRoiL</i> has a field with a negative or zero value.
ippStsBadArgErr	Indicates an error condition if <i>grid</i> has an illegal value.

## See Also

[Structures and Enumerators](#)

[Structures and Enumerators for Platform-Aware Functions](#)

## CFAToRGB

*Restores the RGB image from the gray-scale CFA image.*

## Syntax

```
IppStatusippiCFAToRGB_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize,  
int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

```
IppStatusippiCFAToRGB_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize,  
int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

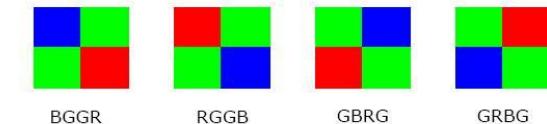
<i>pSrc</i>	Pointer to the source image origin.
<i>srcSize</i>	Size of the source image.
<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>grid</i>	Specifies the configuration of the Bayer grid in the source image. The following values are possible:  ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG
<i>interpolation</i>	Interpolation method, reserved, must be 0.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) - an array of Bayer filters, to 24-bit three-channel RGB image. The order of the color component in the source image - Bayer grid - is specified by the parameter *grid*. Four possible values of this parameter correspond to the allowed variants of the Bayer grid (see Figure below).

### Possible Configurations of the Bayer Grids



Each element of the source image contains an intensity value for only one color component, two others are interpolated using neighbor elements. R and B values are interpolated linearly from the nearest neighbors of the same color. When interpolating R and B values on green pixel, the average values of the two nearest neighbors (above and below, or left and right) of the same colors are used. When interpolating R or B values on the blue or red pixel respectively, the average values of the four nearest blue (red) pixels cornering the red (blue) pixel are used. G values are interpolated using an adaptive interpolation [Sak98] from a pair of nearest neighbors (vertical or horizontal) and taking into account the correlation in the red (or blue) component. The pair is chosen depending on the values of the difference between the red (blue) pixels in the vertical and horizontal directions. If the difference is smaller in the vertical direction - a vertical pair of green pixels is used, if it is smaller in the horizontal direction - a horizontal pair is used. If the difference is the same, all four neighbors are used.

This interpolation requires border pixels for the input pixels near the horizontal or vertical edge of the image. The function uses the mirrored border of two edge rows or columns of the input image. In this case the G values is calculated as the average of four nearest green pixels.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if the <i>srcSize</i> has a field that is less than 2, or if the <i>roiSize</i> has a field with negative or zero value.
ippStsBadArgErr	Indicates an error condition if <i>grid</i> has an illegal value.

## DemosaicAHD

*Restores the RGB image from the gray-scale CFA image using AHD algorithm.*

### Syntax

```
IppStatusippiDemosaicAHD_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize
srcSize, int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, Ipp8u* pTmp, int
tmpStep);
```

---

```
IppStatusippiDemosaicAHD_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize
srcSize, int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, Ipp16u* pTmp, int
tmpStep);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>srcSize</i>	Size of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>grid</i>	Specifies the configuration of the Bayer grid in the source image. The following values are possible (see <a href="#">Figure Possible Configurations of the Bayer Grids</a> ): ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG
<i>pTmp</i>	Pointer to the temporary image of ( <i>srcRoi.width + 6, 30</i> ) size.
<i>tmpStep</i>	Distance in bytes between starts of consecutive lines in the temporary image.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the adaptive homogeneity-directed demosaicing (AHD) algorithm [Hir05]. The algorithm requires the temporary image *pTmp* of size *srcRoi.width + 6.30*.

The type of the Bayer grid (see [Figure Possible Configurations of the Bayer Grids](#)) is specified by the parameter *grid*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.

ippStsSizeErr	Indicates an error condition if the <i>srcSize</i> has a field that is less than 5, or if the <i>roiSize</i> has a field with negative or zero value.
ippStsBadArgErr	Indicates an error condition if <i>grid</i> has an illegal value.

## Format Conversion

---

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified mainly in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

### YCbCr422

*Converts 4:2:2 YCbCr image.*

---

#### Syntax

```
IppStatusippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Include Files

ippcc.h

#### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

#### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2](#) two-channel source image  $pSrc$  to the 4:2:2 three-plane image  $pDst$  and vice versa (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#) for more details on 4:2:2 planar and pixel-order formats).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> of the first plane is less than 2, or <code>roiSize.height</code> is less than or equal to zero.

## YCbCr422ToYCrCb422

*Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.*

### Syntax

```
IppStatusippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst, int dstStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

$pSrc$	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
$srcStep$	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
$pDst$	Pointer to the destination image ROI.
$dstStep$	Distance in bytes between starts of consecutive lines in the destination image.
$roiSize$	Size of the ROI in pixels, its width should be multiple of 2.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) YCbCr source image  $pSrc$  to the 4:2:2 YCrCb two-channel image  $pDst$  that has the following sequence of samples:  $Y_0, Cr_0, Y_1, Cb_0, Y_2, Cr_1, Y_3, Cb_1, \dots$  (see [Table "Pixel-Order Image Formats"](#)). The source image can be either two-channel or three-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <code>roiSize.width</code> is less than 2.

## YCbCr422ToCbYCr422

*Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.*

### Syntax

```
IppStatusippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiYCbCr422ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels; its width should be multiple of 2.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts 4:2:2 YCbCr source image *pSrc* to the 4:2:2 CbYCr two-channel image *pDst* that has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb3, ... (see [Table "Pixel-Order Image Formats"](#)). The source image can be either two-channel or three-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.

---

ippStsSizeErr	Indicates an error condition if <code>roiSize.width</code> of the first plane is less than 2, or <code>roiSize.height</code> is less than or equal to zero.
---------------	---

## YCbCr422ToYCbCr420

Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.

---

### Syntax

#### Case 1: Operation on planar data

```
IppStatusippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data

```
IppStatusippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2](#) image *pSrc* to the [4:2:0](#) image. The source image can be two-channel or three-plane, destination image always is planar with two or three planes (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#)). Two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane *pDstY*, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## Example

The code example below shows how to use the function `ippiYCbCr422ToYCbCr420_8u_C2P3R`.

```
{
    Ipp8u*    ImageI420[3];
    int       stepI420[3];
    Ipp8u*    ImageYUY2;
    int       stepYUY2;
    IppiSize roiSize = { 1024, 768 };
    ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[0]) );
    ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[1]) );
    ImageI420[2] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[2]) );
    ImageYUY2   = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2 );
    ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420, roiSize);

    ippiFree(ImageI420[0]);
    ippiFree(ImageI420[1]);
    ippiFree(ImageI420[2]);
    ippiFree(ImageYUY2);
}
```

## YCbCr422To420\_Interlace

*Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.*

## Syntax

```
IppStatus ippiYCbCr422To420_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane fore destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced `4:2:2` image *pSrc* to the `4:2:0` image *pDst* (see [Table "Planar Image Formats"](#)).

The conversion is performed in accordance with the following formulas:

$$\begin{aligned} Y_{\text{dest}} &= Y_{\text{src}}; \\ Cb_0(Cr_0)_{\text{dest}} &= (3 * Cb_0(Cr_0)_{\text{src}} + Cb_2(Cr_2)_{\text{src}} + 2) / 4; \\ Cb_1(Cr_1)_{\text{dest}} &= (Cb_1(Cr_1)_{\text{src}} + 3 * Cb_3(Cr_3)_{\text{src}} + 2) / 4; \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2, or <code>roiSize.height</code> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize.width</code> is not multiple of 2, or <code>roiSize.height</code> is not multiple of 4.

## YCbCr422ToYCrCb420

*Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.*

### Syntax

```
IppStatusippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr422ToYCrCb420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCrCb, int dstUVStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>pDstY</i>	Pointer to the destination image Y plane.
<i>dstStep</i>	Array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.
<i>dstYStep</i>	Array of distances, in bytes, between the starting points of consecutive lines in the destination image Y plane.
<i>dstUVStep</i>	Array of distances, in bytes, between the starting points of consecutive lines in the destination image UV plane.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2](#) two-channel image *pSrc* that has the following sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ... to the [4:2:0](#) three-plane image *pDst* with the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr422ToYCbCr411

Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

---

## Syntax

### Case 1: Operation on planar data

```
IppStatusippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatusippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data

```
IppStatusippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatusippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **4:2:2** image *pSrc* to the **4:1:1** image. The source image can be two-channel or three-plane (see [Table “Pixel-Order Image Formats”](#) for more details), destination image always is planar with two or three planes (see [Table “Planar Image Formats”](#) for more details). The two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane *pDstY*, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane *pDstCbCr*.

The value of the fields of the *roiSize* have certain limitations:

- its width should be multiple of 4 and cannot be less than 4 for operation on two-channel images;
- its width should be multiple of 4 and cannot be less than 4, and its height should be multiple of 2 and can not be less than 2 for three-plane to two-plane image conversion;
- both height and width should be multiple of 2 and cannot be less than 2 for operation on three-plane images.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if corresponding fileds of the <i>roiSize</i> is less than specified above values.
---------------	---

## YCrCb422ToYCbCr422

*Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.*

---

### Syntax

```
IppStatusippiYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2 YCrCb](#) two-channel image *pSrc* (see [Table “Pixel-Order Image Formats”](#)) to the [4:2:2 YCbCr](#) three-plane image *pDst* (see [Table “Planar Image Formats”](#)).

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## YCrCb422ToYCbCr420

*Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.*

---

### Syntax

```
IppStatusippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This functions converts the [4:2:2 YCrCb](#) two-channel image *pSrc* (see [Table "Pixel-Order Image Formats"](#)) to the [4:2:0 YCbCr](#) three-plane image *pDst* (see [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb422ToYCbCr411

*Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.*

## Syntax

```
IppStatusippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2 YCrCb two-channel image](#) *pSrc* (see [Table “Pixel-Order Image Formats”](#)) to the [4:1:1 YCbCr three-plane image](#) *pDst* (see [Table “Planar Image Formats”](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## CbYCr422ToYCbCr422

*Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.*

## Syntax

```
IppStatusippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatusippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:2CbYCr` two-channel image *pSrc* to the `4:2:2 YCbCr` two-channel or three-plane image *pDst* (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)). The source image has the following sequence of samples: `Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...`. Two-channel destination image has different sequence of samples: `Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ...`.

## Return Values

<code>ppStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2.

## **CbYCr422ToYCbCr420**

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

### Syntax

```
IppStatusippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h,ippi.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib,ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.

<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:2 CbYCr` two-channel image *pSrc* to the `4:2:0 YCbCr` two- or three-plane image *pDst* (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#)). The source image has the following sequence of samples: `Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...`. Three-plane destination image has the following order of pointers: `Y-plane, Cb-plane, Cr-plane`. Two-plane destination image contains luminance samples `Y0, Y1, Y2, ...` in the first plane *pDstY*, and interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...` in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## **CbYCr422ToYCbCr420\_Interlace**

*Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.*

---

## Syntax

```
IppStatusippiCbYCr422ToYCbCr420_Interlace_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h,ippi.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib,ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.

---

<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#) ).

This function converts interlaced **4:2:2 CbYCr** two-channel image *pSrc* to the **4:2:0 YCbCr** three-plane image *pDst* (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Three-plane destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

$$\begin{aligned} Y_{\text{dest}} &= Y_{\text{src}}; \\ Cb_0(Cr_0)_{\text{dest}} &= (3 * Cb_0(Cr_0)_{\text{src}} + Cb_2(Cr_2)_{\text{src}} + 2) / 4; \\ Cb_1(Cr_1)_{\text{dest}} &= (Cb_1(Cr_1)_{\text{src}} + 3 * Cb_3(Cr_3)_{\text{src}} + 2) / 4; \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2, or <i>roiSize.height</i> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize.width</i> is not multiple of 2, or <i>roiSize.height</i> is not multiple of 4.

## CbYCr422ToYCrCb420

Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.

### Syntax

```
IppStatusippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:2CbYCr` two-channel image *pSrc* to the `4:2:0YCrCb` three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## CbYCr422ToYCbCr411

Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.

## Syntax

```
IppStatusippiCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:2CbYCr](#) two-channel image *pSrc* to the [4:1:1YCbCr](#) three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4.

## YCbCr420

*Converts 4:2:0 YCbCr image.*

## Syntax

```
IppStatusippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatusippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.

<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:0](#) three-plane (see [Table “Planar Image Formats”](#)) source image *pSrc* to the 4:2:0 two-plane image and vice versa. Two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToYCbCr422

*Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.*

---

## Syntax

```
IppStatusippiYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:0](#) planar source image *pSrc* to the [4:2:2](#) image *pDst*. The source image can be two- or three-plane image (see [Table "Planar Image Formats"](#)). The first plane of the two-plane source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* can be three-plane or two-channel (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## [YCbCr420ToYCbCr422\\_Filter](#)

*Convert 4:2:0 image to 4:2:2 image with additional filtering.*

## Syntax

```
IppStatusippiYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize
roiSize);
```

```
IppStatusippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int
layout);
```

## Include Files

ippcc.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.								
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.								
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.								
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.								
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.								
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.								
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.								
<i>roiSize</i>	Size of the ROI in pixels.								
<i>layout</i>	Slice layout. Possible values:  <table border="0"> <tbody> <tr> <td>IPP_UPPER</td> <td>Upper (first) slice</td> </tr> <tr> <td>IPP_CENTER</td> <td>Middle slices</td> </tr> <tr> <td>IPP_LOWER</td> <td>Lowermost (last) slice</td> </tr> <tr> <td>IPP_LOWER &amp;&amp; IPP_UPPER &amp;&amp; IPP_CENTER</td> <td>Image is not sliced</td> </tr> </tbody> </table>	IPP_UPPER	Upper (first) slice	IPP_CENTER	Middle slices	IPP_LOWER	Lowermost (last) slice	IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced
IPP_UPPER	Upper (first) slice								
IPP_CENTER	Middle slices								
IPP_LOWER	Lowermost (last) slice								
IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced								

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:0` planar source image `pSrc` to the `4:2:2` image `pDst` and performs additional filtering. The source image can be two- or three-plane image (see [Table "Planar Image Formats"](#)). The first plane of the two-plane source image `pSrcY` contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane `pSrcCbCr` contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image `pDst` can be three-plane or two-channel (see [Table "Pixel-Order Image Formats"](#) and [Table "Planar Image Formats"](#)).

The function flavors `ippiYCbCr420ToYCbCr422_Filter_8u_P3R` and `ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R` additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case `roiSize.width` should be multiple of 2, and `roiSize.height` should be multiple of 8.

The function `ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R` additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice `layout` should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.

### **Caution**

The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

The function may be applied to a not-sliced image as well. In this case `roiSize.width` and `roiSize.height` should be multiple of 2.

### **Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> has wrong value.

## **YCbCr420To422\_Interlace**

*Converts interlaced YCbCr image from 4:2:0 sampling format to 4:2:2 format.*

### **Syntax**

```
IppStatusippiYCbCr420To422_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### **Include Files**

`ippcc.h`

### **Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### **Parameters**

<code>pSrc</code>	Array of pointers to the ROI in each plane for source image.
-------------------	--

<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced planar [4:2:0](#) source image *pSrc* to the [4:2:2](#) image *pDst*. Three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

$$\begin{aligned} Y_{n_{\text{dest}}} &= Y_{n_{\text{src}}}; \\ Cb0(Cr0)_{\text{dest}} &= (5*Cb0(Cr0)_{\text{src}} + 3*Cb2(Cr2)_{\text{src}} + 4)/8; \\ Cb1(Cr1)_{\text{dest}} &= (7*Cb1(Cr1)_{\text{src}} + Cb3(Cr3)_{\text{src}} + 4)/8; \\ Cb2(Cr2)_{\text{dest}} &= (Cb0(Cr0)_{\text{src}} + 7*Cb2(Cr2)_{\text{src}} + 4)/8; \\ Cb3(Cr3)_{\text{dest}} &= (3*Cb1(Cr1)_{\text{src}} + 5*Cb3(Cr3)_{\text{src}} + 4)/8; \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2, or <i>roiSize.height</i> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <i>roiSize.width</i> is not multiple of 2, or <i>roiSize.height</i> is not multiple of 4.

## YCbCr420ToCbYCr422

Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.

### Syntax

```
IppStatusippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the planar [4:2:0](#) two-plane source image to the pixel-order [4:2:2](#) two-channel image. The first plane of the source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* has the following sequence of samples:  $Cb_0, Y_0, Cr_0, Y_1, Cb_1, Y_2, Cr_1, Y_3, Cb_2, \dots$

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToCbYCr422\_Interlace

*Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.*

## Syntax

```
IppStatusippiYCbCr420ToCbYCr422_Interlace_8u_P3C2R(const Ipp8u* pSrc[3], int
srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for source image.
-------------	--

<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the interlaced planar [4:2:0](#) image to the pixel-order [4:2:2](#) two-channel image. Three-plane source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane. The destination image *pDst* has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...

The conversion is performed in accordance with the following formulas:

```

Yndest = Ynsrc;
Cb0(Cr0)dest = (5*Cb0(Cr0)src + 3*Cb2(Cr2)src + 4)/8;
Cb1(Cr1)dest = (7*Cb1(Cr1)src + Cb3(Cr3)src + 4)/8;
Cb2(Cr2)dest = (Cb0(Cr0)src + 7*Cb2(Cr2)src + 4)/8;
Cb3(Cr3)dest = (3*Cb1(Cr1)src + 5*Cb3(Cr3)src + 4)/8;

```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2, or <code>roiSize.height</code> is less than 4.
<code>ippStsDoubleSize</code>	Indicates a warning if <code>roiSize.width</code> is not multiple of 2, or <code>roiSize.height</code> is not multiple of 4.

## YCbCr420ToYCrCb420

Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.

### Syntax

```
IppStatusippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:0](#) two-plane source image *pSrc* to the 4:2:0 three-plane image *pDst*. The first plane of the source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#)).

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCbCr420ToYCrCb420\_Filter

Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image  
with deinterlace filtering.

### Syntax

```
IppStatusippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize
roiSize, int layout);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.
<i>layout</i>	Slice layout. Possible values:
IPP_UPPER	Upper (first) slice
IPP_CENTER	Middle slices
IPP_LOWER	Lowermost (last) slice
IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:0](#) two-plane source image to the 4:2:0 three-plane image. The first plane of the source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* has the following order of pointers: *Y*-plane, *Cr*-plane, *Cb*-plane. The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.

---

### Caution

The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

---

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.

`ippStsSizeErr` Indicates an error condition if any field of the `roiSize` is less than 2.

## YCbCr420ToYCbCr411

Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.

### Syntax

```
IppStatusippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*  
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);  
  
IppStatusippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const  
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);  
  
IppStatusippiYCbCr420To411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*  
pDst[3], int dstStep[3], IppiSize roiSize);  
  
IppStatusippiYCbCr420To1620_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*  
pDst[3], int dstStep[3], IppiSize roiSize);  
  
IppStatusippiYCbCr1620To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*  
pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane for a three-plane source image.
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<code>pSrcY</code>	Pointer to the ROI in the luminance plane for a two-plane source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane for a three-plane destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<code>pDstY</code>	Pointer to the ROI in the luminance plane for a two-plane destination image.

<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:2:0` source image to the `4:1:1` destination image. The source two-plane image is converted to destination three-plane image and vice versa. The first plane of the two-plane image contains luminance samples `Y0, Y1, Y2, ...`, the second plane contains interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...`. The three-plane image has the following order of pointers: `Y-plane, Cb-plane, Cr-plane` (see [Table "Planar Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4 or <code>roiSize.height</code> is less than 2.

## YCrCb420ToYCbCr422

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.

## Syntax

```
IppStatusippiYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*pDst, int dstStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:** `ippcore.h, ippvm.h, ipps.h,ippi.h`

**Libraries:** `ippcore.lib, ippvm.lib, ipps.lib,ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.

---

<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

This function converts the **4:2:0 YCrCb** three-plane image *pSrc* to the **4:2:2 YCbCr** three-plane or two-channel image *pDst* (see [Table “Pixel-Order Image Formats”](#) and [Table “Planar Image Formats”](#)).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## **YCrCb420ToYCbCr422\_Filter**

*Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.*

---

## Syntax

```
IppStatusippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the  $4:2:0$  YCrCb three-plane image  $pSrc$  to the  $4:2:2$  YCbCr three-plane image  $pDst$  (see [Table "Planar Image Formats"](#)).

Additionally, this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 8.

## YCrCb420ToCbYCr422

Converts  $4:2:0$  YCrCb image to  $4:2:2$  CbYCr image.

## Syntax

```
IppStatusippiYCrCb420ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

$pSrc$	Array of pointers to the ROI in each plane of the source image.
$srcStep$	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
$pDst$	Pointer to the destination image ROI.
$dstStep$	Distance in bytes between starts of consecutive lines in the destination image.
$roiSize$	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the  $4:2:0$  YCrCb three-plane image  $pSrc$  (see [Table "Planar Image Formats"](#)) to the  $4:2:2$  CbYCr two-channel image  $pDst$  with the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... (see [Table "Pixel-Order Image Formats"](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb420ToYCbCr420

Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.

### Syntax

```
IppStatusippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:0 YCrCb three-plane image *pSrc* (see [Table "Planar Image Formats"](#)) to the 4:2:0 YCbCr two-plane image that contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane *pDstY*, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane *pDstCbCr*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
-------------	---

ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

### Syntax

```
IppStatusippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:2:0 YCrCb](#) three-plane image *pSrc* (see [Table “Planar Image Formats”](#)) to the [4:1:1 YCbCr](#) two-plane image that contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane *pDstY*, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane *pDstCbCr*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## YCbCr411

*Converts 4:1:1 YCbCr image.*

### Syntax

```
IppStatusippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatusippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:1:1](#) three-plane (see [Table “Planar Image Formats”](#)) source image *pSrc* to the 4:1:1 two-plane image and vice versa. Two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4.

## YCbCr411ToYCbCr422

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.

### Syntax

```
IppStatusippiYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

`ippcc.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) planar source image *pSrc* to the [4:2:2](#) image *pDst*. The first plane of the two-plane source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* can be either three-plane (see [Table "Planar Image Formats"](#)) or two-channel image (see [Table "Pixel-Order Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr411ToYCrCb422

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

## Syntax

```
IppStatusippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.

*roiSize* Size of the ROI in pixels, its width should be multiple of 4.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **4:1:1** three-plane image *pSrc* to the **4:2:2** two-channel or three-plane image *pDst* with different order of components. The source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table "Planar Image Formats"](#)). The three-plane destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table "Planar Image Formats"](#)), and two-channel destination image has the following sequence of samples: Y<sub>0</sub>, Cr<sub>0</sub>, Y<sub>1</sub>, Cb<sub>0</sub>, Y<sub>2</sub>, Cr<sub>1</sub>, Y<sub>3</sub>, Cb<sub>1</sub>, ... (see [Table "Pixel-Order Image Formats"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

## YCbCr411ToYCbCr420, YCbCr411To420

*Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.*

## Syntax

```
IppStatusippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatusippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiYCbCr411To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

`ippcc.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.

<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [4:1:1](#) planar source image *pSrc* (see [Table “Planar Image Formats”](#)) to the [4:2:0](#) planar image *pDst*. Both source and destination images can be three- or two-plane. Three-plane images has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table “Planar Image Formats”](#)). Two-plane images contain luminance samples  $Y_0, Y_1, Y_2, \dots$  in the first plane, and interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$  in the second plane.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## YCbCr411ToYCrCb420

Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.

## Syntax

```
IppStatusippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrcY</code>	Pointer to the ROI in the luminance plane of the source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.
<code>dstStep</code>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<code>roiSize</code>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `4:1:1` two-plane source image `pSrc` to the `4:2:0` three-plane image `pDst` with a different order of components. The first plane of the source image `pSrcY` contains luminance samples `Y0, Y1, Y2, ...`, the second plane `pSrcCbCr` contains interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...`. The destination image has the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see [Table "Planar Image Formats"](#)),

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4 or <code>roiSize.height</code> is less than 2.

## Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if  $(r, g, b)$  is a source pixel, then the destination pixel values  $(R, G, B)$  are computed as follows:

$$R = t_{11} \cdot r + t_{12} \cdot g + t_{13} \cdot b + t_{14}$$

$$G = t_{21} \cdot r + t_{22} \cdot g + t_{23} \cdot b + t_{24}$$

$$B = t_{31} \cdot r + t_{32} \cdot g + t_{33} \cdot b + t_{34}$$

where

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

is the color twist matrix. The color twist matrix used by the Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

## ColorTwist

*Applies a color twist matrix to an image with floating-point pixel values.*

### Syntax

#### Case 1: Not-in-place operation on pixel-order data

```
IppStatusippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

32f\_C3R  
32f\_AC4R

```
IppStatusippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[4][4]);
```

#### Case 2: Not-in-place operation on planar data

```
IppStatusippiColorTwist_32f_P3R(const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

#### Case 3: In-place operation on pixel-order data

```
IppStatusippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

32f\_C3IR  
32f\_AC4IR

#### Case 4: In-place operation on planar data

```
IppStatusippiColorTwist_32f_IP3R(Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

### Include Files

ippcc.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

*pSrc* Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.

## ColorTwist32f

*Applies a color twist matrix to an image with integer pixel values.*

---

## Syntax

### Case 1: Not-in-place operation on pixel-order data

```
IppStatusippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

### Case 2: Not-in-place operation on planar data

```
IppStatusippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc[3], int srcStep,
Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod`:

<code>8u_P3R</code>	<code>16u_P3R</code>	<code>16s_P3R</code>
---------------------	----------------------	----------------------

**Case 3: In-place operation on pixel-order data**

```
IppStatusippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize  
roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod :

8u_C3IR	16u_C3IR	16s_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR

**Case 4: In-place operation on planar data**

```
IppStatusippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize  
roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod :

8u_IP3R	16u_IP3R	16s_IP3R
---------	----------	----------

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h, ipp.i.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from the RGB to the YCbCr format can be done as

$$\begin{aligned}Y &= 0.299*R + 0.587*G + 0.114*B \\Cb &= -0.16874*R - 0.33126*G + 0.5*B + 0.5 \\Cr &= 0.5*R - 0.41869*G - 0.08131*B + 0.5\end{aligned}$$

which can be described in terms of the following color twist matrix:

0.29900f	0.58700f	0.11400f	0.000f
-0.16874f	-0.33126f	0.50000f	128.0f
0.50000f	-0.41869f	-0.08131f	128.0f

Color-twist matrices may also be used to perform many other color conversions.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

# Color Keying

---

## CompColorKey

*Performs color keying of two images.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey);
```

Supported values for mod:

8u\_C1R        16u\_C1R        16s\_C1R

#### Case 2: Operation on multi-channel data

```
IppStatusippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[3]);
```

Supported values for mod:

8u\_C3R        16u\_C3R        16s\_C3R

```
IppStatusippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[4]);
```

Supported values for mod:

8u\_C4R        16u\_C4R        16s\_C4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointer to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>colorKey</i>	Value of the key color for 1-channel images, array of color values for multi-channel images.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and stores the result in the destination image *pDst*.

The [Figure Applying the Function ippiCompColorKey to Sample Images](#) shows an example of how the function *ippiCompColorKey* works.

## Applying the Function ippiCompColorKey to Sample Images



## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if one of the step values is less than or equal to 0.

## AlphaCompColorKey

Performs color keying and alpha composition of two images.

## Syntax

```
IppStatusippiAlphaCompColorKey_8u_AC4R(const Ipp8u* pSrc1, int src1Step, Ipp8u alpha1,
const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst, int dstStep, IppiSize
roiSize, Ipp8u colorKey[4], IppiAlphaType alphaType);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointer to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>alpha1</i> , <i>alpha2</i>	Alpha value.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>colorKey</i>	Array of color values.
<i>alphaType</i>	The type of composition to perform (without pre-multiplying). See <a href="#">Table “Possible Values of alphaType Parameter”</a> for more details.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and additionally performs alpha composition (see supported [Table “Types of Image Compositing Operations”](#)) in accordance with the parameter *alphaType*. Note the alpha channel in the *pDst* is not changed after color keying.

The parameter *alphaType* should not be set to the values intended for operations with pre-multiplying.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if one of the step values is less than or equal to 0.
ippStsAlphaTypeErr	Indicates an error condition if <i>alphaType</i> specifies the unsupported type of composition.

# Gamma Correction

---

## GammaFwd

*Performs gamma-correction of the source image with RGB data.*

---

### Syntax

#### Case 1: Not-in-place operation on integer pixel-order data

```
IppStatusippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

#### Case 2: Not-in-place operation on integer planar data

```
IppStatusippiGammaFwd_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P3R	16u_P3R
--------	---------

#### Case 3: Not-in-place operation on floating-point pixel-order data

```
IppStatusippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

32f_C3R
32f_AC4R

#### Case 4: Not-in-place operation on floating-point planar data

```
IppStatusippiGammaFwd_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

#### Case 5: In-place operation on integer pixel-order data

```
IppStatusippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

#### Case 6: In-place operation on integer planar data

```
IppStatusippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_IP3R	16u_IP3R
---------	----------

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatusippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32fvMin, Ipp32fvMax);
```

Supported values for mod:

32f\_C3IR  
32f\_AC4IR

**Case 8: In-place operation on floating-point planar data**

```
IppStatusippiGammaFwd_32f_IP3R(Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, Ipp32fvMin, Ipp32fvMax);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin, vMax</i>	Minimum and maximum values of the input floating-point data.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [gamma-correction](#) of the source image with RGB data. It uses the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

```
for R,G,B < 0.018
    R' = 4.5 * R
    G' = 4.5 * G
    B' = 4.5 * B
for R,G,B ≥ 0.018
```

```
R' = 1.099 * R0.45 - 0.099
G' = 1.099 * G0.45 - 0.099
B' = 1.099 * B0.45 - 0.099
```

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [\[ITU709\]](#) specification.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsGammaRangeErr	Indicates an error condition if the input data bounds are incorrect, that is <i>vMax</i> is less than or equal to <i>vMin</i> .

## GammalInv

*Converts a gamma-corrected R'G'B' image back to the original RGB image.*

### Syntax

#### Case 1: Not-in-place operation on integer pixel-order data

```
IppStatusippiGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

#### Case 2: Not-in-place operation on integer planar data

```
IppStatusippiGammaInv_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_P3R	16u_P3R
--------	---------

#### Case 3: Not-in-place operation on floating-point pixel-order data

```
IppStatusippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

32f_C3R
32f_AC4R

#### Case 4: Not-in-place operation on floating-point planar data

```
IppStatusippiGammaInv_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data**

```
IppStatusippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

**Case 6: In-place operation on integer planar data**

```
IppStatusippiGammaInv_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_IP3R	16u_IP3R
---------	----------

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatusippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32fvMin, Ipp32fvMax);
```

Supported values for mod:

32f_C3IR
32f_AC4IR

**Case 8: In-place operation on floating-point planar data**

```
IppStatusippiGammaInv_32f_IP3R(Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, Ipp32fvMin, Ipp32fvMax);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

*vMin, vMax*

Minimum and maximum values of the input floating-point data.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a `gamma-correctedR'G'B'` image back to the original `RGB` image. It uses the following equations:

**for**  $R', G', B' < 0.0812$

$$R = R'/4.5$$

$$G = G'/4.5$$

$$B = B'/4.5$$

**for**  $R', G', B' \geq 0.0812$

$$R = \left( \frac{R' + 0.099}{1.099} \right)^{2.22}$$

$$G = \left( \frac{G' + 0.099}{1.099} \right)^{2.22}$$

$$B = \left( \frac{B' + 0.099}{1.099} \right)^{2.22}$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [ITU709](#) specification.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsGammaRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is, <code>vMax</code> is less than or equal to <code>vMin</code> .

## Intensity Transformation

The functions described in this section perform different types of intensity transformation including reduction of the intensity levels in each channel of the image, intensity transformation using lookup tables, and mapping high dynamic range image (HDRI) into low dynamic range image (LDRI).

### ReduceBitsGetBufferSize

*Computes the size of the work buffer for the `ippiReduceBits` function.*

#### Syntax

```
IppStatus ippiReduceBitsGetBufferSize(IppChannels ippChan, IppiSize roiSize, int noise,
IppiDitherType dtype, int* pBufferSize);
```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>ippChan</i>	Number of channels in the source images. Possible values: ippC1, ippC3, or ippAC4.
<i>roiSize</i>	Size, in pixels, of the source images.
<i>noise</i>	Number specifying the amount of noise added (as a percentage of the range [0..100]).
<i>dtype</i>	Type of dithering to be used. For the list of supported types, refer to the <a href="#">ippiReduceBits</a> function description.
<i>pBufferSize</i>	Pointer to the computed value of the buffer size, in bytes.

## Description

The function computes the size of the work buffer, in bytes, for the [ippiReduceBits](#) function and stores the result in the *pBufferSize* parameter.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> is less than, or equal to zero.
ippStsChannelErr	Indicates an error when <i>ippChan</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>noise</i> is less than 0, or greater than 100.
ippStsDataTypeErr	Indicates an error when the specified dithering type is not supported.

## See Also

[ReduceBits](#) Reduces the bit resolution of an image.

## ReduceBits

*Reduces the bit resolution of an image.*

---

## Syntax

### Case 1: Operation on data of the same source and destination bit depths

```
IppStatusippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R

```
IppStatusippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);
```

Supported values for mod:

8u_AC4R	16u_AC4R	16s_AC4R
---------	----------	----------

### Case 2: Operation on data of different source and destination bit depths

```
IppStatusippiReduceBits_8u1u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, int noise, int seed, IppiDitherType dtype, Ipp8u threshold, Ipp8u* pBuffer);
```

```
IppStatusippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);
```

Supported values for mod:

16u8u_C1R	16s8u_C1R	32f8u_C1R	32f16u_C1R	32f16s_C1R
16u8u_C3R	16s8u_C3R	32f8u_C3R	32f16u_C3R	32f16s_C3R
16u8u_C4R	16s8u_C4R	32f8u_C4R	32f16u_C4R	32f16s_C4R

```
IppStatusippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);
```

Supported values for mod:

16u8u_AC4R	16s8u_AC4R	32f8u_AC4R	32f16u_AC4R	32f16s_AC4R
------------	------------	------------	-------------	-------------

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>dstBitOffset</i>	Offset (in bits) in the first byte of the destination image row.										
<i>roiSize</i>	Size of the source and destination ROI in pixels.										
<i>noise</i>	The number specifying the amount of noise added. This parameter is set as a percentage of range [0..100].										
<i>seed</i>	The seed value used by the pseudo-random number generation, should be set to 0.										
<i>dtype</i>	The type of dithering to be used. The following types are supported:  <table border="0"> <tr> <td style="vertical-align: top;"><i>ippDitherNone</i></td> <td>No dithering is done</td> </tr> <tr> <td style="vertical-align: top;"><i>ippDitherStucki</i></td> <td>The Stucki's error diffusion dithering algorithm is used</td> </tr> <tr> <td style="vertical-align: top;"><i>ippDitherFS</i></td> <td>The Floyd-Steinberg error diffusion dithering algorithm is used</td> </tr> <tr> <td style="vertical-align: top;"><i>ippDitherJJN</i></td> <td>The Jarvis-Judice-Ninke error diffusion dithering algorithm is used</td> </tr> <tr> <td style="vertical-align: top;"><i>ippDitherBayer</i></td> <td>The Bayer's threshold dithering algorithm is used</td> </tr> </table>	<i>ippDitherNone</i>	No dithering is done	<i>ippDitherStucki</i>	The Stucki's error diffusion dithering algorithm is used	<i>ippDitherFS</i>	The Floyd-Steinberg error diffusion dithering algorithm is used	<i>ippDitherJJN</i>	The Jarvis-Judice-Ninke error diffusion dithering algorithm is used	<i>ippDitherBayer</i>	The Bayer's threshold dithering algorithm is used
<i>ippDitherNone</i>	No dithering is done										
<i>ippDitherStucki</i>	The Stucki's error diffusion dithering algorithm is used										
<i>ippDitherFS</i>	The Floyd-Steinberg error diffusion dithering algorithm is used										
<i>ippDitherJJN</i>	The Jarvis-Judice-Ninke error diffusion dithering algorithm is used										
<i>ippDitherBayer</i>	The Bayer's threshold dithering algorithm is used										
<i>levels</i>	The number of output levels for halftoning (dithering); can be varied in the range [2 .. (1 << <i>depth</i> )], where <i>depth</i> is the bit depth of the destination image.										
<i>threshold</i>	Threshold level for Stucki's dithering for the function <code>ippiReduceBits_8u1u_C1R</code> .										
<i>pBuffer</i>	Pointer to the buffer for internal calculations. To compute the size of the buffer, use the <code>ReduceBitsGetBufferSize</code> function.										

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reduces the number of intensity levels in each channel of the source image *pSrc* and places the results in respective channels of the destination image *pDst*. Note that for floating point source data type, RGB values must be in the range [0..1].

The *levels* parameter sets the resultant number of intensity levels in each channel of the destination image.

If the *noise* value is greater than 0, some random noise is added to the threshold level used in computations. The amplitude of the noise signal is specified by the *noise* parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the *range* value, where

`range = ((1<< depth) - 1)/(levels - 1)`

and `depth` is the bit depth of the source image.

For floating-point data type, `range = 1.0/(levels - 1)`.

**8u to 1u conversion.** Source image is converted to a bitonal image. The function `ippiReduceBits_8u1u_C1R` supports only one dithering algorithm - Stucki's error diffusion. The destination image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter `dstBitOffset` is required to specify the start position of the destination ROI buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsNoiseValErr</code>	Indicates an error condition if <code>noise</code> has an illegal value.
<code>ippStsDitherTypeErr</code>	Indicates an error condition if the specified dithering type is not supported.
<code>ippStsDitherLevelsErr</code>	Indicates an error condition if <code>levels</code> value is out of admissible range.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## See Also

[ReduceBitsGetBufferSize](#) Computes the size of the work buffer for the `ippiReduceBits` function.

## LUT\_GetSize

Computes the size of the LUT specification structure.

### Syntax

```
IppStatus ippiLUT_GetSize(IppiInterpolationType interpolation, IppDataType dataType,
IppChannels channels, IppiSize roiSize, const int nLevels[], int* pSpecSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>interpolation</code>	Interpolation algorithm, possible values are:
<code>ippNearest</code>	Nearest neighbor interpolation.
<code>ippCubic</code>	Cubic interpolation.
<code>ippLinear</code>	Linear interpolation.

<i>dataType</i>	Data type of the image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , or <code>ipp32f</code> .
<i>channels</i>	Number of channels in the image. Possible values are: <code>ippC1</code> , <code>ippC3</code> , <code>ippC4</code> , or <code>ippAC4</code> .
<i>roiSize</i>	Size, in pixels, of the destination ROI.
<i>nLevels</i>	Number of levels, separate for each channel.
<i>pSpecSize</i>	Pointer to the computed size, in bytes, of the specification structure.

## Description

This function computes the size of the specification structure for the `ippiLUT` function. The result is stored in the *pSpecSize* parameter.

For an example on how to use this function, refer to the example provided with the `ippiLUT` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a value less than 1.
<code>ippStsChannelErr</code>	Indicates an error when <i>channel</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsInterpolationErr</code>	Indicates an error when <i>interpolation</i> has an illegal value.

## See Also

**LUT MODIFIED API.** Maps an image by applying intensity transformation.

## LUT\_Init

*Initializes the LUT specification structure.*

---

## Syntax

```
IppStatusippiLUT_Init_8u(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);

IppStatusippiLUT_Init_16u(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);

IppStatusippiLUT_Init_16s(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);

IppStatusippiLUT_Init_32f(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32f* pValues[], const Ipp32f* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>interpolation</i>	Interpolation algorithm, possible values are:
ippNearest	Nearest neighbor interpolation.
ippCubic	Cubic interpolation.
ippLinear	Linear interpolation.
<i>channels</i>	Number of channels in the image. Possible values are: <code>ippC1</code> , <code>ippC3</code> , <code>ippC4</code> , or <code>ippAC4</code> .
<i>roiSize</i>	Size, in pixels, of the destination ROI.
<i>pValues</i>	Pointer to the array with intensity values, separate for each channel.
<i>pLevels</i>	Pointer to the array with level values, separate for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.
<i>pSpec</i>	Pointer to the LUT specification structure.

## Description

This function initializes the specification structure for the `ippiLUT` function. To compute the size of the structure, use the `ippiLUT_GetSize` function.

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Number of level and intensity values are *nLevels*-1.

The *interpolation* parameter defines the mapping algorithm for the LUT function:

- **ippNearest:** Every source pixel  $pSrc(x, y)$  from the range  $[pLevels[k], pLevels[k+1])$  is mapped to the destination pixel  $pDst(x, y)$  which value is equal to  $pValues[k]$ .
- **ippLinear:** Every source pixel  $pSrc(x, y)$  from the range  $[pLevels[k], pLevels[k+1))$  is mapped to the destination pixel  $pDst(x, y)$  which value is computed according to the following formula:

$$pDst(x, y) = pValues[k] + (pSrc(x, y) - pLevels[k]) * (pValues[k+1] - pValues[k]) / (pLevels[k+1] - pLevels[k])$$

- **ippCubic:** Every source pixel  $pSrc(x, y)$  from the range  $[pLevels[k], pLevels[k+1))$  is mapped to the destination pixel  $pDst(x, y)$  which value is computed as

$$pDst(x, y) = A * pSrc(x, y)^3 + B * pSrc(x, y)^2 + C * pSrc(x, y) + D.$$

The function operates on the assumption that the cubic polynomial curve passes through the following four points:

```
([pLevels[k-1], pLevels[k-1])
 ([pLevels[k], pLevels[k])
 ([pLevels[k+1], pLevels[k+1])
```

( $[pLevels[k+2], pLevels[k+2]]$ )

Based on that, coefficients A, B, C, D are computed by solving the following set of linear equations:

$$A*pLevels[k-1]^3 + B*pLevels[k-1]^2 + C*pLevels[k-1] + D = pValues[k-1]$$

$$A*pLevels[k]^3 + B*pLevels[k]^2 + C*pLevels[k] + D = pValues[k]$$

$$A*pLevels[k+1]^3 + B*pLevels[k+1]^2 + C*pLevels[k+1] + D = pValues[k+1]$$

$$A*pLevels[k+2]^3 + B*pLevels[k+2]^2 + C*pLevels[k+2] + D = pValues[k+2]$$

Pixels in the *pSrc* image that are not in the range [*pLevels[0]*, *pLevels[nLevels-1]*) are copied to the *pDst* image without any transformation.

For an example on how to use this function, refer to the example provided with the [ippiLUT](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a value less than 1.
<code>ippStsChannelErr</code>	Indicates an error when <i>channel</i> has an illegal value.
<code>ippStsLUTnofLevelsErr</code>	Indicates an error when <i>nLevels</i> is less than 2.
<code>ippStsInterpolationErr</code>	Indicates an error when <i>interpolation</i> has an illegal value.

## See Also

[LUT MODIFIED API](#). Maps an image by applying intensity transformation.

[LUT\\_GetSize](#) Computes the size of the LUT specification structure.

## LUT

*MODIFIED API*. Maps an image by applying intensity transformation.

---

## Syntax

### Case 1: Not-in-place operation on one-channel integer data

```
IppStatusippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>
---------------------	----------------------	----------------------

### Case 2: Not-in-place operation on multi-channel integer data

```
IppStatusippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

---

```
IppStatusippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

#### **Case 3: Not-in-place operation on one-channel floating-point data**

```
IppStatusippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

#### **Case 4: Not-in-place operation on multi-channel floating-point data**

```
IppStatusippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

32f_C3R	32f_AC4R
---------	----------

```
IppStatusippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

#### **Case 5: In-place operation on one-channel integer data**

```
IppStatusippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR
---------	----------	----------

#### **Case 6: In-place operation on multi-channel integer data**

```
IppStatusippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR

```
IppStatusippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C4IR	16u_C4IR	16s_C4IR
---------	----------	----------

#### **Case 7: In-place operation on one-channel floating-point data**

```
IppStatusippiLUT_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

#### **Case 8: In-place operation on multi-channel floating-point data**

```
IppStatusippiLUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for mod:

32f_C3IR
----------

`32f_AC4IR`

```
IppStatusippiLUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination mage ROI for the in-place operation.
<code>srcDstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<code>roiSize</code>	Size of the source ROI, in pixels.
<code>pSpec</code>	Pointer to the LUT specification structure.

## Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

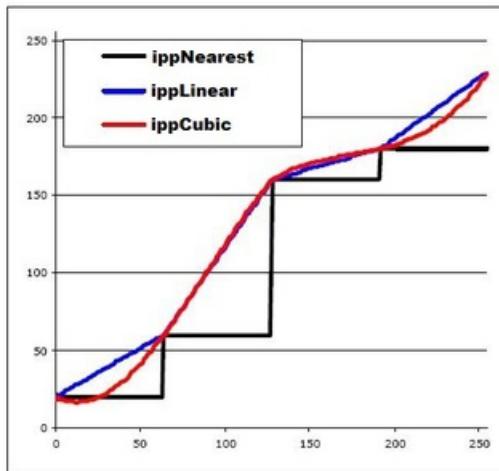
---

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

Before using this function, you need to compute the size of the specification structure using the [LUT\\_GetSize](#) function and initialize the structure using [LUT\\_Init](#).

This function performs intensity transformation of the source image `pSrc` using the lookup table (LUT) specified by the arrays `pLevels`, `pValues`, and `interpolation` type specified in the [LUT\\_Init](#) function when `pSpec` is initialized.

The figure below shows particular curves that are used in all the `ippiLUT` function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.



## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error when <code>roiSize</code> has a field with a value less than 1.
ippStsStepErr	Indicates an error when <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.
ippStsBadArgErr	Indicates an error when <code>pSpec</code> initialization is incorrect.

## Example

The code example below demonstrates how to use `LUT_GetSize`, `LUT_Init`, and `ippiLUT` functions.

```
#include "ippcore.h"
#include "ippi.h"
#include <iostream>
#include <iomanip>

void func_LUTLinear()
{
    IppStatus status;
    Ipp32f pSrc[8 * 8];
    int srcStep = 8 * sizeof(Ipp32f);
    IppiSize roiSize = { 8, 8 };

    Ipp32f pDst[8 * 8];
    int dstStep = 8 * sizeof(Ipp32f);
    Ipp32f pLevels[5] = { 0.0, 0.128, 0.256, 0.512, 1.0 };
    const Ipp32f* ppLevels[1] = { pLevels };
    Ipp32f pValues[5] = { 0.2, 0.4, 0.6, 0.8, 1.0 };
    const Ipp32f* ppValues[1] = { pValues };
    int nLevels[1] = { 5 };
    int specSize;
    IppiLUT_Spec* pSpec;

    status = ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize);
```

```

    std::cout << "pSrc:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << std::fixed << std::setprecision(2) <<pSrc[roiSize.width * row + col] <<
"\t";
        }
        std::cout << "\n";
    }

    ippiLUT_GetSize(ipplinear, ipp32f, ippC1, roiSize, nLevels, &specSize);

    pSpec = (IppiLUT_Spec*)ippMalloc(specSize);

    ippiLUT_Init_32f(ipplinear, ippC1, roiSize, ppValues, ppLevels, nLevels, pSpec);
    status = ippiLUT_32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize, pSpec);

    std::cout << "pDst:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << pDst[roiSize.width * row + col] << "\t";
        }
        std::cout << "\n";
    }

    ippFree(pSpec);
}

```

### Result:

pSrc:

```

0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00

```

pDst:

```

0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20

```

### See Also

[Regions of Interest in Intel IPP](#)

[LUT\\_GetSize](#) Computes the size of the LUT specification structure.

[LUT\\_Init](#) Initializes the LUT specification structure.

## LUTPalette, LUTPaletteSwap

*Maps an image by applying intensity transformation in accordance with a palette table.*

## Syntax

## Case 1: Operations on one-channel data

```
IppStatusippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<dstDatatype>* pTable,  
int nBitSize);
```

Supported values for mod:

8u_C1R	16u_C1R
8u32u_C1R	16u8u_C1R
	16u32u_C1R

```
IppStatus ippILUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, const Ipp8u* pTable, int nBitSize);
```

Supported values for mod:

8u24u\_C1R 16u24u\_C1R

## Case 2: Operations on multi-channel data

```
IppStatusippiILUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const Ipp<datatype>* const pTable[3], int nBitSize);
```

Supported values for mod:

8u\_C3R 16u\_C3R  
8u\_AC4R 16u\_AC4R

```
IppStatusippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const Ipp<datatype>* const pTable[4], int nBitSize);
```

Supported values for mod:

8u C4R 16u C4R

## Include Files

ippi.h

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

nsrc

Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pTable</i>	Pointer to the palette table, or an array of pointers to the palette tables for each source channel.
<i>nBitSize</i>	Number of significant bits in the source image.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiLUTPalette` performs intensity transformation of the source image *pSrc* using the palette lookup table *pTable*. This table is a vector with  $2^{nBitSize}$  elements that contain intensity values specified by the user. The function uses *nBitSize* lower bits of intensity value of each source pixel as an index in the *pTable* and assigns the correspondent intensity value from the table to the respective pixel in the destination image *pDst*. The number of significant bits *nBitSize* should be in the range [1, 8] for functions that operate on *8u* source images, and [1, 16] for functions that operate on *16u* source images.

Some function flavors that operate on the 3-channel source image additionally create a 4-th channel - alpha channel - in the destination image and place it at first position. The channel values of the alpha channel can be set to the arbitrary constant value *alphaValue*. If this value is less than 0 or greater than the upper boundary of the data range, the channel values are not set.

The function flavor `ippiLUTPaletteSwap` reverses the order of channels in the destination image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsOutOfRangeErr</code>	Indicates an error if <i>nBitSize</i> is out of the range.

## Example

The code example below shows how to use the function `ippiLUTPalette_8u32u_C1R`.

```
#include "ippcore.h"
#include "ippi.h"
#include <iostream>

void func_LUTPalette()
{
    IppStatus status;
    Ipp8u pSrc[8 * 8];
    int srcStep = 8 * sizeof(Ipp8u);
    IppiSize roiSize = { 8, 8 };

    Ipp32u pDst[8 * 8];
    int dstStep = 8 * sizeof(Ipp32f);
    int nBitSize = 3;
```

```

Ipp32u pTable[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };

status =ippiImageJaehne_8u_C1R(pSrc, srcStep, roiSize);

std::cout << "pSrc:\n";
for (int row = 0; row < roiSize.height; row++) {
    for (int col = 0; col < roiSize.width; col++) {
        std::cout << (int)pSrc[roiSize.width * row + col] << "\t";
    }
    std::cout << "\n";
}

status =ippiLUTPalette_8u32u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, pTable, nBitSize);

std::cout << "pDst:\n";
for (int row = 0; row < roiSize.height; row++) {
    for (int col = 0; col < roiSize.width; col++) {
        std::cout << pDst[roiSize.width * row + col] << "\t";
    }
    std::cout << "\n";
}
}

```

**Result:**

```

pSrc:
1 68 165 209 209 165 68 1
68 209 255 250 250 255 209 68
165 255 227 188 188 227 255 165
209 250 188 141 141 188 250 209
209 250 188 141 141 188 250 209
165 255 227 188 188 227 255 165
68 209 255 250 250 255 209 68
1 68 165 209 209 165 68 1

```

```

pDst:
2 5 6 2 2 6 5 2
5 2 8 3 3 8 2 5
6 8 4 5 5 4 8 6
2 3 5 6 6 5 3 2
2 3 5 6 6 5 3 2
6 8 4 5 5 4 8 6
5 2 8 3 3 8 2 5
2 5 6 2 2 6 5 2

```

**ToneMapLinear, ToneMapMean**

Maps an HDRI image to the LDRI image.

**Syntax**

```

IppStatusippiToneMapLinear_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);

IppStatusippiToneMapMean_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);

```

## Include Files

ippcc.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the HDRI source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the LDRI destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

They both operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the source high dynamic range image (HDRI) *pSrc* into low dynamic range image (LDRI) *pDst*. Pixel values of the source image must be positive.

The function `ippiToneMapLinear` implements the Linear Scale-Factor method converting each source pixel *pSrc[i]* in accordance with the formula:

$$pSrc[i] = pSrc[i]/Lmax, \quad Lmax = \max\{pSrc[i]\}.$$

The function `ippiToneMapMean` implements the Mean Value method converting each source pixel *pSrc[i]* in accordance with the formula:

$$pSrc[i] = 0.5*pSrc[i]/Lave, \quad Lave = \text{average}\{pSrc[i]\}.$$

If the value of *Lmax* or *Lave* is less than 0, then the function does not perform the operation and returns the warning message.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsNoOperation</code>	Indicates a warning if the values of <i>Lmax</i> or <i>Lave</i> are less than 0.

# Threshold and Compare Operations

7

This chapter describes the Intel® IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions.

## Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified *threshold*.

The type of comparison operation used to threshold pixel values is specified by the *ippCmpOp* parameter; this operation can be either “greater than” or “less than” (see [Structures and Enumerators](#) in Chapter 2 for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

### Threshold

*Performs thresholding of pixel values in an image buffer.*

#### Syntax

##### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

##### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatusippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
<i>ippCmpOp</i>	The operation specified for comparing pixel values and the <i>threshold</i> . Comparison for either “less than” or “greater than” can be used.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value according to the type of comparison operation specified in the *ippCmpOp*. The following values for *ippCmpOp* are possible:

- **ippCmpLess** specifies the “less than” comparison and defines the *threshold* value as a lower bound. Comparison is performed by the following formula:

$$pDst[n] = \begin{cases} \text{threshold}, & pSrc[n] < \text{threshold} \\ pSrc[n], & \text{otherwise} \end{cases}$$

- **ippCmpGreater** specifies the “greater than” comparison and defines the *threshold* value as an upper bound. Comparison is performed by the following formula:

$$pDst[n] = \begin{cases} \text{threshold}, & pSrc[n] > \text{threshold} \\ pSrc[n], & \text{otherwise} \end{cases}$$

If the result of comparison is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value
<code>ippStsNotSupportedModeErr</code>	Indicates an error if the comparison mode is not supported.

## Example

The code example below shows how to use the `ippiThreshold_8u_C1R` function.

```
void func_threshold()
{
    IppiSize ROI = {5,4};
    Ipp8u src[9*4] = {1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1};

    Ipp8u dst[9*4];
    Ipp8u threshold = 6;
   ippiThreshold_8u_C1R(src, 9, dst, 9, ROI, threshold, ippCmpGreater);
}
```

**Result:**

dst	
1 2 4 6 6 8 4 2 1	
1 2 4 6 6 8 4 2 1	
1 2 4 6 6 8 4 2 1	
1 2 4 6 6 8 4 2 1	

## Threshold\_GT

Performs thresholding of pixel values in an image, using the comparison for "greater than".

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatusippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

#### Case 4: In-place operation on multi-channel data

```
IppStatusippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Include Files

ippi.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LT

*Performs thresholding of pixel values in an image buffer, using the comparison for "less than".*

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for *mod*:

8u\_C1R

16u\_C1R

16s\_C1R

32f\_C1R

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatusippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatusippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.

*threshold*

The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "less than". If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_Val

*Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.*

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------	----------------------

### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for *mod*:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

**Case 3: In-place operation on one-channel data**

```
IppStatusippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3], IppCmpOp
ippCmpOp);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

**Include Files**

`ippi.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI (for the in-place operation).
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>threshold</code>	The threshold value to use for each pixel. In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
<code>value</code>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
<code>ippCmpOp</code>	The operation to use for comparing pixel values and the <code>threshold</code> . Comparison for either "less than" or "greater than" can be used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_GTVal

*Performs thresholding of pixel values in an image. Pixels that are greater than threshold, are set to a specified value.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3]);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

#### **Case 3: In-place operation on one-channel data**

```
IppStatusippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

#### **Case 4: In-place operation on multi-channel data**

```
IppStatusippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

```
IppStatusippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
---------	----------	----------	----------

### **Include Files**

ippi.h

### **Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### **Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LTVal

Performs thresholding of pixel values in an image.

Pixels that are less than *threshold*, are set to a specified value.

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatusippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

**Case 3: In-place operation on one-channel data**

```
IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

**Case 4: In-place operation on multi-channel data**

```
IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

```
IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
---------	----------	----------	----------

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "less than". If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LTValGTVal

*Performs double thresholding of pixel values in an image buffer.*

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
IppStatusippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> thresholdLT,
Ipp<datatype> valueLT, Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Not-in-place operation on multi-channel data

```
IppStatusippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3],
const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3], const Ipp<datatype>
valueGT[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Case 3: In-place operation on one-channel data

```
IppStatusippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype>
thresholdGT, Ipp<datatype> valueGT);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------	----------

### Case 4: In-place operation on multi-channel data

```
IppStatusippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3],
const Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);
```

Supported values for mod

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>thresholdLT</i>	The lower threshold value to use for each pixel. In case of multi-channel data, an array of three lower threshold values (one for each color channel) is used.
<i>valueLT</i>	The lower output value to be set for each pixel that is less than <i>thresholdLT</i> . In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
<i>thresholdGT</i>	The upper threshold value to use for each pixel. In case of multi-channel data, an array of three upper threshold values (one for each color channel) is used.
<i>valueGT</i>	The upper output value to be set for each pixel that exceeds <i>thresholdGT</i> . In case of multi-channel data, an array of three upper output values (one for each color channel) is used.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using two specified levels *thresholdLT* and *thresholdGT*. Pixel values in the source image are compared to these levels. If the pixel value is less than *thresholdLT*, the corresponding output pixel is set to *valueLT*. If the pixel value is greater than *thresholdGT*, the output pixel is set to *valueGT*. Otherwise, it is set to the source pixel value. The value of *thresholdLT* should be less than or equal to *thresholdGT*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsThresholdErr</i>	Indicates an error when <i>thresholdLT</i> is greater than <i>thresholdGT</i> .
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Example

The code example below illustrates thresholding with two levels.

```
IppStatus threshold( void ) {
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    int i;

    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;

    returnippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi, 2,1,6,7 );
}
```

The destination image `x` contains:

```
01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07
```

## ComputeThreshold\_Otsu

*Computes the value of the Otsu threshold.*

### Syntax

```
IppStatusippiComputeThreshold_Otsu_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
roiSize, Ipp8u* pThreshold);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>pThreshold</code>	Pointer to the Otsu threshold value.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the Otsu threshold ([\[Otsu79\]](#)) for the source image `pSrc` in accordance with the following formula:

$$\frac{\min(\sigma(pSrc(x,y) \leq T) + \sigma(pSrc(x,y) > T))}{T}$$

where  $0 \leq x < roiSize.width$ ,  $0 \leq y < roiSize.height$ , and  $T$  is the Otsu threshold.

$$\sigma(pSrc(x,y)) = \sqrt{\frac{\sum_{\substack{x < roiSize.width \\ y < roiSize.height \\ x,y \geq 0}} (pSrc(x,y) - mean)^2}{roiSize.height \cdot roiSize.width}}$$

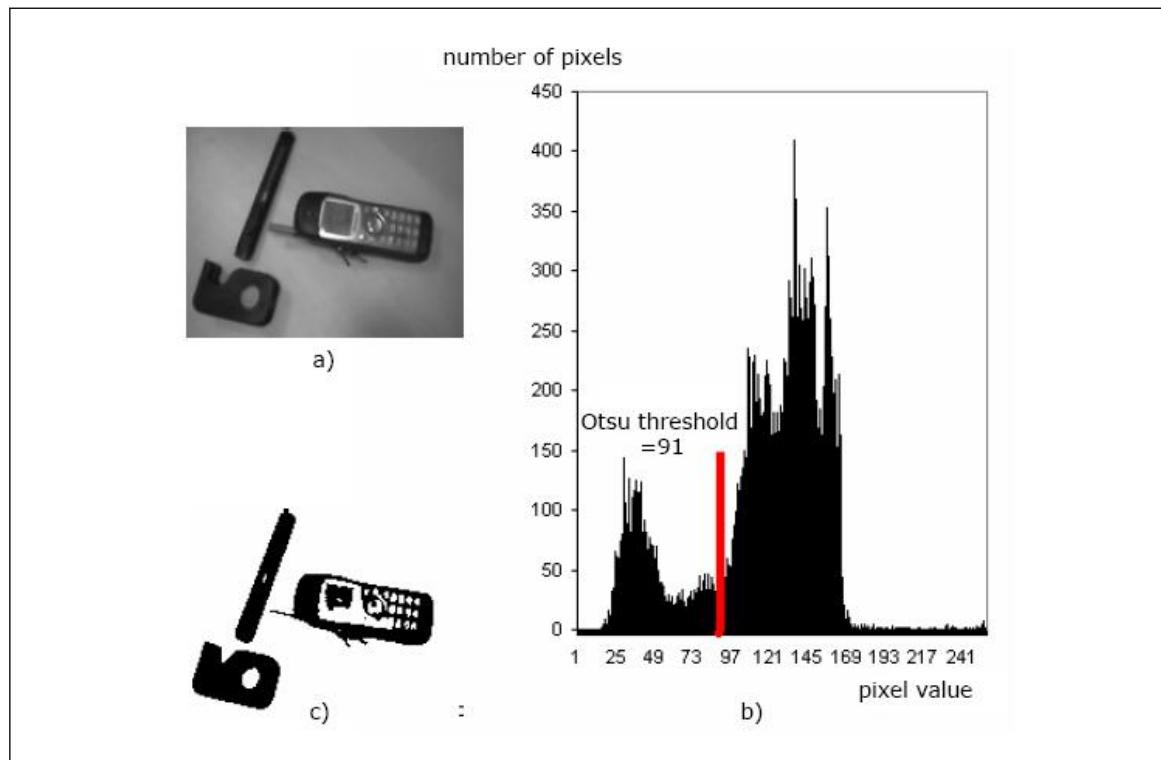
where

$$mean = \frac{\sum_{\substack{x < roiSize.width \\ y < roiSize.height \\ x,y \geq 0}} pSrc(x,y)}{roiSize.height \cdot roiSize.width}$$

The computed Otsu threshold can be used in the thresholding functions described above.

For example, the following figures shows how the Otsu threshold can be used for background/foreground selection. The figure a) below shows the initial image, the figure b) shows the histogram of the initial image with the red line indicating the computed Otsu threshold. The figure c) demonstrates the resulting image obtained by applying the function `ippiThreshold_8u_C1R` with a computed Otsu threshold value to the source image.

### Using Otsu Threshold for Background/Foreground Selection



### Return Values

`ippStsNoErr`

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> is less than or equal to 0.

## Compare Operations

---

This section describes functions that compare images. Each compare function writes its results to a one-channel `Ipp8u` output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0. You can compare either two images, or an image and a constant value, using the following compare conditions: “greater”, “greater or equal”, “less”, “less or equal”, “equal”. Compare condition is specified as a function argument of `IppCmpOp` type (see [Structures and Enumerators](#) in Chapter 2 for more information). Images containing floating-point data can also be compared for being equal within a given tolerance *eps*.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.

### Compare

*Compares pixel values of two images using a specified compare operation.*

---

#### Syntax

```
IppStatusippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize,
IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

```
IppStatusippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize,
IppCmpOp ippCmpOp);
```

Supported values for *mod*:

8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R
---------	----------	----------	----------

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function compares the corresponding pixels of ROI in two source images *pSrc1, pSrc2* using the *ippCmpOp* compare operation, and writes the results to a one-channel Ipp8u image *pDst*. If the result of the compare is true, the corresponding output pixel is set to an IPP\_MAX\_8U value; otherwise, it is set to 0.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>src1Step, src2Step, or dstStep</i> has a zero or negative value.

## CompareC

*Compares pixel values of a source image to a given value using a specified compare operation.*

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Operation on multi-channel data

```
IppStatusippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

```
IppStatusippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

```
IppStatusippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R
---------	----------	----------	----------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

## Description

This function operates with ROI ([Regions of Interest in Intel IPP](#)).

This function compares pixels of the each channel of the source image ROI *pSrc* to a given *value* specified for each channel using the *ippCmpOp* compare operation, and writes the results to a one-channel Ipp8u image *pDst*. If the result of the compare is true, that is, all pixels of all channels meet the specified condition, then the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

**ippStsStepErr**

Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

## Example

The code example below shows how to use the comparison function and create a mask image:

```
IppStatus compare ( void ) {
    Ipp8u x[5*4], y[5*4];
    IppiSize roi = {5,4};
    int i;
    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
    returnippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi, ippCmpGreater );
}
```

The mask image *y* contains:

```
00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF
```

## CompareEqualEps

*Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance *eps*.*

### Syntax

```
IppStatusippiCompareEqualEps_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
```

```
IppStatusippiCompareEqualEps_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

*pSrc1, pSrc2*

Pointers to the source image ROIs.

*src1Step, src2Step*

Distances in bytes between starts of consecutive lines in the source images.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>eps</i>	The tolerance value.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* are equal within the tolerance *eps*, and writes the results to a one-channel Ipp8u image *pDst*. If the absolute value of difference of the pixel values in *pSrc1* and *pSrc2* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an IPP\_MAX\_8U value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the *eps* tolerance for the compare condition to be true.

This function processes images with floating-point data only.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.
ippStsEpsValErr	Indicates an error condition if <i>eps</i> has a negative value.

## CompareEqualEpsC

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance *eps*.

---

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

### Case 2: Operation on multi-channel data

```
IppStatusippiCompareEqualEpsC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

```
IppStatusippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

```
IppStatusippiCompareEqualEpsC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>eps</i>	The tolerance value.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if pixel values of the source image ROI *pSrc* are equal to a given constant *value* within the tolerance *eps*, and writes the results to a one-channel Ipp8u image *pDst*. If the absolute value of difference between the pixel value in *pSrc* and *value* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an IPP\_MAX\_8U value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of *value* must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsEpsValErr	Indicates an error condition if <i>eps</i> has a negative value.

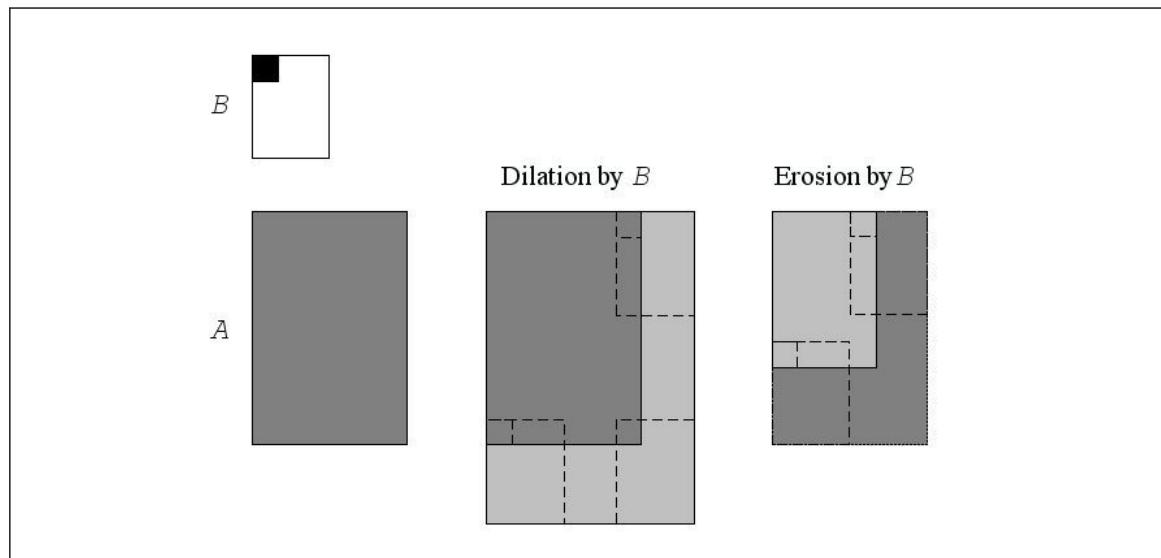
# Morphological Operations

This chapter describes the Intel® IPP image processing functions that perform morphological operations on images.

Generally, the *erosion* and *dilation* smooth the boundaries of objects without significantly changing their area. *Opening* and *closing* smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image  $A$  called the *object of interest* and a kernel element  $B$  called the *structuring element*. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element  $B$  is most often a square or a circle, but it could be of any shape. Just like in convolution,  $B$  is a kernel or template with an anchor point. [Figure "Dilation and Erosion of A by B"](#) shows dilation and erosion of object  $A$  by  $B$ . In the figure,  $B$  is rectangular with an anchor point at upper left shown as a dark square.

## Dilation and Erosion of $A$ by $B$



Let  $B_t$  is the SE with pixel  $t$  in the anchor position,  $\bar{B}$  is transpose of the SE.

Dilation of binary image  $A$   $\{A(t) = 1, t \in A; 0 - \text{otherwise}\}$  by binary SE  $B$  is

$$A \oplus B = \{t : B_t \cap A \neq \emptyset\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of  $B$  is marked “on”, if at least one pixel of  $B$  is inside of  $A$ .

Erosion of the binary image  $A$  by the binary SE  $B$  is

$$A \ominus B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of  $B$  is marked “on”, if  $B$  is entirely within  $A$ .

Generalization of dilation and erosion for the gray-scale image  $A$  and the binary SE  $B$  is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} A(u) \right\}, \quad A \ominus B = \left\{ \min_{u \in B_t \cap A} A(u) \right\}$$

Generalization of dilation and erosion for the gray-scale image  $A$  and the gray-scale SE  $B$  is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} (A(u) + B(u - t)) \right\}, \quad A \ominus B = \left\{ \min_{u \in B_t \cap A} (A(u) + B(u - t)) \right\}$$

Opening operation of  $A$  by  $B$  is  $A \circ B = (A \ominus B) \oplus \bar{B}$ .

Closing operation of  $A$  by  $B$  is  $A \bullet B = (A \oplus B) \ominus \bar{B}$ .

Top-hat operation of  $A$  by  $B$  is  $A - A \circ B$ .

Black-hat operation of  $A$  by  $B$  is  $A \bullet B - A$ .

Black-hat operation of  $A$  by  $B$  is  $A \oplus B - A \ominus B$ .

Morphological reconstruction [Vincent93] $\rho_A(C)$  of an image  $A$  from the image  $C$ ,  $A(t) \geq C(t) \forall t$  by dilation with the mask  $B$  is an image

$$C_k : k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \min \{ (C_i \oplus B)(t), A(t) \}$$

Morphological reconstruction  $\rho_A(C)$  of an image  $A$  from the image  $C$ ,  $A(t) \leq C(t) \forall t$  by erosion with the mask  $B$  is an image

$$C_k : k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \max \{ (C_i \ominus B)(t), A(t) \}$$

[Figure "Morphological Operations Performed by Intel IPP"](#) presents the results of different morphological operations applied to the initial image. In these operations, the SE is a matrix of 3x3 size with the following values:

$$\begin{bmatrix} -8 & 0 & -8 \\ 0 & 8 & 0 \\ -8 & 0 & -8 \end{bmatrix}$$

for common and advanced morphology, and

$$\begin{bmatrix} -5 & 0 & -5 \\ 0 & 5 & 0 \\ -5 & 0 & -5 \end{bmatrix}$$

for gray morphology.

The anchor cell is in the center cell (1,1) of the matrix.

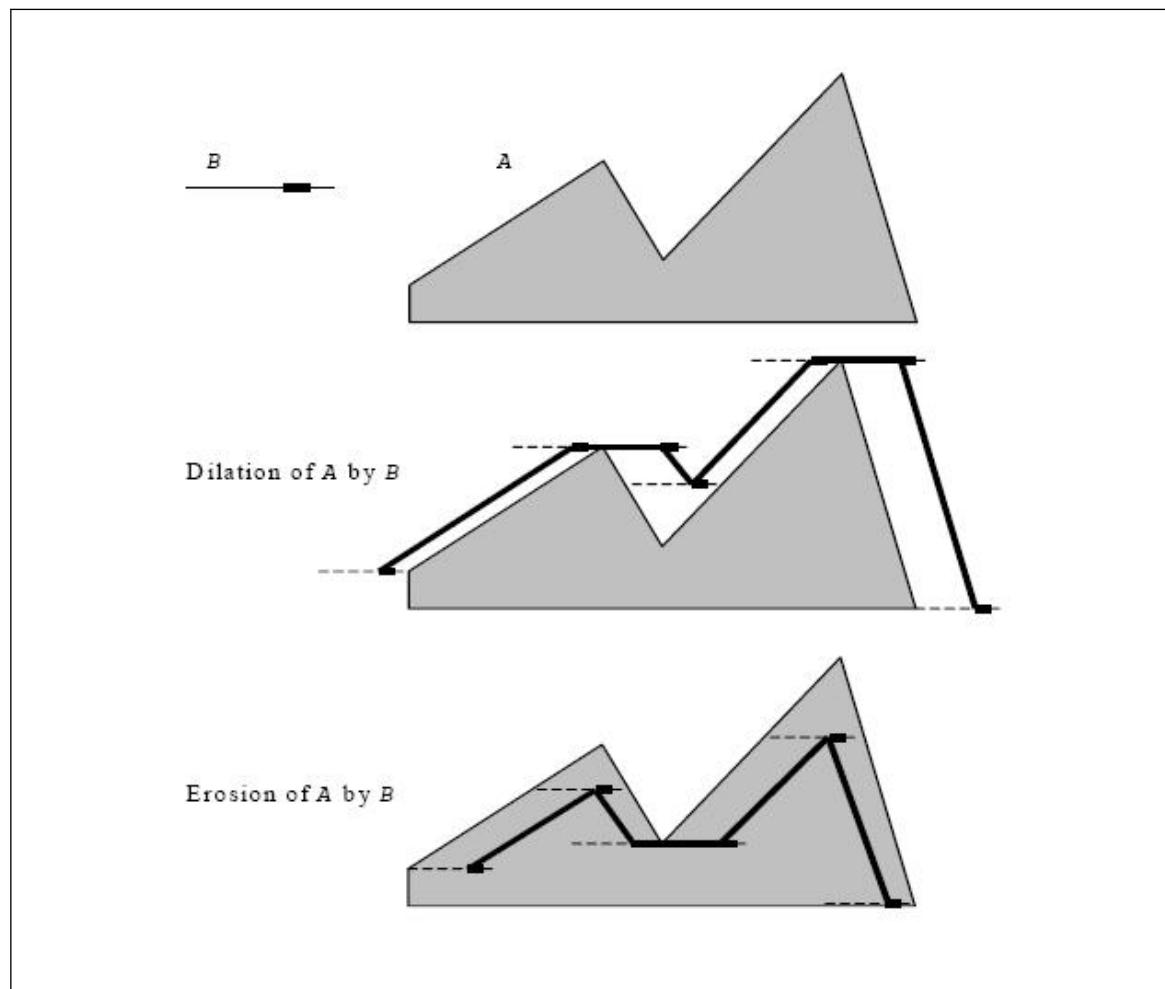
### Morphological Operations Performed by Intel IPP



## Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element  $B$ . [Figure "1D Cross Section of Dilation and Erosion of A by B"](#) is a 1D cross section of dilation and erosion of a grayscale image  $A$  by a flat structuring element  $B$ . In the figure,  $B$  has an anchor slightly to the right of the center as shown by the dark mark on  $B$ .

### 1D Cross Section of Dilation and Erosion of A by B



In [Figure "1D Cross Section of Dilation and Erosion of A by B"](#) above, dilation is mathematically

$$\sup_{y \in B_t} A$$

and erosion is

$$\inf_{y \in B_t} A$$

## Dilate3x3

Performs dilation of an image using a 3x3 mask.

## Syntax

```
IppStatusippiDilate3x3_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its eight neighboring pixels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Dilate

---

Performs dilation of an image using a specified mask.

## Syntax

```
IppStatusippiDilate_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```

IppStatusippiDilate_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiDilate_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u*
pBuffer);

```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>borderType</i>	Type of the border; supported values:  ippBorderDefault      The border is set to <code>ippBorderConst</code> with <code>borderValue= MIN_VALUE, where</code> <code>MIN_VALUE=IPP_MIN_8U/16U/16S/32F/1U</code>

	ippBorderRepl	Border is replicated from the edge pixels.
	ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.
	ippBorderConst	Values of all border pixels are set to a constant.
	ippBorderInMem	Border is obtained from the source image pixels in memory.
		Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
borderValue		Pointer to the vector of values for the constant border type.
srcBitOffset		Offset in the first byte of the source image row.
dstBitOffset		Offset in the first byte of the destination image row.
pMorphSpec		Pointer to the morphology specification structure.
pBuffer		Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsNotEvenStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a not pixel multiple value.
ippStsBadArgErr	Indicates an error condition if <i>borderType</i> has an incorrect value.

## DilateBorder

---

*Performs dilation of an image.*

---

## Syntax

```
IppStatusippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R 32f\_C3R

```
IppStatusippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R 32f\_C4R

```
IppStatusippiDilateBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:

	ippBorderRepl	Border is replicated from the edge pixels.
	ippBorderInMem	Border is obtained from the source image pixels in memory.
<i>borderValue</i>		Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>		Pointer to the specification structure.
<i>pBuffer</i>		Pointer to the external buffer required for dilation operations.

## Description

This function operates with ROI.

This function expands a rectangular ROI area inside a one-channel two-dimensional image using a mask specified in the specification structure *pSpec*. Before using this function, you need to initialize the structure using the `MorphologyBorderInit` function.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error when: <ul style="list-style-type: none"> <li><i>roiSize</i> has a field with a zero or negative value</li> <li><i>roiSize.width</i> is more than the maximum ROI <i>roiWidth</i> passed to the initialization function</li> </ul>
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width* &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error when one of the step values for 16-bit integer images is not divisible by 2.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsInplaceModeNotSupportedE	Indicates an error when the <i>pSrc</i> pointer is equal to the <i>pDst</i> pointer.

## Example

The code example below demonstrates how to use the `ippiMorphologyBorderGetSize_16u_C1R`, `ippiMorphologyBorderInit_16u_C1R`, and `ippiDilateBorder_16u_C1R` functions to perform dilation of the source image.

```
IppStatus func_MorfDilateBorder()
{
    IppiMorphState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
```

```

Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2,
                      5, 1, 2, 1, 2,
                      1, 2, 1, 2, 1,
                      1, 2, 1, 2, 1,
                      2, 1, 5, 1, 2};

Ipp16u pDst[5*5];
int srcStep = 5*sizeof(Ipp16u);
int dstStep = 5*sizeof(Ipp16u);
int dstSize = 5;
IppStatus status = ippStsNoErr;
int specSize = 0, bufferSize = 0;
IppiBorderType borderType= ippBorderRepl;
Ipp16u borderValue = 0;

status =ippiMorphologyBorderGetSize_16u_C1R( roiSize, maskSize, &specSize, &bufferSize );
if (status != ippStsNoErr) return status;

pSpec = (IppiMorphState*)ippsMalloc_8u(specSize);
pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);

status =ippiMorphologyBorderInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
if (status != ippStsNoErr) {
    ippsFree(pBuffer);
    ippsFree(pSpec);
    return status;
}

status =ippiDilateBorder_16u_C1R( pSrc, srcStep, pDst, dstStep, roiSize, borderType,
borderValue, pSpec, pBuffer);

ippsFree(pBuffer);
ippsFree(pSpec);
return status;
}

```

The result is as follows:

```

pDst->
5 5 4 4 2
5 5 4 4 2
5 5 2 2 2
2 5 5 5 2
2 5 5 5 2

```

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphologyBorderInit](#) Initializes the morphology specification structure for erosion or dilation operations.

## DilateGetBufferSize

*Computes the size of the working buffer for the Dilate function.*

## Syntax

```
IppStatusippiDilateGetBufferSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType
datatype, int numChannels, IppSizeL* pBufferSize);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>dataType</i>	Data type for the morphological function.
<i>numChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the buffer size value for the morphological initialization function.

## Description

This function computes the size of the working buffer required for the `ippiDilate` functions with the `_L` suffix.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the pointers is NULL.
<code>ippStsSizeErr</code>	Width of the image, or width or height of the structuring element is less than, or equal to zero.

## DilateGetSpecSize

---

*Computes the size of the internal state or specification structure for the Dilate function.*

## Syntax

```
IppStatusippiDilateGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppSizeL*
pSpecSize);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>pSpecSize</i>	Pointer to the size of the specification structure.

## Description

This function computes the size of the specification structure required for the `ippiDilate` functions with the `_L` suffix.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Width of the image, or width or height of the structuring element is less than, or equal to zero.

## DilateInit

---

*Initializes the internal state or specification structure for the Dilate function.*

---

## Syntax

```
IppStatusippiDilateInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize,  
IppiMorphStateL* pMorphSpec);
```

## Include Files

`ippcv_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>pMask</i>	Pointer to the structuring element (mask).
<i>maskSize</i>	Size of the structuring element.
<i>pMorphSpec</i>	Pointer to the morphology specification structure.

## Description

This function initializes the internal state or specification structure for the `ippiDilate` functions with the `_L` suffix.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.
ippStsAnchorErr	Anchor point is outside the structuring element.

## Erode3x3

---

Performs erosion of an image using a 3x3 mask.

---

### Syntax

```
IppStatusippiErode3x3_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppiSize roiSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Erode

*Performs erosion of an image using a specified mask.*

### Syntax

```
IppStatusippiErode_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL
dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[1],
const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL
dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[3],
const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL
dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[4],
const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiErode_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u*
pBuffer);
```

### Include Files

ippcv\_1.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>borderType</i>	Type of the border; supported values:  ippBorderDefault      The border is set to <code>ippBorderConst</code> with <code>borderValue= MAX_VALUE</code> , where <code>MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U</code> ippBorderRepl      Border is replicated from the edge pixels. ippBorderMirror      Border pixels are mirrored from the source image boundary pixels. ippBorderConst      Values of all border pixels are set to a constant. ippBorderInMem      Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>	Pointer to the vector of values for the constant border type.
<i>srcBitOffset</i>	Offset in the first byte of the source image row.
<i>dstBitOffset</i>	Offset in the first byte of the destination image row.
<i>pMorphSpec</i>	Pointer to the morphology specification structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.

Source and destination images can be of different sizes, but the ROI size is the same for both images (*dstRoiSize*). The output pixel is set to the minimum of the corresponding input pixel and its neighboring pixels that are picked out by the non-zero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

ippStsNotEvenStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a not pixel multiple value.
ippStsBadArgErr	Indicates an error condition if <i>borderType</i> has an incorrect value.

## ErodeBorder

---

Performs erosion of an image.

---

### Syntax

```
IppStatusippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R 32f\_C3R

```
IppStatusippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R 32f\_C4R

```
IppStatusippiErodeBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for <code>lu_C1R</code> flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:  ippBorderRepl      Border is replicated from the edge pixels. ippBorderInMem      Border is obtained from the source image pixels in memory.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer required for erosion operations.

## Description

This function operates with ROI.

This function performs erosion of a rectangular ROI area inside a one-channel 2D image using a mask specified in the specification structure *pSpec*. Before using this function, you need to initialize the structure using the [MorphologyBorderInit](#) function.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when: <ul style="list-style-type: none"><li>• <i>roiSize</i> has a field with a zero or negative value</li><li>• <i>roiSize.width</i> is more than the maximum ROI <i>roiWidth</i> passed to the initialization function</li><li>• <i>srcBitOffset</i> or <i>dstBitOffset</i> is less than zero</li></ul>
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> < <i>pixelSize</i> >.
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values for 16-bit integer images is not divisible by 2.
<code>ippStsBorderErr</code>	Indicates an error when <i>borderType</i> has an illegal value.
<code>ippStsInplaceModeNotSupportedE</code>	Indicates an error when the <i>pSrc</i> pointer is equal to the <i>pDst</i> pointer.

## See Also

[Regions of Interest in Intel IPP](#)  
[User-defined Border Types](#)

**MorphologyBorderInit** Initializes the morphology specification structure for erosion or dilation operations.

## ErodeGetSize

*Computes the size of the working buffer for the Erode function.*

### Syntax

```
IppStatusippiErodeGetSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType
datatype, int numChannels, IppSizeL* pBufferSize);
```

### Include Files

ippcv\_1.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>dataType</i>	Data type for the morphological function.
<i>numChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the buffer size value for the morphological initialization function.

### Description

This function computes the size of the working buffer required for the `ippiErode` functions with the `_L` suffix.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Width of the image, or width or height of the structuring element is less than, or equal to zero.

## ErodeGetSpecSize

*Computes the size of the internal state or specification structure for the Erode function.*

### Syntax

```
IppStatusippiErodeGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppSizeL*
pSpecSize);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>pSpecSize</i>	Pointer to the size of the specification structure.

## Description

This function computes the size of the specification structure required for the `ippiErode` functions with the `_L` suffix.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Width of the image, or width or height of the structuring element is less than, or equal to zero.

## ErodeInit

---

*Initializes the internal state or specification structure for the Erode function.*

## Syntax

```
IppStatusippiErodeInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize,  
IppiMorphStateL* pMorphSpec);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>pMask</i>	Pointer to the structuring element (mask).
<i>maskSize</i>	Size of the structuring element.
<i>pMorphSpec</i>	Pointer to the morphology specification structure.

## Description

This function initializes the internal state or specification structure for the `ippiErode` functions with the `_L` suffix.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Width of the image, or width or height of the structuring element is less than, or equal to zero.
<code>ippStsAnchorErr</code>	Anchor point is outside the structuring element.

## GrayDilateBorder

*Performs gray-kernel dilation of an image.*

### Syntax

```
IppStatusippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
IppStatusippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);
```

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI.
<code>border</code>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<code>pState</code>	Pointer to the morphology state structure.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and the anchor cell. This structure must be initialized by [MorphGrayInit](#) beforehand.

---

**NOTE**

The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

---

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if <i>border</i> has an illegal value.

## GrayErodeBorder

---

*Performs gray-kernel erosion of an image.*

---

**Syntax**

```
IppStatusippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
IppStatusippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);
```

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>border</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and the anchor cell. This structure must be initialized by `MorphGrayInit` beforehand.

---

### NOTE

The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>border</i> has an illegal value.

---

## MorphAdvInit

*Initializes the specification structure for advanced morphological operations.*

---

### Syntax

```
IppStatusippiMorphAdvInit_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize  
maskSize, IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R				32f_C3R
8u_C4R				32f_C4R

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSpec</i>	Pointer to the specification structure.
<i>roiSize</i>	Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>pBuffer</i>	Pointer to the external buffer for advanced morphological operations.

## Description

This function operates with ROI.

This function initializes the specification structure *pSpec* in the external buffer. This structure is used by the [MorphOpenBorder](#) and [MorphCloseBorder](#) functions that perform open and close morphological operations.

All advanced morphological operations are performed on the source image pixels corresponding to non-zero values of the structuring element (mask) *pMask*.

---

### NOTE

This function required that the image ROI does not exceed the maximum width and height *roiSize* specified by the initialization functions.

---

For an example on how to use this function, see the code example provided with the [MorphCloseBorder](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.
<code>ippStsAnchorErr</code>	Indicates an error when <code>anchor</code> is outside the mask.

## See Also

[Regions of Interest in Intel IPP](#)

[MorphAdvGetSize](#) Computes the size of the specification structure for advanced morphological operations.

[MorphOpenBorder](#) Opens an image.

[MorphCloseBorder](#) Closes an image.

[MorphTophatBorder](#) Performs top-hat operation on an image.

[MorphBlackhatBorder](#) Performs black-hat operation on an image.

**MorphGradientBorder** Calculates morphological gradient of an image.

## MorphAdvGetSize

*Computes the size of the specification structure for advanced morphological operations.*

### Syntax

#### Case 2: Computing the size of morphology specification structure

```
IppStatusippiMorphAdvGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int*  
pSpecSize, int* pBufferSize);
```

Supported values for *mod*:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R				32f_C3R
8u_C4R				32f_C4R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.
<i>maskSize</i>	Size of the mask, in pixels.
<i>pSpecSize</i>	Pointer to the size of the morphology specification structure.
<i>pBufferSize</i>	Pointer to the size of the external buffer required for advanced morphological operations.

### Description

This function operates with ROI.

This function computes the size of the specification structure and the size of the buffer required for advanced morphological operations. Call this function before using the `ippiMorphAdvInit` function.

For an example on how to use this function, see the code example provided with the `ippiMorphCloseBorder` function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>maskSize</code> or <code>roiSize</code> has a field with a zero or negative value.

## See Also

[Regions of Interest in Intel IPP](#)

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

[MorphCloseBorder](#) Closes an image.

## MorphGetBufferSize

---

*Computes the size of the working buffer for advanced morphological operations.*

---

### Syntax

```
IppStatusippiMorphGetBufferSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType  
dataType, int numChannels, IppSizeL* pBufferSize);
```

### Include Files

ippcv\_1.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Maximum size of the image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>dataType</i>	Data type for the processing function.
<i>numChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the buffer size value for the initialization function.

### Description

This function computes the size of the working buffer required for advanced morphological operations.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

## MorphGetSpecSize

---

*Computes the size of the internal state or specification structure for advanced morphological operations.*

---

### Syntax

```
ippiMorphGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType dataType, int  
numChannels, IppSizeL* pSpecSize);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Maximum size of the image ROI, in pixels.
<i>maskSize</i>	Size of the structuring element.
<i>dataType</i>	Data type for the processing function.
<i>numChannels</i>	Number of channels in the image.
<i>pSpecSize</i>	Pointer to the size of the specification structure.

## Description

This function computes the size of the specification structure for advanced morphological operations.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

# MorphInit

*Initializes the internal state or specification structure for advanced morphological operations.*

## Syntax

```
IppStatusippiMorphInit_L( IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize,  
IppDataType dataType, int numChannels, IppiMorphAdvStateL* pMorphSpec);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Maximum size of the image ROI, in pixels.
<i>pMask</i>	Pointer to the structuring element (mask).
<i>maskSize</i>	Size of the structuring element.

<i>dataType</i>	Data type for the processing function.
<i>numChannels</i>	Number of channels in the image.
<i>pMorphSpec</i>	Pointer to the advanced morphology specification structure.

## Description

This function initializes the internal state or specification structure for advanced morphological operations.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.
ippStsAnchorErr	Anchor point is outside the structuring element.

## MorphologyBorderGetSize

*Computes the size of the morphology specification structure.*

---

### Syntax

```
IppStatusippiMorphologyBorderGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int*  
pSpecSize, int* pBufferSize);
```

Supported values for mod:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R				32f_C3R
8u_C4R				32f_C4R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the image ROI, in pixels.
<i>maskSize</i>	Size of the mask, in pixels.
<i>pSpecSize</i>	Pointer to the size of the morphology specification structure.
<i>pBufferSize</i>	Pointer to the size of the buffer required for dilation or erosion operations.

## Description

This function operates with ROI.

This function computes the size of the morphology specification structure *pMorphSpec* and the size of the buffer required for dilation and erosion operations. Call this function before using the *ippiMorphologyBorderInit* function.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error when <i>maskSize</i> has a field with a zero or negative value, or if width or height of <i>roiSize</i> is less than 1.

## See Also

[Regions of Interest in Intel IPP](#)

[MorphologyBorderInit](#) Initializes the morphology specification structure for erosion or dilation operations.

## MorphologyBorderInit

*Initializes the morphology specification structure for erosion or dilation operations.*

### Syntax

```
IppStatusippiMorphologyBorderInit_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R				32f_C3R
8u_C4R				32f_C4R

### Include Files

*ippcv.h*

### Domain Dependencies

Headers: *ippcore.h*, *ippvm.h*, *ipps.h*, *ippi.h*

Libraries: *ippcore.lib*, *ippvm.lib*, *ipps.lib*, *ippi.lib*

### Parameters

<i>roiSize</i>	Size of the image ROI, in pixels.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask, in pixels.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer required for dilation or erosion operations.

### Description

This function operates with ROI.

This function initializes the specification structure *pSpec* in the external buffer. Before using this function, you need to compute the size of the specification structure using the [MorphologyBorderGetSize](#) function. This structure is used by the [ippiDilateBorder](#) and [ippiErodeBorder](#) functions that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element (mask) *pMask*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>maskSize</i> has a field with a zero or negative value, or if width or height of <i>roiSize</i> is less than 1.

## See Also

[Regions of Interest in Intel IPP](#)

[MorphologyBorderGetSize](#) Computes the size of the morphology specification structure.

[DilateBorder](#) Performs dilation of an image.

[ErodeBorder](#) Performs erosion of an image.

## MorphBlackhat

---

*Performs top-hat operation on an image.*

---

### Syntax

```
IppStatusippiMorphBlackhat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u*  
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u  
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s*  
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s  
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int  
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,  
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*  
pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,  
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u  
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,  
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u  
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,  
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u  
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);  
  
IppStatusippiMorphBlackhat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*  
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f  
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```
IppStatusippiMorphBlackhat_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```
IppStatusippiMorphBlackhat_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image row.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
ippBorderDefault	The border is set to ippBorderConst with <i>borderValue</i> = MAX_VALUE, where MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.
ippBorderConst	Values of all border pixels are set to a constant.
ippBorderFirstStage InMem	You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the   operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the   operation.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the `ippBorderRepl`, `ippBorderConst`, `ippBorderDefault`, or `ippBorderMirror` values and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` modifiers.

`borderValue`, `borderValue[3]`,  
`borderValue[4]`

Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

`pMorphSpec`

Pointer to the morphology specification structure.

`pBuffer`

Pointer to the external buffer.

## Description

Before using this function, you need to initialize the morphology specification structure using the `ippiMorphInit` function.

This function performs a black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the subtraction of the initial source image from the closed source image.

---

### NOTE

The function can only process a ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <code>roiSize</code> has a field with a zero or negative value</li> <li>• ROI width is more than ROI width passed to the initialization function</li> </ul>
<code>ippStsStepErr</code>	Indicates an error condition when <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values is not a multiple of an element size.
<code>ippStsBadArgErr</code>	Indicates an error when <code>borderType</code> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

## MorphBlackhatBorder

*Performs black-hat operation on an image.*

### Syntax

```
IppStatusippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R 32f\_C3R

```
IppStatusippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R 32f\_C4R

```
IppStatusippiMorphBlackhatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for the 1u_C1R flavor).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.

<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for the <code>lu_C1R</code> flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
	<code>ippBorderRepl</code> Border is replicated from the edge pixels.
	<code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by [MorphAdvInit](#) beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.

### NOTE

The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## See Also

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

# MorphClose

*Closes an image.*

## Syntax

```
IppStatusippiMorphClose_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphClose_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image row.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:  ippBorderDefault      The border is set to <code>ippBorderConst</code> with <code>borderValue= MAX_VALUE</code> , where <code>MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U</code> ippBorderRepl          Border is replicated from the edge pixels. ippBorderMirror        Border pixels are mirrored from the source image boundary pixels. ippBorderConst         Values of all border pixels are set to a constant.  ippBorderFirstStageInMem      You can use this border type together with the <code>ippBorderRepl</code> , <code>ippBorderMirror</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> types using the <code> </code> operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the <code> </code> operation.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> modifiers.  Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  Pointer to the morphology specification structure. Pointer to the external buffer.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	
<i>pMorphSpec</i>	
<i>pBuffer</i>	

## Description

Before using this function, you need to initialize the morphology specification structure using the `ippiMorphInit` function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the *pMorphSpec* structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

**NOTE**

The function can only process a ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

**Return Values**

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <i>roiSize</i> has a field with a zero or negative value</li> <li>• ROI width is more than ROI width passed to the initialization function</li> </ul>
ippStsStepErr	Indicates an error condition when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when <i>borderType</i> has an illegal value.

**See Also**

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

## MorphCloseBorder

*Closes an image.*

**Syntax**

```
IppStatusippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R 32f\_C3R

```
IppStatusippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

**Supported values for mod:**

8u\_C4R      32f\_C4R

```
IppStatusippiMorphCloseBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for <i>1u_C1R</i> flavor).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for <i>1u_C1R</i> flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
	ippBorderRepl      Border is replicated from the edge pixels.
	ippBorderInMem      Border is obtained from the source image pixels in memory.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <i>ippBorderConst</i> border type.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer required for dilation operations.

## Description

Before using this function, you need to initialize the morphology specification structure using the *MorphAdvInit* function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the *pSpec* structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

### **NOTE**

This function requires that the image ROI does not exceed the maximum width and height *roiSize* specified by the initialization functions.

Usage example of this function is similar to the example provided with the [MorphOpenBorder](#) function description.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <i>roiSize</i> has a field with a zero or negative value</li> <li>• one of the ROI width or height is more than the corresponding size of ROI passed to the initialization functions</li> <li>• <i>srcBitOffset</i> or <i>dstBitOffset</i> is less than zero</li> </ul>
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values for 16-bit integer images is not divisible by 2.
ippStsBorderErr	Indicates an error condition if <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiMorphAdvGetSize_16u_C1R`, `ippiMorphAdvInit_16u_C1R`, and `ippiMorphCloseBorder_16u_C1R` functions.

```
IppStatus func_MorfCloseBorder()
{
    IppiMorphAdvState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
    Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2,
                         5, 1, 2, 1, 2,
                         1, 2, 1, 2, 1,
                         2, 1, 5, 1, 2};

    Ipp16u pDst[5*5];
    int srcStep = 5*sizeof(Ipp16u);
    int dstStep = 5*sizeof(Ipp16u);
    int dstSize = 5;
    IppStatus status = ippStsNoErr;
    int specSize = 0, bufferSize = 0;
    IppiBorderType borderType= ippBorderRepl;
```

```

Ipp16u borderValue = 0;

status =ippiMorphAdvGetSize_16u_C1R( roiSize, maskSize, &specSize, &bufferSize );
if (status != ippStsNoErr) return status;

pSpec = (IppiMorphAdvState*)ippsMalloc_8u(specSize);
pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);

status =ippiMorphAdvInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
if (status != ippStsNoErr) {
    ippsFree(pBuffer);
    ippsFree(pSpec);
    return status;
}

status =ippiMorphCloseBorder_16u_C1R (pSrc, srcStep, pDst, dstStep, roiSize, borderType,
borderValue, pSpec, pBuffer);

ippsFree(pBuffer);
ippsFree(pSpec);
return status;
}

```

The result is as follows:

```

pDst->
5 4 4 2 2
5 4 4 2 2
2 2 2 2 2
2 2 5 2 2
2 2 2 2 2

```

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

## MorphGradient

---

*Calculates morphological gradient of an image.*

### Syntax

```
IppStatusippiMorphGradient_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatusippiMorphGradient_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatusippiMorphGradient_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorphSpec, Ipp8u* pBuffer);
```

```

IppStatusippiMorphGradient_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphGradient_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphGradient_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphGradient_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphGradient_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphGradient_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image row.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
ippBorderDefault	The border is set to <code>ippBorderConst</code> with <code>borderValue= MAX_VALUE</code> , where <code>MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U</code>
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.

<code>ippBorderConst</code>	Values of all border pixels are set to a constant.
<code>ippBorderFirstStage</code> <code>InMem</code>	You can use this border type together with the <code>ippBorderRepl</code> , <code>ippBorderMirror</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> types using the <code> </code> operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the <code> </code> operation.
<code>borderValue</code> , <code>borderValue[3]</code> , <code>borderValue[4]</code>	Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> modifiers.
<code>pMorphSpec</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pBuffer</code>	Pointer to the morphology specification structure.
	Pointer to the external buffer.

## Description

Before using this function, you need to initialize the morphology specification structure using the `ippiMorphInit` function.

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell.

The result is equivalent to the subtraction of an opened source image from a closed source image.

---

### NOTE

The function can only process a ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <code>roiSize</code> has a field with a zero or negative value</li> <li>• ROI width is more than ROI width passed to the initialization function</li> </ul>
<code>ippStsStepErr</code>	Indicates an error condition when <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values is not a multiple of an element size.

`ippStsBadArgErr` Indicates an error when `borderType` has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

# MorphGradientBorder

*Calculates morphological gradient of an image.*

```
IppStatusippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C3R 32f\_C3R

```
IppStatusippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C4R 32f\_C4R

```
IppStatusippiMorphGradientBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; the possible value is <code>ippBorderRepl</code> , which means that a replicated border is used.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external work buffer.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by [MorphAdvInit](#) beforehand.

The result is equivalent to the subtraction of an opened source image from a closed source image.

### NOTE

The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## See Also

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

# MorphOpen

*Opens an image.*

## Syntax

```
IppStatusippiMorphOpen_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u*
pBuffer );

IppStatusippiMorphOpen_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatusippiMorphOpen_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatusippiMorphOpen_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );
```

## Include Files

ippcv\_1.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image row.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:  ippBorderDefault      The border is set to <code>ippBorderConst</code> with <code>borderValue= MAX_VALUE</code> , where <code>MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U</code> ippBorderRepl          Border is replicated from the edge pixels. ippBorderMirror        Border pixels are mirrored from the source image boundary pixels. ippBorderConst         Values of all border pixels are set to a constant.  ippBorderFirstStage    You can use this border type together with the <code>ippBorderRepl</code> , <code>ippBorderMirror</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> types using the <code> </code> operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the <code> </code> operation.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> modifiers.  <code>borderValue</code> , <code>borderValue[3]</code> , <code>borderValue[4]</code> Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  <code>pMorphSpec</code> Pointer to the morphology specification structure.  <code>pBuffer</code> Pointer to the external buffer.

## Description

Before using this function, you need to initialize the morphology specification structure using the `ippiMorphInit` function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the `pMorphSpec` structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

**NOTE**

The function can only process a ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

**Return Values**

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <i>roiSize</i> has a field with a zero or negative value</li> <li>• ROI width is more than ROI width passed to the initialization function</li> </ul>
ippStsStepErr	Indicates an error condition when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when <i>borderType</i> has an illegal value.

**See Also**

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

## MorphOpenBorder

*Opens an image.*

**Syntax**

```
IppStatusippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R 32f\_C3R

```
IppStatusippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

**Supported values for mod:**

8u\_C4R      32f\_C4R

```
IppStatusippiMorphOpenBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for the <i>1u_C1R</i> flavor).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for the <i>1u_C1R</i> flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
	ippBorderRepl      Border is replicated from the edge pixels.
	ippBorderInMem      Border is obtained from the source image pixels in memory.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <i>ippBorderConst</i> border type.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer required for dilation operations.

## Description

Before using this function, you need to initialize the morphology specification structure by using [MorphAdvInit](#) function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the *pSpec* structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

### NOTE

The function can only process a ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <i>roiSize</i> has a field with a zero or negative value</li> <li>• one of the ROI width or height is more than the corresponding size of ROI passed to the initialization functions</li> <li>• <i>srcBitOffset</i> or <i>dstBitOffset</i> is less than zero</li> </ul>
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values for 16-bit integer images is not divisible by 2.
ippStsBorderErr	Indicates an error condition if <i>borderType</i> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

## MorphTophat

*Performs top-hat operation on an image.*

### Syntax

```
IppStatusippiMorphTophat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```
IppStatusippiMorphTophat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatusippiMorphTophat_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image row.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:
ippBorderDefault	The border is set to ippBorderConst with borderValue= MAX_VALUE, where MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.
ippBorderConst	Values of all border pixels are set to a constant.

<code>ippBorderFirstStage</code>	You can use this border type together with the <code>ippBorderRepl</code> , <code>ippBorderMirror</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> types using the <code> </code> operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the <code> </code> operation.
<code>InMem</code>	
<code>Mixed</code>	Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderDefault</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> modifiers.
<code>borderValue</code> , <code>borderValue[3]</code> , <code>borderValue[4]</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pMorphSpec</code>	Pointer to the morphology specification structure.
<code>pBuffer</code>	Pointer to the external buffer.

## Description

Before using this function, you need to initialize the morphology specification structure using the [ippiMorphInit](#) function.

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

### NOTE

The function can only process a ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition when: <ul style="list-style-type: none"> <li>• <code>roiSize</code> has a field with a zero or negative value</li> <li>• ROI width is more than ROI width passed to the initialization function</li> </ul>
<code>ippStsStepErr</code>	Indicates an error condition when <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values is not a multiple of an element size.
<code>ippStsBadArgErr</code>	Indicates an error when <code>borderType</code> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

## MorphTophatBorder

---

*Performs top-hat operation on an image.*

---

### Syntax

#### Case 1: Operating with morphology state structure

```
IppStatusippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R      32f\_C1R

```
IppStatusippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R      32f\_C3R

```
IppStatusippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R      32f\_C4R

#### Case 2: Operating with morphology specification structure

```
IppStatusippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

16u\_C1R      16s\_C1R

```
IppStatusippiMorphTophatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset, in bits, from the first byte of the source image (for the <code>lu_C1R</code> flavor).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset, in bits, from the first byte of the destination image (for the <code>lu_C1R</code> flavor).
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border. Possible values are:  ippBorderRepl      Border is replicated from the edge pixels. ippBorderInMem      Border is obtained from the source image pixels in memory.
<i>borderValue</i> , <i>borderValue[3]</i> , <i>borderValue[4]</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell. The structure must be initialized by [MorphAdvInit](#) beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

---

### NOTE

The function can process only images with ROI that does not exceed the maximum width and height *roiSize* specified by the initialization functions.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if <i>borderType</i> has an illegal value.

## See Also

[MorphAdvInit](#) Initializes the specification structure for advanced morphological operations.

## MorphGrayInit

---

*Initializes morphology state structure for gray-kernel morphology operations.*

### Syntax

```
IppStatusippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState, IppiSize roiSize,
const Ipp32s* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatusippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState, IppiSize roiSize,
const Ipp32f* pMask, IppiSize maskSize, IppiPoint anchor);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pState</i>	Pointer to the gray-kernel morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the gray-kernel morphology state structure *pState* in the external buffer. Its size should be computed by the function [MorphGrayGetSize](#). It is used by the functions [GrayDilateBorder](#) and [GrayErodeBorder](#) that perform gray-kernel dilation and erosion of the source image pixels corresponding to the specified *pMask* of size *maskSize*. The anchor cell *anchor* is positioned in the arbitrary point in the mask and is used for positioning the mask.

**WARNING**

The structure can be used to process images with ROI that does not exceed the specified maximum width and height *roiSize*.

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.
ippStsAnchorErr	Indicates an error if <i>anchor</i> is outside the mask.

## MorphGrayGetSize

*Computes the size of the gray-kernel morphology state structure.*

**Syntax**

```
IppStatusippiMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, int* pSize);  
IppStatusippiMorphGrayGetSize_32f_C1R(IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, int* pSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>pSize</i>	Pointer to the size of the advanced morphology state structure.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the morphology state structure *pState* for gray-kernel dilation and erosion. This function should be run prior to the function [MorphGrayInit](#).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with a zero or negative value.

# MorphReconstructGetSize

*Computes the size of the buffer for morphological reconstruction operation.*

```
IppStatusippiMorphReconstructGetBufferSize(IppiSize roiSize, IppDataType dataType, int numChannels, int* pBufSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the buffer.
<i>dataType</i>	Data type of the image.
<i>numChannels</i>	Number of channels in the image.
<i>pBufSize</i>	Pointer to the size of the work buffer (in bytes), returned by the <i>ippiMorphReconstructGetBufferSize</i> function.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the buffer for the morphological reconstruction of the source image. This buffer can be used by the functions [MorphReconstructDilate](#) and [MorphReconstructErode](#).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSize</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

# MorphReconstructDilate

*Reconstructs an image by dilation.*

## Syntax

```
IppStatusippiMorphReconstructDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer, IppiNorm
norm);
```

Supported values for `mod`:

8u\_C1IR 16u\_C1IR 64f\_C1IR

```
IppStatusippiMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuffer, IppiNorm norm);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pSrcDst</code>	Pointer to the decreased and reconstructed image ROI.
<code>srcDstStep</code>	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.
<code>roiSize</code>	Size of the source and destination image ROI.
<code>norm</code>	Type of norm to form the mask for dilation; the following values are possible:
	ippiNormInf                    Infinity norm (8-connectivity, 3x3 rectangular mask).
	ippiNormL1                    L1 norm (4-connectivity, 3x3 cross mask).
<code>pBuffer</code>	Pointer to the buffer.

## Description

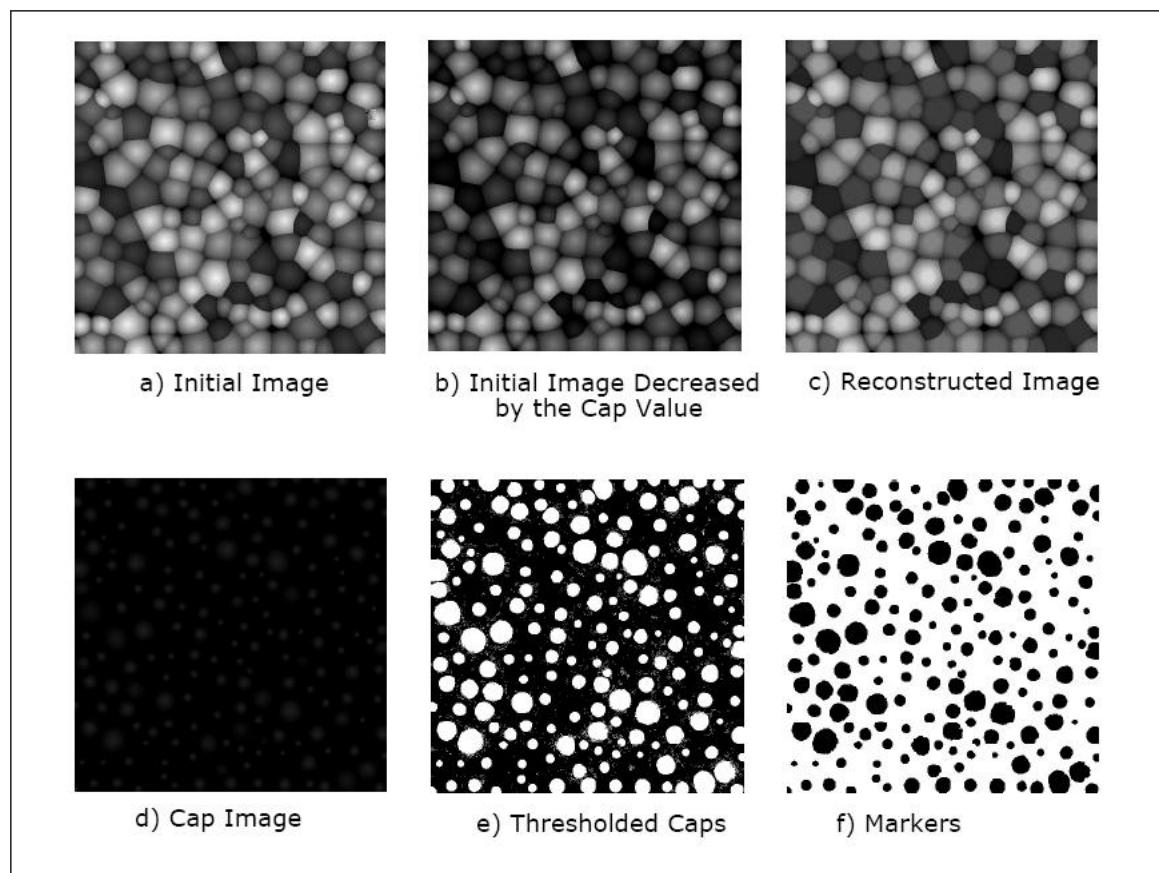
This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs morphological reconstruction of the decreased source image by dilation [[Vincent93](#)]. The operation is performed in the working buffer whose size should be computed using the function [MorphReconstructGetBufferSize](#) beforehand.

This operation enables detection of the regional maximums that can be used as markers for successive watershed segmentation.

**Example** below shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. The figure below shows the results of these operations.

### Building Markers for Segmentation by the Morphological Reconstruction



### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>srcDstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if <i>norm</i> has an illegal value.

### Example

To better understand usage of this function, refer to the `MorphReconstructDilate.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

# MorphReconstructErode

*Reconstructs an image by erosion.*

## Syntax

```
IppStatusippiMorphReconstructErode_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuf, IppiNorm norm);
```

Supported values for mod:

8u\_C1IR 16u\_C1IR 64f\_C1IR

```
IppStatusippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuf, IppiNorm norm);
```

## Include Files

ippcv.h.

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pSrcDst</i>	Pointer to the decreased and reconstructed image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>norm</i>	Type of norm to form the mask for dilation; the following values are possible:  ippiNormInf                  Infinity norm (8-connectivity, 3x3 rectangular mask). ippiNormL1                  L1 norm (4-connectivity, 3x3 cross mask).
<i>pBuf</i>	Pointer to the buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs morphological reconstruction of the increased source image by erosion [[Vincent93](#)]. The operation is performed in the working buffer whose size should be computed using the function [MorphReconstructGetBufferSize](#) beforehand.

This operation enables detection of the regional minimums that can be used as markers for successive watershed segmentation.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>srcDstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if <i>norm</i> has an illegal value.

## MorphSetMode

---

Sets the mask processing mode for advanced morphological operations.

---

### Syntax

```
IppStatusippiMorphSetMode(int mode, IppiMorphAdvState* pMorphSpec);
IppStatusippiMorphSetMode_L(int mode, IppiMorphAdvStateL* pMorphSpec);
```

### Include Files

ippcv.h  
ippcv\_1.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>mode</i>	Mask processing <i>mode</i> ; supported values: IPP_MORPH_DEFAULT Invert the mask IPP_MORPH_MASK_NO_FLIP Do not invert the mask; use the same mask for the first and second stage.
<i>pMorphSpec</i>	Pointer to the specification structure for advanced morphological operations.

### Description

This function sets the mask processing *mode* for the second stage of an advanced morphological operation. Before using this function, initialize the specification structure using the *ippiMorphInit* function.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNotSupportedModeErr	Indicates an error when <i>mode</i> has an invalid value.
ippStsNullPtrErr	Indicates an error when <i>pMorphSpec</i> is NULL.

**See Also**

[MorphInit](#) Initializes the internal state or specification structure for advanced morphological operations.

# Filtering Functions

This section describes the Intel® IPP image processing functions that perform linear and non-linear filtering operations on an image.

You can use filtering in image processing operations like edge detection, blurring, noise removal, and feature detection.

Most filtering functions operate with regions of interest (ROI). The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. Most functions use different source and destination image buffers. These functions are not in-place.

## See Also

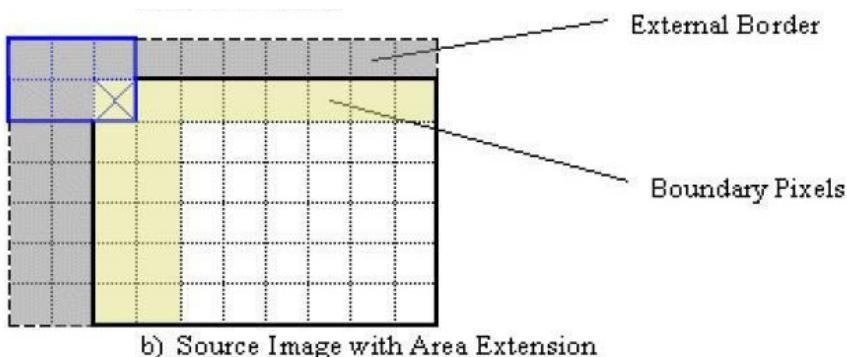
[Regions of Interest in Intel IPP](#)

## Borders in Neighborhood Operations

Filtering functions described in this section perform neighborhood operations. They operate on the assumption that for each pixel to be processed, all neighborhood pixels required for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. For more information about anchors and how to define the anchor cell position refer to [Neighborhood Operations](#).

As the following figure illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image and are located outside the image area.



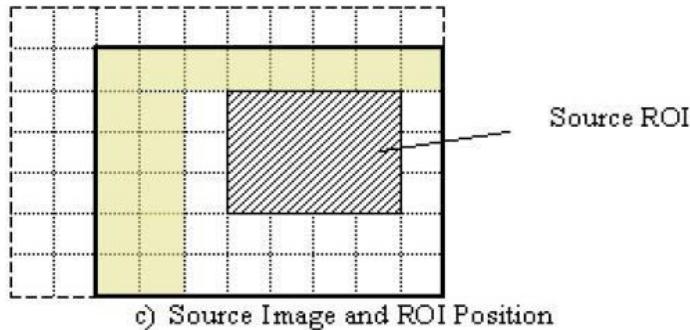
The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow, while the collection of all scanned external pixels (called *border* pixels) is shaded gray.

If you want to apply some filtering operation to the whole source image, you must figure out what additional border pixels are required for the operation, and define these non-existent pixels. To do this, you can use the Intel IPP functions `ippiCopyConstBorder`, `ippiCopyReplicateBorder`, or `ippiCopyWrapBorder`, which fill the border of the extended image with the pixel values that you define, or you can apply your own extension method.

### Note

If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.

If you want to apply the filtering operation to the part of the source image, or ROI, then the necessity of extending the image area with border pixels depends on the ROI size and position within the image. The figure below shows that if ROI does not cover yellow (internal boundary) pixels, then no external pixels are scanned, and border extension is not required.



If boundary pixels are part of ROI, you still need to extend some area of the source image.

To provide valid results of filtering operations, the application must check the following:

- ROI parameters passed to the filtering function have such values
- All required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

## See Also

[Neighborhood Operations](#)  
[Regions of Interest in Intel IPP](#)  
[User-defined Border Types](#)  
[CopyConstBorder](#)  
[CopyReplicateBorder](#)  
[CopyWrapBorder](#)

## User-defined Border Types

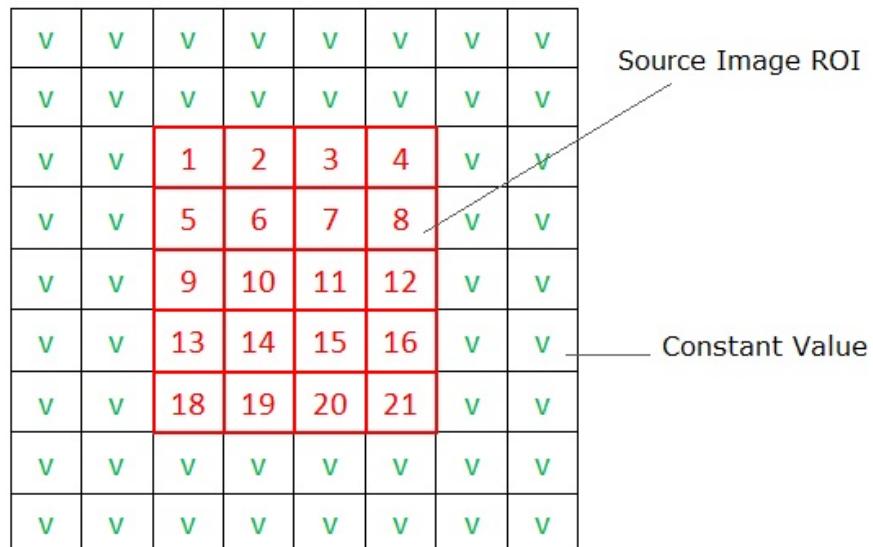
Some of the Intel® IPP image processing functions operate on user-defined border types. It means that the values of border pixels are assigned in accordance with the *borderType* (or *border*) and *borderValue* parameters.

Intel® IPP supports the following border types:

- Constant border
- Replicated border
- Mirrored border
- Mirrored border with replication
- Border in memory
- Mixed borders

### Constant Border

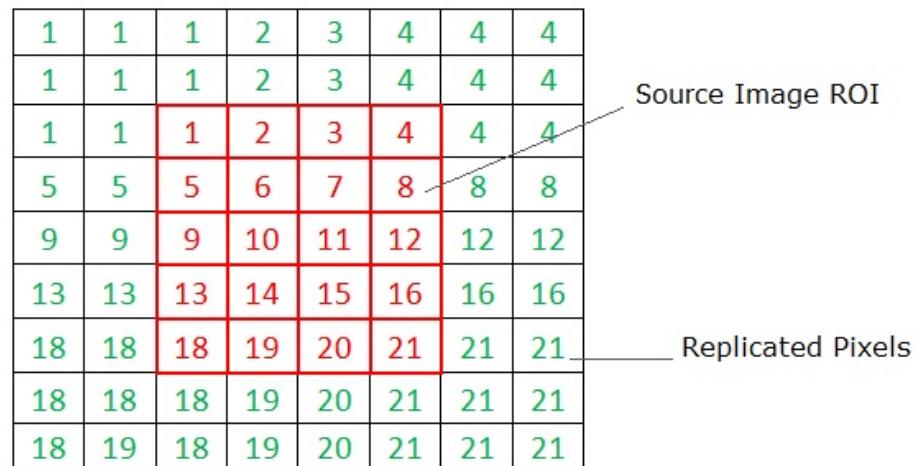
This type of border corresponds to the *ippBorderConst* value in the *IppiBorderType* enumerator. When using a constant border, values for all border pixels are set to the constant value that you specify in the *borderValue* parameter. In the figure below, this constant value is marked as v. Squares marked in red correspond to pixels copied from the source image ROI.



V	V	V	V	V	V	V	V
V	V	V	V	V	V	V	V
V	V	1	2	3	4	V	V
V	V	5	6	7	8	V	V
V	V	9	10	11	12	V	V
V	V	13	14	15	16	V	V
V	V	18	19	20	21	V	V
V	V	V	V	V	V	V	V
V	V	V	V	V	V	V	V

## Replicated Border

This type of border corresponds to the `ippBorderRep1` value in the `IppiBorderType` enumerator. When using a replicated border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are replicated from the boundary pixels of the source image.



1	1	1	2	3	4	4	4
1	1	1	2	3	4	4	4
1	1	1	2	3	4	4	4
5	5	5	6	7	8	8	8
9	9	9	10	11	12	12	12
13	13	13	14	15	16	16	16
18	18	18	19	20	21	21	21
18	18	18	19	20	21	21	21
18	19	18	19	20	21	21	21

## Mirrored Border

This type of border corresponds to the `ippBorderMirror` value in the `IppiBorderType` enumerator. When using a mirrored border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels.

11	10	9	10	11	12	11	10
7	6	5	6	7	8	7	6
3	2	1	2	3	4	3	2
7	6	5	6	7	8	7	6
11	10	9	10	11	12	11	10
15	14	13	14	15	16	15	14
20	19	18	19	20	21	20	19
15	14	13	14	15	16	15	14
11	10	9	10	11	12	11	10

### Mirrored Border with Replication

This type of border corresponds to the `ippBorderMirrorR` value in the `IppiBorderType` enumerator. When using a mirrored border with replication, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels. The difference of this border type from the mirrored border is that the anchor cell value is replicated to the border pixels.

6	5	5	6	7	8	8	7
2	1	1	2	3	4	4	3
2	1	1	2	3	4	4	3
6	5	5	6	7	8	8	7
10	9	9	10	11	12	12	11
14	13	13	14	15	16	16	15
19	18	18	19	20	21	21	20
19	18	18	19	20	21	21	20
14	13	13	14	15	16	16	15

### Border in Memory

This type of border corresponds to the `ippBorderInMem` value and its flags combinations in the `IppiBorderType` enumerator. Use this border type if the ROI does not cover internal border pixels of the source image. In this case, values for border pixels are obtained from the source image pixels in memory. In the figure below, squares marked in red correspond to pixels copied from the source image ROI. Squares with black values correspond to source image pixels in memory.

40	41	42	43	44	45	46	47
55	54	53	52	51	50	49	48
56	57	1	2	3	4	58	59
63	62	5	6	7	8	61	60
64	65	9	10	11	12	66	67
71	70	13	14	15	16	69	68
72	73	18	19	20	21	74	75
83	82	81	80	79	78	77	76
84	85	86	87	88	89	90	91

Several Intel IPP filters operate in two or more stages. For example, the `ippiMorphOpenBorder` function performs filtering by applying the `Erode` and `Dilate` filters sequentially. You should note the following when setting borders for multistage filters:

- If you set the `ippBorderInMem` value or its flags combinations, the function tries to access pixels outside of image borders to get border pixels for each filtering stage. For example, the `ippiMorphOpenBorder` function uses two stages and with 5x5 mask will access  $\text{floor}(5/2) * 2 = 4$  pixels in each direction across the current ROI.
- If you set `ippBorderFirstStageInMem`, the function tries to access  $\text{floor}(5/2) = 2$  pixels outside of the image borders to get pixels for the first stage of filtering. The second filter will use one of the following border types to reconstruct image borders: `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR`. To specify the border type for the second and next stages, use the bitwise OR operation between one of the listed above border types and `ippBorderFirstStageInMem`.

## Mixed Borders

You can use mixed borders by using a bitwise OR operation between one of the `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR` types and any of the following border types: `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight`, or `ippBorderFirstStageInMem`. In this case, values for border pixels are obtained from the source image pixels in memory in the direction specified by the flag.

The figure below demonstrates the use of the `ippBorderConst` with the `ippBorderInMemTop` and `ippBorderInMemRight` borders. Squares marked in red correspond to pixels copied from the source image, that is the source image ROI. As you can see from the figure, top and right border pixels are obtained from the source image pixels in memory, while the rest of the border pixels are set to the constant value V.

V	V	42	43	44	45	46	47
V	V	53	52	51	50	49	48
V	V	1	2	3	4	58	59
V	V	5	6	7	8	61	60
V	V	9	10	11	12	66	67
V	V	13	14	15	16	69	68
V	V	18	19	20	21	74	75
V	V	V	V	V	V	V	V
V	V	V	V	V	V	V	V

**NOTE**

The combination of `ippBorderInMem` and its flags always has priority over any other border flags or types. For example, if you specify `ippBorderFirstStageInMem | ippBorderRep1 | ippBorderInMemLeft`, the left border will use `InMem` mode for each stage and other borders will use `InMem` for the first stage and replication for remaining stages.

The figure below demonstrates the use of the `ippBorderConst` with the `ippBorderInMemTop`, `ippBorderInMemRight`, and `ippBorderFirstStageInMem` flags for two-stage filtering with one pixel border for both stages.

- First stage:** squares marked in red correspond to pixels copied from the source image, which is the source image ROI, and squares marked in blue correspond to ROI assigned to the first stage filter. As you can see from the figure, the first stage enlarges ROI for top and right sides to consume more memory and provide valid pixels for the second stage memory border.
- Second stage:** red squares and blue pixels correspond to resulting pixels from the first stage filter. Blue pixels lie outside of the ROI providing border values for the second stage in top and right directions. Left and bottom border pixels use the constant value `v` in accordance with the border flags combination.

**First stage:**

46	42	43	44	45	46	47
54	53	52	51	50	49	48
61	1	2	3	4	58	59
61	5	6	7	8	61	60
66	9	10	11	12	66	67
69	13	14	15	16	69	68
74	18	19	20	21	74	75
54	53	52	51	50	49	74

Source Image ROI  
First stage ROI  
Source Image  
Pixels in Memory  
for both stages  
Source Image  
Pixels in Memory  
for first stage only

**Second stage:**

v	37	25	70	21	22
v	1	2	3	4	16
v	5	6	7	8	73
v	9	10	11	12	58
v	13	14	15	16	36
v	18	19	20	21	14
v	v	v	v	v	v

Source Image ROI  
Pixels in Memory  
from the first stage  
Constant Value

## See Also

[Regions of Interest in Intel IPP](#)  
[Borders in Neighborhood Operations](#)

# Filters with Borders

---

This section describes Intel® IPP filtering functions that automatically create a required border and define appropriate pixel values.

## See Also

[Regions of Interest in Intel IPP](#)

### FilterBilateral

*Performs bilateral filtering of an image.*

---

#### Syntax

##### Case 1: Operation on pixel-order data

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u_C1R	32f_C1R	64f_C1R
--------	---------	---------

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u_C3R	32f_C3R	64f_C3R
--------	---------	---------

##### Case 2: Operation on planar data

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3],
Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_P3R	32f_P3R	64f_P3R
--------	---------	---------

### Case 3: Operation on pixel-order data with platform-aware functions

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R_L	32f_C1R_L	64f_C1R_L
----------	-----------	-----------

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R_L	32f_C3R_L	64f_C3R_L
----------	-----------	-----------

### Case 4: Operation on planar data with platform-aware functions

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL
srcStep[3], Ipp<dstdatatype>* pDst[3], IppSizeL dstStep[3], IppiSizeL dstRoiSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const
IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_P3R_L	32f_P3R_L	64f_P3R_L
----------	-----------	-----------

### Case 5: Operation on pixel-order data with Threading Layer (TL) functions based on the Platform Aware API

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec_LT*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R_LT	32f_C1R_LT	64f_C1R_LT
-----------	------------	------------

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_LT*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R_LT	32f_C3R_LT	64f_C3R_LT
-----------	------------	------------

**Case 6: Operation on planar data with Threading Layer (TL) functions based on the Platform Aware API**

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL
srcStep[3], Ipp<dstdatatype>* pDst[3], IppSizeL dstStep[3], IppiSizeL dstRoiSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const
IppiFilterBilateralSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_P3R\_LT

32f\_P3R\_LT

64f\_P3R\_LT

**Case 7: Operation on pixel-order data with Threading Layer (TL) functions based on the Classic API**

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec_T* pSpec, Ipp8u*
pBuffer);
```

Supported values for mod:

8u\_C1R\_T

32f\_C1R\_T

64f\_C1R\_T

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_T* pSpec, Ipp8u*
pBuffer);
```

Supported values for mod:

8u\_C3R\_T

32f\_C3R\_T

64f\_C3R\_T

**Case 8: Operation on planar data with Threading Layer (TL) functions based on the Classic API**

```
IppStatusippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3],
Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_T*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_P3R\_T

32f\_P3R\_T

64f\_P3R\_T

**Include Files**

ippi.h

ippi\_l.h

ippi\_tl.h

**Domain Dependencies**

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderInMem    Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
<i>pBorderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
<i>pSpec</i>	Pointer to the bilateral context structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function applies the bilateral filter with a square kernel to the source image. The linear dimension of the kernel is defined in the initialization function [FilterBilateralInit](#). The bilateral context structure contains the parameters of filtering.

Before using the `ippiFilterBilateral` function, compute the size of the bilateral context structure and the external buffer using the [FilterBilateralGetBufferSize](#) function and initialize the structure using the [FilterBilateralInit](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> , <i>pDst</i> , <i>pSpec</i> , or <i>pBuffer</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error when the <i>pSpec</i> structure does not match the function.

ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterBilateralGetBufferSize](#) Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

[FilterBilateralInit](#) Initializes the bilateral context structure.

## FilterBilateralGetBufferSize

*Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.*

### Syntax

```
IppStatusippiFilterBilateralGetBufferSize(IppiFilterBilateralType filter, IppiSize  
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

### Platform-aware function

```
IppStatusippiFilterBilateralGetBufferSize_L(IppiFilterBilateralType filter, IppiSizeL  
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

### Threading Layer (TL) function based on the Platform Aware API

```
IppStatusippiFilterBilateralGetBufferSize_LT(IppiFilterBilateralType filter, IppiSizeL  
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

### Threading Layer (TL) function based on the Classic API

```
IppStatusippiFilterBilateralGetBufferSize_T(IppiFilterBilateralType filter, IppiSize  
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

### Include Files

ippi.h  
ippi\_1.h  
ippi\_tl.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>filter</i>	Type of the bilateral filter. Possible value is ippiFilterBilateralGauss - Gaussian bilateral filter.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.

<i>kernelWidthHeight</i>	Linear dimension of the square kernel. The value 1 corresponds to the distance between two adjacent pixels.
<i>dataType</i>	Data type of the source and destination images. Possible values are <code>ipp8u</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<i>numChannels</i>	Number of channels in the images.
<i>distMethodType</i>	Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible value are:
	<i>numChannels</i> value      Possible <i>distMethodType</i> values
1	ippDistNormL1
3	ippDistNormL1, ippDistNormL2
	.
<i>pSpecSize</i>	Pointer to the computed size of the specification structure.
<i>pBufferSize</i>	Pointer to the computed size of the external buffer.

## Description

This function computes the size of the bilateral context structure and external work buffer for the `FilterBilateral` function. The results are stored in *pSpecSize* and *pBufferSize*.

Use the computed *pBufferSize* and *pSpecSize* values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSpecSize</i> or <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>kernelWidthHeight</i> is less than, or equal to zero.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the <i>filter</i> or <i>distMethodType</i> value is not supported.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

`FILterBilateral` Performs bilateral filtering of an image.

`FilterBilateralInit` Initializes the bilateral context structure.

## FilterBilaterallInit

*Initializes the bilateral context structure.*

## Syntax

```
IppStatusippiFilterBilateralInit(IppiFilterBilateralType filter, IppiSize dstRoiSize,
int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec*
pSpec);
```

### Platform-aware function

```
IppStatusippiFilterBilateralInit_L(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec* pSpec);
```

### Threading Layer (TL) function based on the Platform Aware API

```
IppStatusippiFilterBilateralInit_LT(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec_LT* pSpec);
```

### Threading Layer (TL) function based on the Classic API

```
IppStatusippiFilterBilateralInit_T(IppiFilterBilateralType filter, IppiSize
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec_T* pSpec);
```

## Include Files

ippi.h  
ippi\_l.h  
ippi\_tl.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>filter</i>	Type of the bilateral filter. Possible value is ippiFilterBilateralGauss - Gaussian bilateral filter.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>kernelWidthHeight</i>	Linear dimension of the square kernel. The value 1 corresponds to the distance between two adjacent pixels.
<i>dataType</i>	Data type of the source and destination images. Possible values are ipp8u, ipp32f, and ipp64f.
<i>numChannels</i>	Number of channels in the images.
<i>distMethod</i>	Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible value are:  <i>numChannels</i> value      Possible <i>distMethod</i> values

	1	ippDistNormL1
	3	ippDistNormL1, ippDistNormL2
valSquareSigma		Square of the range parameter, which controls smoothing based on the differences in intensity between pixels.
posSquareSigma		Square of the spatial parameter, which controls smoothing based on the geometric distance between pixels.
pSpec		Pointer to the bilateral context structure.

## Description

This function initializes the bilateral context structure *pSpecSize* for bilateral filtering. Before using this function, compute the size of the context structure using the [FilterBilateralGetBufferSize](#) function.

The *kernelWidthHeight* parameter specifies the linear dimension of the square filter kernel. The value 1 corresponds to the distance between the centers of two adjacent pixels.

Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel *d* is computed by the following formula:

$$d = \frac{\sum_{i,j} w1_{ij} * w2_{ij} * v_{ij}}{\sum_{i,j} w1_{ij} * w2_{ij}}$$

For all indices *i* and *j* that fit within the square kernel

where

- $v_{ij}$  is the value (or channel values) of a pixel in the kernel with coordinates *i* and *j*
- $w1_{ij} = \text{Fun}(\text{valSquareSigma}, \text{Intensity Distance}(v_{ij}, v_{00}))$

The *distMethodType* parameter specifies the method of defining the differences in intensity between pixels. FilterBilateral functions support two methods: *ippDistNormL1*, which defines the difference in intensity as the L1-norm, and *ippDistNormL2*, which defines the difference in intensity as the L2-norm.

- $w2_{ij} = \text{Fun}(\text{posSquareSigma}, \text{Geometric Distance}(v_{ij}, v_{00}) = \sqrt{i^2 + j^2})$

$\text{Fun}(S, I) = \exp(-I^2 / 2S)$

where

- *S* is *valSquareSigma* or *posSquareSigma*
- *I* is the difference between pixel values or position

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when <i>pSpec</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.

ippStsMaskSizeErr	Indicates an error when <i>kernelWidthHeight</i> is less than, or equal to zero.
ippStsNotSupportedModeErr	Indicates an error when the <i>filter</i> or <i>distMethod</i> value is not supported.
ippStsBadArgErr	Indicates an error when <i>valSquareSigma</i> or <i>posSquareSigma</i> is less than, or equal to zero.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[Structures and Enumerators for Platform-Aware Functions](#)

[FilterBilateralGetBufferSize](#) Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

[FILterBilateral](#) Performs bilateral filtering of an image.

## FilterBilateralBorderGetBufferSize

*Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.*

### Syntax

#### Processing images of 32-bit sizes

```
IppStatusippiFilterBilateralBorderGetBufferSize(IppiFilterBilateralType filter,  
IppiSize dstRoiSize, int radius, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

#### Platform-aware function

```
IppStatusippiFilterBilateralBorderGetBufferSize_L(IppiFilterBilateralType filter,  
IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

#### Threading layer function

```
IppStatusippiFilterBilateralBorderGetBufferSize_LT(IppiFilterBilateralType filter,  
IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels,  
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_t1.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_t1.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_t1.lib, ippi\_t1.lib

## Parameters

<i>filter</i>	Type of the bilateral filter. Possible value is <code>ippiFilterBilateralGauss</code> - Gaussian bilateral filter.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>radius</i>	Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.
<i>dataType</i>	Data type of the source and destination images. Possible values are <code>ipp8u</code> and <code>ipp32f</code> .
<i>numChannels</i>	Number of channels in the images.
<i>distMethodType</i>	Method of defining intensive distance between pixels. Possible value is <code>ippDistNormL1</code> .
<i>pSpecSize</i>	Pointer to the computed size of the specification structure.
<i>pBufferSize</i>	Pointer to the computed size of the external buffer.

## Description

This function computes the size of the bilateral context structure and external work buffer for the [FilterBilateralBorder](#) function. The results are stored in *pSpecSize* and *pBufferSize*.

Use the computed *pBufferSize* and *pSpecSize* values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the [FilterBilateralBorder](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSpecSize</i> or <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>radius</i> is less than, or equal to zero.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the <i>filter</i> or <i>distMethodType</i> value is not supported.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterBilateralBorder](#) Performs bilateral filtering of an image.

## FilterBilateralBorderInit

*Initializes the bilateral context structure.*

## Syntax

```
IppStatusippiFilterBilateralBorderInit(IppiFilterBilateralType filter, IppiSize
dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec*
pSpec);
```

### Platform-aware function

```
IppStatusippiFilterBilateralBorderInit_L(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec*
pSpec);
```

### Threading layer function

```
IppStatusippiFilterBilateralBorderInit_LT(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec_LT*
pSpec);
```

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_t1.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_t1.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_t1.lib`, `ippi_t1.lib`

## Parameters

<i>filter</i>	Type of the bilateral filter. Possible value is <code>ippiFilterBilateralGauss</code> - Gaussian bilateral filter.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>radius</i>	Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.
<i>dataType</i>	Data type of the source and destination images. Possible values are <code>ipp8u</code> and <code>ipp32f</code> .
<i>numChannels</i>	Number of channels in the images.
<i>distMethod</i>	Method of defining intensive distance between pixels. Possible value is <code>ippDistNormL1</code> .
<i>valSquareSigma</i>	Square of the sigma for intensive distance between pixels.
<i>posSquareSigma</i>	Square of the sigma for geometric distance between pixels.
<i>pSpec</i>	Pointer to the bilateral context structure.

## Description

This function initializes the bilateral context structure *pSpecSize* for bilateral filtering. Before using this function, compute the size of the context structure using the [FilterBilateralBorderGetBufferSize](#) function.

The *radius* parameter specifies the radius of the round filter kernel. The radius value equal to 1 corresponds to the distance between centers of the closest pixels.

Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel *d* is computed by the following formula:

$$d = \frac{\sum_{i,j} w1_{ij} * w2_{ij} * v_{ij}}{\sum_{i,j} w1_{ij} * w2_{ij}}$$

For all *i* and *j* that  $i^2 + j^2 \leq radius^2$  (for central pixel of the kernel  $i=0, j=0$ )

where

- $v_{ij}$  is the value (or channel values) of a pixel in the kernel with coordinates *i* and *j*
- $w1_{ij} = \text{Fun}(\text{valSquareSigma}, \text{Intensity Distance}(v_{ij}, v_{00}))$

The *distMethodType* parameter specifies the method of defining the intensive distance between pixels. Currently the only supported method is `ippDistNormL1`, which defines the intensive distance as norma L1.

- $w2_{ij} = \text{Fun}(\text{posSquareSigma}, \text{Geometric Distance}(v_{ij}, v_{00}) = \sqrt{i^2 + j^2})$

$\text{Fun}(S, I) = \exp(-I^2 / 2S)$

where

- *S* is *valSquareSigma* or *posSquareSigma*
- *I* is the difference between pixel values or position

For an example on how to use this function, refer to the example provided with the [FilterBilateralBorder](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSpec</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>radius</i> is less than, or equal to zero.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the <i>filter</i> or <i>distMethod</i> value is not supported.
<code>ippStsBadArgErr</code>	Indicates an error when <i>valSquareSigma</i> or <i>posSquareSigma</i> is less than, or equal to zero.

`ippStsNumChannelsErr` Indicates an error when `numChannels` has an illegal value.

## See Also

[FilterBilateralBorderGetBufferSize](#) Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

[FilterBilateralBorder](#) Performs bilateral filtering of an image.

## FilterBilateralBorder

*Performs bilateral filtering of an image.*

---

### Syntax

#### Processing images of 32-bit sizes

```
IppStatusippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
Ipp<datatype>* pBorderValue, IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C1R                  8u\_C3R

32f\_C1R

32f\_C3R

#### Platform-aware functions

```
IppStatusippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL
srcStep, Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, Ipp<datatype>* pBorderValue, const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u\_C1R\_L

8u\_C3R\_L

#### Threading layer functions

```
IppStatusippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL
srcStep, Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSize dstRoiSize, IppiBorderType
borderType, Ipp<datatype>* pBorderValue, IppiFilterBilateralSpec_LT* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u\_C1R\_L

8u\_C3R\_L

### Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_t1.h`

Flavors with the `_L` suffix: `ippi_l.h`

### Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_t1.h`:

**Libraries:**ippcore.lib,ippvm.lib,ipps.lib,ippi.lib,ippcore\_tl.lib,ippi\_tl.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderInMem    Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
<i>pBorderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
<i>pSpec</i>	Pointer to the bilateral context structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function applies the bilateral filter with the round kernel to the source image. The radius of the kernel is defined in the corresponding initialization function [FilterBilateralBorderInit](#). The bilateral context structure contains the parameters of filtering.

Before using the `ippiFilterBilateralBorder` function, compute the size of the bilateral context structure and the external buffer using the [FilterBilateralBorderGetBufferSize](#) function and initialize the structure using the [FilterBilateralBorderInit](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> , <i>pDst</i> , <i>pSpec</i> , or <i>pBuffer</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.

ippStsContextMatchErr	Indicates an error when the <i>pSpec</i> structure does not match the function.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilterBilateralBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterBilateralBorderGetBufferSize](#) Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

[FilterBilateralBorderInit](#) Initializes the bilateral context structure.

## FilterBoxBorderGetBufferSize

*Computes the size of the external buffer for the FilterBoxBorder function.*

---

## Syntax

```
IppStatusippiFilterBoxBorderGetBufferSize(IppiSize roiSize, IppiSize maskSize,  
IppDataType dataType, int numChannels, int* pBufferSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>roiSize</i>	Maximum size of the destination image ROI.
<i>maskSize</i>	Size of the filter mask, in pixels.
<i>dataType</i>	Data type of the image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , or <code>ipp32f</code> .
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterBoxBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterBoxBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterBoxBorder` function description.

## Return Values

ippStsNoErr	Indicates no error.
ippStsSizeErr	Indicates an error when <i>roiSize</i> is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsNumChannelsError	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterBoxBorder](#) Blurs an image using a simple box filter.

## FilterBoxBorder

*Blurs an image using a simple box filter.*

### Syntax

```
IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype>* borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

```
IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype> borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
16u_C4R	16s_C4R		
16u_AC4R	16s_AC4R		

```
IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype> borderValue[4], Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>border</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderInMem    Border is obtained from the source image pixels in memory. ippBorderMirror    Border pixels are mirrored from the source image boundary pixels.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterBoxBorderGetBufferSize` function.

This function operates with ROI.

This function sets each pixel in the destination image as the average of all pixels of the source image in the rectangular neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels. If *pSrc* is equal to *pDst*, `ippiFilterBoxBorder` operates as an in-place function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> or <i>pDst</i> is NULL.

---

ippStsSizeErr	Indicates an error if <i>roiSize</i> has a field with zero or negative value.
ippStsMaskSizeErr	Indicates an error if <i>mask</i> has an illegal value.
ippStsBorderErr	Indicates an error when <i>border</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilterBoxBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples> :

## FilterBox

*Blurs an image using a simple box filter.*

---

### Syntax

```
IppStatusippiFilterBox_64f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image as the average of all the input image pixels in the rectangular neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of smoothing or blurring on the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> is NULL.
ippStsSizeErr	Indicates an error if <i>dstRoiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsMaskSizeErr	Indicates an error if <i>maskSize</i> has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if <i>anchor</i> is outside the mask size.
ippStsMemAllocErr	Indicates a memory allocation error.

## FilterGaussianBorder

Performs Gaussian filtering of an image with user-defined borders.

---

### Syntax

```
IppStatusippiFilterGaussianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> borderValue,  
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R            16u\_C1R            16s\_C1R            32f\_C1R

```
IppStatusippiFilterGaussianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> borderValue[3],  
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R            16u\_C3R            16s\_C3R            32f\_C3R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

---

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the Gaussian specification structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the Gaussian filter to the source image ROI *pSrc*. The kernel of the Gaussian filter is the matrix of size *kernelSize* $\times$ *kernelSize* with the standard deviation *sigma*. The values of the Gaussian kernel elements are computed by the [FilterGaussianInit](#) function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.

Before using the `ippiFilterGaussianBorder` function, compute the size of the Gaussian specification structure and the external buffer using the [FilterGaussianGetBufferSize](#) function and initialize the structure using the [FilterGaussianInit](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values is not divisible by <code>sizeof(Ipp&lt;dataType&gt;)</code> .
<code>ippStsBorderErr</code>	Indicates an error when <i>borderType</i> has an illegal value.
<code>ippStsBadArgErr</code>	Indicates an error when <i>kernelSize</i> is even, or less than 3.

## Example

To better understand usage of this function, refer to the `FilterGaussianBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterGaussianGetBufferSize](#) Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

[FilterGaussianInit](#) Initializes the Gaussian context structure.

## SumWindow

*Sums pixel values in a rectangular area applied to an image.*

---

## Syntax

```
IppStatusippiSumWindow_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16s32f_C1R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16s32f_C3R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16s32f_C4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16s32f_AC4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16u32f_C1R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16u32f_C3R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16u32f_C4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_16u32f_AC4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);  
  
IppStatusippiSumWindow_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);
```

---

```
IppStatusippiSumWindow_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>BorderType</i>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBoderRepl        Border is replicated from the edge pixels. ippBorderInMem      Border is obtained from the source image pixels in memory. ippBorderMirror     Border pixels are mirrored from the source image boundary pixels.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
<i>borderValue</i>	Constant value to assign to border pixels. This parameter is applicable only to the ippBorderConst border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the [SumWindowGetBufferSize](#) function.

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* to the sum of all the source image pixels in the rectangular neighborhood of size *maskSize* with the anchor cell at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation while processing the image boundary pixels, the application must correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error if <i>maskSize</i> has a field with a zero or negative value.
<code>ippStsBorderErr</code>	Indicates an error if <i>BorderType</i> has an illegal value.

## SumWindowGetBufferSize

*Computes the size of the external buffer for the SumWindow function.*

---

### Syntax

```
IppStatusippiSumWindowGetBufferSize(IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>roiSize</i>	Maximum size of the destination ROI in pixels.
<i>maskSize</i>	Size of the filter mask in pixels.
<i>dataType</i>	Data type of the image. Possible values are: <code>ipp8u</code> , <code>ipp16s</code> , <code>ipp16u</code> , and <code>ipp32f</code>
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, and 4.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external work buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size, in bytes, of the external work buffer for the `ippiSumWindow` function. The result is stored in the *pBufferSize* parameter.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error if <i>roiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if <i>maskSize</i> has an illegal value.
ippStsDataTypeErr	Indicates an error if <i>dataType</i> has an illegal value.
ippStsNumChannelsError	Indicates an error if <i>numChannels</i> has an illegal value.

## SumWindowRow

Sums pixel values in the row mask applied to the image.

### Syntax

```
IppStatusippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);
```

Supported values for *mod*:

8u32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	16u32f_C4R	16s32f_C4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the horizontal row mask in pixels.
<i>anchor</i>	Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the horizontal row mask of size *maskSize* with the anchor cell *anchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if <i>pSrc</i> , <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error if <i>dstRoiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if <i>maskSize</i> has a field with a zero or negative value.
ippStsAnchorErr	Indicates an error if <i>anchor</i> is outside the mask size.
ippStsMemAllocErr	Indicates a memory allocation error.

## SumWindowColumn

*Sums pixel values in the column mask applied to the image.*

---

### Syntax

```
IppStatusippiSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);
```

Supported values for mod:

8u32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	16u32f_C4R	16s32f_C4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.

---

<i>maskSize</i>	Size of the vertical column mask in pixels.
<i>anchor</i>	Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the vertical column mask of size *maskSize* with the anchor cell *anchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if <i>pSrc</i> , <i>pDst</i> is NULL.
ippStsSizeErr	Indicates an error if <i>dstRoiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if <i>maskSize</i> has a field with a zero or negative value.
ippStsAnchorErr	Indicates an error if <i>anchor</i> is outside the mask size.
ippStsMemAllocErr	Indicates a memory allocation error.

## FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize

Compute the size of the work buffer for the maximum/minimum filter.

---

## Syntax

```
IppStatusippiFilterMaxBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
IppStatusippiFilterMinBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>maskSize</i>	Size of the filter kernel.
<i>dataType</i>	Data type of the source and destination images.

<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external buffer.

## Description

The `ippiFilterMaxBorderGetBufferSize` and `ippiFilterMinBorderGetBufferSize` functions compute the size, in bytes, of the external work buffer for the `ippiFilterMaxBorder` and `ippiFilterMinBorder` functions, respectively. The result is stored in the *pBufferSize* parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>maskSize</i> is less than, or equal to zero.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsNumChannelErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterMaxBorder](#), [FilterMinBorder](#) Filter an image using the maximum/minimum filter.

## FilterMaxBorder, FilterMinBorder

*Filter an image using the maximum/minimum filter.*

## Syntax

### Case 1: Operating on one-channel data

```
IppStatus ippiFilterMaxBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);
```

```
IppStatusippiFilterMinBorder_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16u
borderValue, Ipp8u* pBuffer);
```

```
IppStatusippiFilterMinBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);
```

### **Case 2: Operating on multi-channel data**

```
IppStatusippiFilterMaxBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[4], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[4], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMaxBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMinBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterMinBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

*pSrc*

Pointer to the source image ROI.

*srcStep*

Distance, in bytes, between the starting points  
of consecutive lines in the source image.

*pDst*

Pointer to the destination image ROI.

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI, in pixels.
<i>maskSize</i>	Size of the filter kernel.
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to a constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderInMem    Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBorderValue[3], pBorderValue[4]</i>	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using the `ippiFilterMaxBorder` and `ippiFilterMinBorder` functions, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterMaxBorderGetBufferSize` or `ippiFilterMinBorderGetBufferSize` functions, respectively.

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

The `ippiFilterMaxBorder` and `ippiFilterMinBorder` functions apply the maximum/minimum filters, respectively, to the source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel size of the filter is arbitrary and depends on the *mask* value.

The anchor cell is the center cell of the kernel, highlighted in red. The anchor cell is defined as:

$x = (\text{maskSize.width} - 1)/2$

$y = (\text{maskSize.height} - 1)/2$

where

$(x, y)$  are cell coordinates.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsBorderErr	Indicates an error when <i>mask</i> is less than, or equal to zero.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterMaxBorderGetBufferSize](#), [FilterMinBorderGetBufferSize](#) Compute the size of the work buffer for the maximum/minimum filter.

## DecimateFilterRow, DecimateFilterColumn

*Decimates an image by rows or by columns.*

---

### Syntax

```
IppStatusippiDecimateFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);
IppStatusippiDecimateFilterColumn_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>fraction</i>	Specifies how the decimating is performed. Possible values: ippPolyphase_1_2, ippPolyphase_3_5, ippPolyphase_2_3, ippPolyphase_7_10, ippPolyphase_3_4.

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

Functions `DecimateFilterRow` and `DecimateFilterColumn` perform decimating of the source image by rows or by columns respectively. These functions use the set of special internal polyphase filters. The parameter `fraction` specifies how the decimating is performed, for example, if the parameter is set to `ippPolyphase_3_5`, then each 5 pixels in the row (or column) of the source image give 3 pixels to the destination image, if the parameter is set to `ippPolyphase_1_2`, then each two pixels in the row (or column) of the source image give 1 pixel to the destination image, an so on.

To ensure valid operation, the application must correctly define additional border pixels (see [Borders in Neighborhood Operations](#)). For all `fraction` values the width of the border is four columns/rows all around the source image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error if <code>srcStep</code> or <code>dstStep</code> is less than or equal to zero.
<code>ippStsDecimateFractionErr</code>	Indicates an error if <code>fraction</code> has an illegal value.

## Median Filters

The median filter functions perform non-linear filtering of a source image data.

These functions use either an arbitrary rectangular mask, or the following predefined masks of the `IppiMaskSize` type to filter an image:

<code>ippMskSize3x1</code>	Horizontal mask of length 3
<code>ippMskSize5x1</code>	Horizontal mask of length 5
<code>ippMskSize1x3</code>	Vertical mask of length 3
<code>ippMskSize3x3</code>	Square mask of size 3
<code>ippMskSize1x5</code>	Vertical mask of length 5
<code>ippMskSize5x5</code>	Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the `mask` mean value. [Table "Median Filter Mask, Neighborhood, and Anchor Cell"](#) lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Mask size in mask names is indicated in (xy) order. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the upper left corner of the mask.

### Median Filter Mask, Neighborhood, and Anchor Cell

Mask	Neighborhood Size		Anchor Cell
	Columns	Rows	
<code>ippMskSize3x1</code>	3	1	[1, 0]

<b>Mask</b>	<b>Neighborhood Size</b>		<b>Anchor Cell</b>
	<b>Columns</b>	<b>Rows</b>	
ippMskSize5x1	5	1	[2, 0]
ippMskSize1x3	1	3	[0, 1]
ippMskSize3x3	3	3	[1, 1]
ippMskSize1x5	1	5	[0, 2]
ippMskSize5x5	5	5	[2, 2]

Median filters have the effect of removing the isolated intensity spikes and can be used to reduce noise in an image.

For details on algorithms used in Intel IPP for median filtering, see [[APMF](#)].

## FilterMedianBorderGetBufferSize

*Computes the size of the work buffer for the FilterMedianBorder function.*

### Syntax

```
IppStatusippiFilterMedianBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);

IppStatusippiFilterMedianBorderGetBufferSize_T(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

### Include Files

ippi.h

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>maskSize</i>	Size of the filter mask, in pixels.
<i>dataType</i>	Data type of the source and destination images. Possible values are ipp8u, ipp16u, ipp16s, or ipp32f.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.
<i>pBufferSize</i>	Pointer to the computed size of the external work buffer, in bytes.

### Description

The `ippiFilterMedianBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterMedianBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterMedianBorder` function description.

### Return Values

ippStsNoErr	Indicates no error.
-------------	---------------------

ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when one of the <i>dstRoiSize</i> fields has a negative or zero value.
ippStsMaskSizeErr	Indicates an error when <i>mask</i> has a field with a negative, zero, or even value.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsNumChannelsError	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterMedianBorder](#) Performs median filtering of an image.

## FilterMedianBorder

*Performs median filtering of an image.*

### Syntax

#### Case 1: Operating on one-channel data

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, Ipp<datatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operating on multi-channel data

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[4], Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

#### Case 3: Operating on one-channel data with Threading Layer (TL) functions based on the Classic API

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, Ipp<datatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_CIR_T
----------

### Case 4: Operating on multi-channel data with Threading Layer (TL) functions based on the Classic API

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R_T
8u_AC4R_T

```
IppStatusippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[4], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R_T
----------

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image.						
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.						
<code>pDst</code>	Pointer to the destination image.						
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.						
<code>dstRoiSize</code>	Size of the destination ROI, in pixels.						
<code>maskSize</code>	Size of the filter mask, in pixels.						
<code>borderType</code>	Type of border. Possible values are: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Border is replicated from the edge pixels.</td></tr> <tr> <td><code>ippBorderInMem</code></td><td>Border is obtained from the source image pixels in memory.</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Border is replicated from the edge pixels.	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
<code>ippBorderConst</code>	Values of all border pixels are set to constant.						
<code>ippBorderRepl</code>	Border is replicated from the edge pixels.						
<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.						
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.						

---

<i>pBorderValue[3], pBorderValue[4]</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI.

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterMedianBorderGetBufferSize` function.

The `ippiFilterMedianBorder` function applies a median filter to an image ROI. The anchor cell is the center of the filter kernel. The size of the source image ROI is equal to the size of the destination image ROI *dstRoiSize*.

This function sets each pixel in the destination buffer as the median value of all source pixels values from the neighborhood of the processed pixel.

This function removes noise and does not cut out signal brightness drops, as an averaging filter does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> , <i>pDst</i> , <i>pBuffer</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <i>dstRoiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error if <i>maskSize</i> has a field with a zero, negative, or even value.
<code>ippStsNotEvenStepErr</code>	Indicates an error if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilterMedianBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterMedianBorderGetBufferSize](#) Computes the size of the work buffer for the `FilterMedianBorder` function.

## FilterMedianGetBufferSize

*Computes the size of the external buffer for `ippiFilterMedian` function.*

---

## Syntax

```
IppStatus ippiFilterMedianGetBufferSize_32f(IppiSize dstRoiSize, IppiSize maskSize,
Ipp32u nChannels, Ipp32u* pBufferSize);

IppStatus ippiFilterMedianGetBufferSize_64f(IppiSize dstRoiSize, IppiSize maskSize,
Ipp32u nChannels, Ipp32u* pBufferSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>nChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the computed value of the external buffer size.

## Description

This function computes the size in bytes of an external memory buffer that is required for the [ippiFilterMedian](#) function, and stores the result in the *pBufferSize*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error if the <i>pBufferSize</i> pointer is NULL.
ippStsSizeErr	Indicates an error if one of the fields of <i>dstRoiSize</i> has a zero or negative value.
ippStsMaskSizeErr	Indicates an error if one of the fields of <i>maskSize</i> has a value less than or equal to 1.
ippStsNumChannelsErr	Indicates an error if <i>nChannels</i> is not equal to 1, 3, or 4.

## FilterMedian

*Filters an image using a median filter.*

## Syntax

```
IppStatusippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for *mod*:

32f\_C3R  
32f\_C4R  
64f\_C1R

## Include Files

ippi.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>maskSize</code>	Size of the mask in pixels.
<code>anchor</code>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<code>pBuffer</code>	Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders in Neighborhood Operations](#)). The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to the size of the destination image ROI `dstRoiSize`.

Some flavors of the function require the external buffer `pBuffer`. Prior to using this functions, compute the size of the external buffer by using the function [FilterMedianGetBufferSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error if <code>maskSize</code> has a field with zero, negative, or even value.
<code>ippStsAnchorErr</code>	Indicates an error if <code>anchor</code> is outside the mask size.

## FilterMedianCross

[Filters an image using a cross median filter.](#)

## Syntax

```
IppStatusippiFilterMedianCross_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of the <code>IppiMaskSize</code> type.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either `ippMskSize3x3` or `ippMskSize5x5` (see [Table “Median Filter Mask, Neighborhood, and Anchor Cell”](#)). The function operates on the assumption that the pixels outside the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide appropriate values for the `pSrc` and `dstRoiSize` arguments, or define additional border pixels (see [Borders in Neighborhood Operations](#)). The size of the source image ROI is equal to the size of the destination image ROI `dstRoiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

---

ippStsMaskSizeErr	Indicates an error if <i>mask</i> has an illegal value.
-------------------	---

## FilterMedianWeightedCenter3x3

*Filters an image using a median filter with a weighted center pixel.*

---

### Syntax

```
IppStatusippiFilterMedianWeightedCenter3x3_8u_C1R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int weight);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>weight</i>	Weight of the pixel, must be an odd number.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the fixed square mask of the 3x3 size with the anchor cell as the center cell of the mask. The parameter *weight* specifies the weight of the processed pixel, that is how many times its value is included into calculations. The value of this parameter should be odd. If it is even, the function changes its value to the nearest less odd number and returns the warning message.

The function operates on the assumption that the pixels outside of the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders in Neighborhood Operations](#)). The size of the source image ROI is equal to the size of the destination image ROI *dstRoiSize*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.

ippStsSizeErr	Indicates an error if <i>dstRoiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsWeightErr	Indicates an error if <i>weight</i> is less than or equal to 0.
ippStsEvenMedianWeight	Indicates a warning if <i>weight</i> has an even value.

## FilterMedianColor

*Filters an image using a color median filter.*

---

### Syntax

```
IppStatusippiFilterMedianColor_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of the <i>IppiMaskSize</i> type.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the `ippiFilterMedianColor` function instead. For each input pixel, this function computes differences between red (R), green (G), and blue (B) color components of pixels in the *mask* neighborhood and the input pixel. The distance between the input pixel *i* and the neighborhood pixel *j* is formed as the sum of absolute values:

$$\text{abs } (R(i)-R(j)) + \text{abs } (G(i)-G(j)) + \text{abs } (B(i)-B(j))$$

After scanning the entire neighborhood, the function sets the output value for pixel  $i$  as the value of the neighborhood pixel with the smallest distance to  $i$ .

The function `ippiFilterMedianColor` supports square masks of size either `ippMskSize3x3` or `ippMskSize5x5` and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error if <code>mask</code> has an illegal value.

## General Linear Filters

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. This array must exactly have `kernelSize.width * kernelSize.height` entries. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the lower right corner of the kernel.

The output value is computed as a sum of neighbor pixels values, with kernel matrix elements used as weight factors. Summation formulas implement a convolution operation, which means that kernel coefficients are used in direct order.

### NOTE

In Intel IPP 8.2 and lower versions, kernel coefficients are used in inverse order.

Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

## FilterBorderGetSize

*Computes the size of the filter specification structure and the size of the work buffer.*

### Syntax

```
IppStatus ippiFilterBorderGetSize (IppiSize kernelSize, IppiSize dstRoiSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

### Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>kernelSize</code>	Size of the rectangular kernel in pixels.
<code>dstRoiSize</code>	Maximal size of the destination image ROI (in pixels).
<code>dataType</code>	Data type of the source image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , or <code>ipp64f</code> .
<code>kernelType</code>	Data type of the filter kernel. Possible values are <code>ipp16s</code> , <code>ipp32f</code> , or <code>ipp64f</code> .
<code>numChannels</code>	Number of channels in the image. Possible values are 1, 3, or 4.
<code>pSpecSize</code>	Pointer to the size of the filter specification structure.
<code>pBufferSize</code>	Pointer to the size of the work buffer required for filtering.

## Description

This function operates with ROI.

This function computes the size of the filter specification structure `pSpec` and the size of the buffer required for filtering operations. Call this function before using the `ippiFilterBorderInit` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>kernelSize</code> has a field with a zero or negative value, or if <code>dstRoiSize</code> is less than 1.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> or <code>kernelType</code> has an illegal value.
<code>ippStsChannelErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[FilterBorderInit](#) Initializes the filter specification structure.

[FilterBorder](#) Filters an image using a rectangular filter.

## FilterBorderInit

*Initializes the filter specification structure.*

### Syntax

```
IppStatus ippiFilterBorderInit_16s(const Ipp16s* pKernel, IppiSize kernelSize, int divisor, IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec);
```

```
IppStatusippiFilterBorderInit_32f(const Ipp32f* pKernel, IppiSize kernelSize,
IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec*
pSpec);
```

```
IppStatusippiFilterBorderInit_64f(const Ipp64f* pKernel, IppiSize kernelSize,
IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec*
pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pKernel</i>	Pointer to the kernel values.								
<i>kernelSize</i>	Size of the rectangular kernel in pixels.								
<i>divisor</i>	Integer value by which the computed result is divided.								
<i>dataType</i>	Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, or ipp64f.								
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.								
<i>roundMode</i>	Rounding mode, possible values: <table border="0"> <tr> <td>ippRndZero</td><td>Floating-point values are truncated to zero. This mode is not supported forippiFilterBorderInit_64f.</td></tr> <tr> <td>ippRndNear</td><td>Floating-point values are rounded to the nearest even integer when the fractional part equals to 0.5; otherwise they are rounded to the nearest integer.</td></tr> <tr> <td>ippRndFinancial</td><td>Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5. This mode is not supported forippiFilterBorderInit_64f.</td></tr> <tr> <td>ippRndHintAccurate</td><td>The result of calculations is accurate. This mode is supported only when:           <ul style="list-style-type: none"> <li>• <i>dataType</i> is equal to 8u and <i>numChannels</i> is equal to 1, 3, or 4</li> <li>• <i>dataType</i> is equal to 16s and <i>numChannels</i> is equal to 1</li> </ul> </td></tr> </table>	ippRndZero	Floating-point values are truncated to zero. This mode is not supported forippiFilterBorderInit_64f.	ippRndNear	Floating-point values are rounded to the nearest even integer when the fractional part equals to 0.5; otherwise they are rounded to the nearest integer.	ippRndFinancial	Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5. This mode is not supported forippiFilterBorderInit_64f.	ippRndHintAccurate	The result of calculations is accurate. This mode is supported only when: <ul style="list-style-type: none"> <li>• <i>dataType</i> is equal to 8u and <i>numChannels</i> is equal to 1, 3, or 4</li> <li>• <i>dataType</i> is equal to 16s and <i>numChannels</i> is equal to 1</li> </ul>
ippRndZero	Floating-point values are truncated to zero. This mode is not supported forippiFilterBorderInit_64f.								
ippRndNear	Floating-point values are rounded to the nearest even integer when the fractional part equals to 0.5; otherwise they are rounded to the nearest integer.								
ippRndFinancial	Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5. This mode is not supported forippiFilterBorderInit_64f.								
ippRndHintAccurate	The result of calculations is accurate. This mode is supported only when: <ul style="list-style-type: none"> <li>• <i>dataType</i> is equal to 8u and <i>numChannels</i> is equal to 1, 3, or 4</li> <li>• <i>dataType</i> is equal to 16s and <i>numChannels</i> is equal to 1</li> </ul>								
<i>pSpec</i>	Pointer to the filter specification structure.								

## Description

This function initializes the filter specification structure *pSpec* in the external buffer. Before using this function, you need to compute the size of the specification structure using the `FilterBorderGetSize` function. This structure is used by the `FilterBorder` function that performs filtering operations on the source image pixels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>kernelSize</code> has a field with a zero or negative value.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsRoundModeNotSupportedErr</code>	Indicates an error when the specified rounding mode is not supported.
<code>ippStsAccurateModeNotSupported</code>	Indicates a warning when the <code>ippRndHintAccurate</code> mode is not supported.

## See Also

[FilterBorderGetSize](#) Computes the size of the filter specification structure and the size of the work buffer.

[FilterBorder](#) Filters an image using a rectangular filter.

## FilterBorder

*Filters an image using a rectangular filter.*

---

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype> borderValue[1], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`
- `16u_C1R`
- `16s_C1R`
- `32f_C1R`
- `64f_C1R`

### Case 2: Operation on multi-channel data

```
IppStatusippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype> borderValue[3], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C3R`
- `16u_C3R`
- `16s_C3R`

`32f_C3R`

```
IppStatusippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*>
pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype>
borderValue[4], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C4R`  
`16u_C4R`  
`16s_C4R`  
`32f_C4R`

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between starting points of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination image ROI in pixels.
<code>border</code>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl        Border is replicated from the edge pixels. ippBorderInMem      Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between the <code>ippBorderRepl</code> or <code>ippBorderConst</code> values and <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> .
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pSpec</code>	Pointer to the filter specification structure.

*pBuffer* Pointer to the work buffer for filtering operations.

## Description

Before using this function, you need to initialize the filter specification structure using the `ippiFilterBorderInit` function.

This function operates with ROI.

This function performs filtering of a rectangular ROI inside a two-dimensional image using a specified structure *pSpec*. Type of the image border is defined by the value of the *border* parameter.

To change the function behavior (add offset to the result or set the rounding mode), use `ippiFilterBorderSetMode` after the `ippiFilterBorderInit` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value, or if <i>dstRoiSize.width</i> is more than the maximum ROI <i>roiWidth</i> passed to the initialization function.
<code>ippStsStepErr</code>	Indicates an error when the <i>srcStep</i> value is less than, or equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when <i>border</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilterBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

`FilterBorderInit` Initializes the filter specification structure.

`FilterBorderSetMode` Adds the offset value after filtering operation for `ipp8u` and `ipp16u` data types , and sets the rounding mode.

## FilterBorderSetMode

*Adds the offset value after filtering operation for ipp8u and ipp16u data types , and sets the rounding mode.*

## Syntax

```
IppStatusippiFilterBorderSetMode(IppHintAlgorithm hint, int offset,  
IppiFilterBorderSpec* pSpec);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>hint</i>	Suggests using specific code for rounding. Supported values:  ippAlgHintNone, ippAlgHintFast	Default modes. The function performs rounding in accordance with the <i>roundMode</i> parameter passed to the <i>Init</i> function, but function performance takes precedence over accuracy and some output pixels can differ by +1 from the exact result.
	ippAlgHintAccurate	All output pixels are exact; accuracy takes precedence over performance.
<i>offset</i>	Constant that is added to the final signed result before converting it to unsigned for <i>ipp8u</i> and <i>ipp16u</i> data types.	
<i>pSpec</i>	Pointer to the initialized filter specification structure.	

## Description

This function adds the *offset* value after filtering operation for *ipp8u* and *ipp16u* data types with the *ippiFilterBorder* function:

```
pDst=(summ(src[i]*kern[i]))+offset
```

You can also use this function to set the rounding mode for the filtering result.

The *8u\_C1R*, *8u\_C3R*, *8u\_C4R*, and *16s\_C1RFILTERBorder* function flavors initialized with the *ipp16s* coefficients support *ippAlgHintNone* and *ippAlgHintAccurate* rounding modes.

Use this function after the *ippiFilterBorderInit* function and before calling *ippiFilterBorder*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when one of the specified pointers is NULL.
<i>ippStsNotSupportedModeErr</i>	The offset value is not supported (for <i>ipp16s</i> and <i>ipp32f</i> data types).
<i>ippStsAccurateModeNotSupported</i>	The accurate mode is not supported for some data types. The result of rounding may be not exact.

## Example

To better understand usage of this function, refer to the *FilterBorderSetMode.c* example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

**FilterBorder** Filters an image using a rectangular filter.

**FilterBorderInit** Initializes the filter specification structure.

## FilterGetBufSize

*Computes the size of the work buffer.*

## Syntax

```
IppStatus ippiFilterGetBufSize_64f_C1R(IppiSize kernelSize, int roiWidth, int* pSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>roiWidth</i>	Width of the image ROI in pixels.
<i>pSize</i>	Pointer to the size of the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the work buffer *pSize* that is required for the function [ippiFilter](#) (flavor that operates on data of the `Ipp64f` type).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <i>kernelSize</i> has a field with a zero or negative value, or <i>roiWidth</i> is less than or equal to zero.

## Filter

Filters an image using a general rectangular kernel.

## Syntax

```
IppStatusippiFilter_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp64f* pKernel, IppiSize kernelSize, IppiPoint anchor, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

---

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI.

The `ippiFilter` function uses the general rectangular kernel of *kernelSize* size to filter an image ROI. This function sums the products of the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

Kernel coefficients are used in inverse order. The sum is written to the destination pixel.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders in Neighborhood Operations](#)).

This function requires a temporary work buffer. Before using this function flavor, you need to compute the buffer size using the `ippiFilterGetBufSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> or <i>kernelSize</i> has a field with a zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error when the <i>divisor</i> has a zero value.
<code>ippStsStepErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <i>srcStep</i> is less than <math>(roiSize.width + kernelSize.width) * \text{sizeof}(\text{Ipp64f})</math></li> <li>• <i>dstStep</i> is less than <math>roiSize.width * \text{sizeof}(\text{Ipp64f})</math></li> </ul>

## See Also

[Regions of Interest in Intel IPP](#)

[Borders in Neighborhood Operations](#)

[FilterGetBufSize](#) Computes the size of the work buffer.

## Separable Filters

---

Separable filters use a spatial kernel consisting of a single column (as in the `FilterColumn` function) or a single row (as in the `FilterRow` function) to filter the source image.

## FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize\_Low

Compute the size of working buffer for the strow filter.

### Syntax

```
IppStatusippiFilterRowBorderPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

8u16s_C1R	16s_C1R	16u_C1R	32f_C1R
8u16s_C3R	16s_C3R	16u_C3R	32f_C3R

```
IppStatusippiFilterRowBorderPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

16s_C1R
16s_C3R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

### Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the size of the working buffer required for the functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
ippStsSizeErr	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.

## FilterRowBorderPipeline, FilterRowBorderPipeline\_Low

Apply the filter with border to image rows.

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int
kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue, int
divisor, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R 16s\_C1R 16u\_C1R

```
IppStatusippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc, int srcStep,
Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp16s borderValue, int divisor, Ipp8u* pBuffer);
```

#### Case 2: Operation on one-channel floating point data

```
IppStatusippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f** ppDst,
IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

#### Case 3: Operation on three-channel integer data

```
IppStatusippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int
kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue[3],
int divisor, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C3R 16s\_C3R 16u\_C3R

```
IppStatusippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc, int srcStep,
Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp16s borderValue[3], int divisor, Ipp8u* pBuffer);
```

#### Case 4: Operation on three-channel floating point data

```
IppStatusippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f** ppDst,
IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp32f borderValue[3], Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>ppDst</i>	Double pointer to the destination image ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the row kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>xAnchor</i>	Anchor value specifying the kernel row alignment with respect to the position of the input pixel.
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible:
	ippBorderZero                          Values of all border pixels are set to zero.
	ippBorderConst                        Values of all border pixels are set to constant.
	ippBorderRepl                        Replicated border is used.
	ippBorderWrap                        Wrapped border is used
	ippBorderMirror                      Mirrored border is used
	ippBorderMirrorR                    Mirrored border with replication is used
<i>borderValue</i>	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).
<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

These functions operate with ROI (see Regions [Regions of Interest in Intel IPP](#) ).

The function `ippiFilterRowBorderPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the horizontal row filter of the separable convolution kernel to the source image `pSrc`. The filter coefficients are placed in the reversed order. For integer data:

$$ppDst[i][j] = \frac{1}{\text{divisor}} \cdot \sum_{k=0}^{\text{kernelSize}-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

and for floating point data:

$$ppDst[i][j] = \sum_{k=0}^{\text{kernelSize}-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here  $j = 0, \dots, roiSize.width - 1, i = 0, \dots, roiSize.height - 1$ . The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel  $pSrc[i, l] \notin [0, roiSize.width-1]$ ) are set in accordance with the specified parameters *borderType* and *borderValue*.

This function can be used to organize the separable convolution as a step of the image processing pipeline. The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions [ippiFilterRowBorderPipelineGetBufferSize](#) and [ippiFilterRowBorderPipelineGetBufferSize\\_Low](#) respectively

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> has a wrong value.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

## FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize\_Low

Compute the size of working buffer for the column filter.

### Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize,
int* pBufferSize);
```

Supported values for *mod*:

<code>16s_C1R</code>	<code>16u_C1R</code>	<code>16s8u_C1R</code>	<code>16s8s_C1R</code>	<code>32f_C1R</code>
<code>16s_C3R</code>	<code>16u_C3R</code>	<code>16s8u_C3R</code>	<code>16s8s_C3R</code>	<code>32f_C3R</code>

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int
kernelSize, int* pBufferSize);
```

Supported values for *mod*:

<code>16s_C1R</code>
<code>16s_C3R</code>

### Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the size of the working buffer required for the functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.

## FilterColumnPipeline, FilterColumnPipeline\_Low

*Apply the filter to image columns.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatusippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<srcDatatype>* pKernel,
int kernelSize, int divisor, Ipp8u* pBuffer);
```

Supported values for mod:

16s_C1R	16s8u_C1R	16s8s_C1R	16u_C1R
16s_C3R	16s8u_C3R	16s8s_C3R	16u_C3R

```
IppStatusippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc, Ipp16s* pDst, int
dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u*
pBuffer);
```

```
IppStatusippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc, Ipp16s* pDst, int
dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u*
pBuffer);
```

## Case 2: Operation on floating-point data

```
IppStatusippiFilterColumnPipeline_<mod>(const Ipp<datatype>** ppSrc, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const Ipp<datatype>* pKernel, int kernelSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`  
`32f_C3R`

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>ppSrc</code>	Double pointer to the source image ROI.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the destination ROI in pixels.
<code>pKernel</code>	Pointer to the strow kernel values.
<code>kernelSize</code>	Size of the kernel in pixels.
<code>divisor</code>	Value by which the computed result is divided (for operations on integer data only).
<code>pBuffer</code>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiFilterColumnPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the column filter of the separable convolution kernel to the source image `pSrc`. The filter coefficients are placed in the reversed order. For integer data:

$$pDst[i, j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

and for floating point data:

$$pDst[i, j] = \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

Here  $j = 0, \dots, \text{roiSize.width}-1, i=0, \dots, \text{roiSize.height}-1$ .

The size of the source image is

$(\text{roiSize.height} + \text{kernelSize} - 1) * \text{roiSize.width}$ .

The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions *ippiFilterColumnPipelineGetBufferSize* and *ippiFilterColumnPipelineGetBufferSize\_Low* respectively.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than $\text{roiSize.width} * \langle \text{pixelSize} \rangle$
<i>ippStsNotEvenStepErr</i>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<i>ippStsBadArgErr</i>	Indicates an error condition if <i>divisor</i> is equal to 0.

## Example

To better understand usage of this function, refer to the *FilterColumnPipeline\_Low.c* example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## FilterSeparable

*Apply the filter to an image.*

---

### Syntax

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16s_C1R	16u_C1R	32f_C1R
8u_C3R	16s_C3R	16u_C3R	32f_C3R
8u_C4R	16s_C4R	16u_C4R	32f_C4R
8u_C1R_T	16s_C1R_T	16u_C1R_T	32f_c1R_T
8u_C3R_T	16s_C3R_T	16u_C3R_T	32f_C3R_T
8u_C4R_T	16s_C4R_T	16u_C4R_T	32f_C4R_T

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<srcdatatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R
-----------

8u16s_C3R
8u16s_C4R
8u16s_C1R_T
8u16s_C3R_T
8u16s_C4R_T

### Platform-aware functions

```
IppStatusippiFilterSeparable_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R_L	16s_C1R_L	16u_C1R_L	32f_C1R_L
8u_C3R_L	16s_C3R_L	16u_C3R_L	32f_C3R_L
8u_C4R_L	16s_C4R_L	16u_C4R_L	32f_C4R_L
8u_C1R_LT	16s_C1R_LT	16u_C1R_LT	32f_C1R_LT
8u_C3R_LT	16s_C3R_LT	16u_C3R_LT	32f_C3R_LT
8u_C4R_LT	16s_C4R_LT	16u_C4R_LT	32f_C4R_LT

```
IppStatusippiFilterSeparable_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
Ipp<srcdatatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R_L
8u16s_C3R_L
8u16s_C4R_L
8u16s_C1R_LT
8u16s_C3R_LT
8u16s_C4R_LT

### Include Files

ippcv.h

ippcv\_l.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>roiSize</i>	Size of the source and destination ROI in pixels.	
<i>borderType</i>	Type of border. Possible values are:	
	<i>ippBorderConst</i>	Values of all border pixels are set to constant.
	<i>ippBorderRept</i>	Replicated border is used.
	<i>ippBorderWrap</i>	Wrapped border is used.
	<i>ippBorderMirror</i>	Mirrored border is used.
	<i>ippBorderMirrorR</i>	Mirrored border with replication is used.
<i>borderValue</i>	Constant value (constant vector in case of three- or four-channel data) to assign to the pixels in the <i>ippBorderConst</i> border type (not applicable for other border types).	
<i>pSpec</i>	Pointer to the filter specification structure.	
<i>pBuffer</i>	Pointer to the working buffer.	

## Description

This function operates with ROI.

Before using this function, compute the size of the external buffer *pBuffer* using the [ippiFilterSeparableGetBufferSize](#) function.

This function applies the horizontal row filter of the separable convolution kernel to the source image *pSrc* and the column filter of the separable convolution kernel to the intermediate result.

For integer data:

$$\text{intermediate}[i][j] = \frac{1}{\text{divisor}} * \sum_{k=0}^{\text{rowKernelSize}-1} (\text{pSrc}[i, j + k - \text{xAnchor}] * \text{pRowKernel}[k])$$

$$\text{pDst}[i, j] = \text{offset} + \frac{1}{\text{divisor}} * \sum_{k=0}^{\text{columnKernelSize}-1} (\text{intermediate}[i + k, j] * \text{pColumnKernel}[k])$$

and for floating point data:

$$\text{intermediate}[i][j] = \sum_{k=0}^{\text{rowKernelSize}-1} (\text{pSrc}[i, j + k - \text{xAnchor}] * \text{pRowKernel}[k])$$

$$\text{pDst}[i, j] = \sum_{k=0}^{\text{columnKernelSize}-1} (\text{intermediate}[i + k, j] * \text{pColumnKernel}[k])$$

Here  $j = 0, \dots, \text{roiSize.width} - 1$ ,  $i = 0, \dots, \text{roiSize.height} - 1$ . The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel  $\text{pSrc}[i, l]$   $l \notin [0, \text{roiSize.width}-1]$ ) are set in accordance with the specified parameters *borderType* and *borderValue*.  $\text{xAnchor} = (\text{rowKernelSize} - 1) / 2$ .

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error condition.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if the <i>srcStep</i> or <i>dstStep</i> value is less than <i>roiSize.width* &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterSeparableInit](#) Initializes the filter specification structure.

[FilterSeparableGetBufferSize](#) Computes the size of the work buffer.

## FilterSeparableGetBufferSize

*Computes the size of the work buffer.*

### Syntax

```
IppStatusippiFilterSeparableGetBufferSize(IppiSize roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pBufferSize);

IppStatusippiFilterSeparableGetBufferSize_L(IppiSizeL roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pBufferSize);

IppStatusippiFilterSeparableGetBufferSize_T(IppiSize roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pBufferSize);

IppStatusippiFilterSeparableGetBufferSize_LT(IppiSizeL roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pBufferSize);
```

### Include Files

ippcv.h

ippcv\_1.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the image ROI in pixels.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>dataType</i>	Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.

<i>kernelType</i>	Data type of the filter kernel. Possible values are <code>Ipp16s</code> and <code>Ipp32f</code> .
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, and 4.
<i>pBufferSize</i>	Pointer to the size of the work buffer required for filtering.

## Description

This function computes the size of the buffer required for filtering operations. Call this function before using the [ippiFilterSeparable](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>kernelSize</code> has a field with a zero or negative value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> has an illegal value.
<code>ippStsChannelErr</code>	Indicates an error condition if <code>numChannels</code> has an illegal value.

## See Also

[FilterSeparable](#) Apply the filter to an image.

## FilterSeparableGetSpecSize

Computes the size of the filter specification structure.

## Syntax

```
IppStatusippiFilterSeparableGetSpecSize(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);

IppStatusippiFilterSeparableGetSpecSize_L(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);

IppStatusippiFilterSeparableGetSpecSize_T(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);

IppStatusippiFilterSeparableGetSpecSize_LT(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);
```

## Include Files

`ippcv.h`  
`ippcv_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>dataType</i>	Data type of the source image. Possible values are <code>Ipp8u</code> , <code>Ipp16s</code> , <code>Ipp16u</code> , <code>Ipp32f</code> .
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, and 4.
<i>pSpecSize</i>	Pointer to the size of the filter specification structure.

## Description

This function computes the size of the filter specification structure *pSpec*. Call this function before using the [ippiFilterSeparableInit](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> has a field with a zero or negative value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <i>dataType</i> has an illegal value.
<code>ippStsChannelErr</code>	Indicates an error condition if <i>numChannels</i> has an illegal value.

## See Also

[FilterSeparableInit](#) Initializes the filter specification structure.

## FilterSeparableInit

*Initializes the filter specification structure.*

### Syntax

```
IppStatus ippiFilterSeparableInit_16s(const Ipp16s* pRowKernel, const Ipp16s* pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);

IppStatus ippiFilterSeparableInit_32f(const Ipp32f* pRowKernel, const Ipp32f* pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);
```

### Platform-aware functions

```
IppStatus ippiFilterSeparableInit_16s_L(const Ipp16s* pRowKernel, const Ipp16s* pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);

IppStatus ippiFilterSeparableInit_32f_L(const Ipp32f* pRowKernel, const Ipp32f* pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);

IppStatus ippiFilterSeparableInit_16s_T(const Ipp16s* pRowKernel, const Ipp16s* pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);
```

```
IppStatusippiFilterSeparableInit_32f_T(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);

IppStatusippiFilterSeparableInit_16s_LT(const Ipp16s* pRowKernel, const Ipp16s*
pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType,
int numChannels, IppiFilterSeparableSpec* pSpec);

IppStatusippiFilterSeparableInit_32f_LT(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);
```

## Include Files

ippcv.h  
ippcv\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pRowKernel</i> , <i>pColumnKernel</i>	Pointer to the row and column kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>divisor</i>	Integer value by which the computed result is divided.
<i>scaleFactor</i>	Integer value by which the computed result is divided.
<i>dataType</i>	Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, and 4.
<i>pSpec</i>	Pointer to the filter specification structure.

## Description

This function initializes the filter specification structure *pSpec* in the external buffer. Before using this function, you need to compute the size of the specification structure using the `FilterSeparableGetSpecSize` function. This structure is used by the `FilterSeparable` function that performs filtering operations on the source image pixels.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>kernelSize</i> has a field with a zero or negative value.
ippStsDataTypeErr	Indicates an error condition if <i>dataType</i> has an illegal value.
ippStsChannelErr	Indicates an error condition if <i>numChannels</i> has an illegal value.
ippStsDivisorErr	Indicates an error condition if <i>divisor</i> has a zero value.

## See Also

[FilterSeparable](#) Apply the filter to an image.

[FilterSeparableGetSpecSize](#) Computes the size of the filter specification structure.

# Smoothing Filters

Smoothing filters use edge-preserving real-time Iterative Least Squares algorithm for image processing. This algorithm is described in the article "Real-time Image Smoothing via Iterative Least Squares. Wei Liu, Pingping Zhang, Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid". Link to the publication: <https://arxiv.org/abs/2003.07504>.

## FilterILSInit

*Initializes the filter specification structure.*

### Syntax

```
IppStatusippiFilterILSInit(IppiFilterILSType filter, IppiSize dstRoiSize, IppDataType dataType, int numChannels, Ipp64f lambda, Ipp64f eps, Ipp64f pow, Ipp64f gamma, IppiFilterILSSpec* pSpec, Ipp8u* pBufInit);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>filter</i>	Filter type: Norm or Welsch.
<i>dataType</i>	Data type flavors.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>numChannels</i>	Number of channels.
<i>pSpecSize</i>	Pointer to the size (in bytes) of the specification structure.
<i>pBufInitSize</i>	Pointer to the size (in bytes) of the init temp buffer.
<i>pBufferSize</i>	Pointer to the size (in bytes) of the work buffer.
<i>lambda</i>	Lambda for Norm and Welsch.
<i>eps</i>	Eps for Norm.
<i>pow</i>	P for Norm.
<i>gamma</i>	Gamma for Welsch.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBufInit</i>	Pointer to the buffer for init function.

### Description

This function initializes the filter specification structure *pSpec* in the work buffer.

Before using this function, you need to compute the size of the specification structure using the `ippiFilterILSGetBufferSize` function. This structure is used by the `ippiFilterILS` function that performs edge-preserving image smoothing via Iterative Least Squares algorithm.

The `ippiFilterILSInit` function takes the following parameters: `lambda`, `eps`, `p`, `gamma` with values that are described in the article "Real-time Image Smoothing via Iterative Least Squares. Wei Liu, Pingping Zhang, Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid". Link to the publication: <https://arxiv.org/abs/2003.07504>.

Use the `ippiFilterILSGetBufferSize` function to allocate the work buffer `pBuffer` and `pBufInit`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if one of the steps has a zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.

## FilterILSGetBufferSize

*Computes the size of the work buffer.*

---

### Syntax

```
IppStatus ippiFilterILSGetBufferSize (IppiFilterILSType filter, IppiSize dstRoiSize,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufInitSize, int*
pBufferSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>filter</code>	Filter type: <code>Norm</code> or <code>Welsch</code> .
<code>dataType</code>	Data type flavors.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>numChannels</code>	Number of channels.
<code>pSpecSize</code>	Pointer to the size (in bytes) of the specification structure.
<code>pBufInitSize</code>	Pointer to the size (in bytes) of the init temp buffer.
<code>pBufferSize</code>	Pointer to the size (in bytes) of the work buffer.

### Description

This function computes the size of the buffers required for filtering operations. Call this function before using the `ippiFilterILSInit` and `ippiFilterILS` functions.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if one of the steps has a zero or negative value.
ippStsSizeErr	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.

## FilterILS

*Filters an image using smoothing algorithm.*

### Syntax

```
IppStatusippiFilterILS_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize dstRoiSize, int nIter, IppiFilterILSSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C1R  
8u\_C3R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Step through the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Step in destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>nIter</i>	Number of iterations.
<i>pSpec</i>	Pointer to the internal data structure.
<i>pBuffer</i>	Pointer to the external work buffer.

### Description

This function implements edge-preserving image smoothing via Iterative Least Squares algorithm.

This algorithm is described in the article "Real-time Image Smoothing via Iterative Least Squares. Wei Liu, Pingping Zhang, Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid". Link to the publication: <https://arxiv.org/abs/2003.07504>.

The function `ippiFilterILS` uses the internal data structure `pSpec` and the work buffer `pBuffer`. Therefore, the work buffer `pBuffer` must be allocated and the internal data structure `pSpec` must be initialized before the function call.

Use the `ippiFilterILSGetBufferSize` function to allocate the work buffer `pBuffer`.

Use the `ippiFilterILSInit` function to initialize the internal data structure `pSpec`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if one of the steps has a zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.

# Wiener Filters

---

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter [Lim90]. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

## FilterWienerGetSize

*Computes the size of the external buffer for `ippiFilterWiener` function.*

---

### Syntax

```
IppStatusippiFilterWienerGetSize(IppiSize dstRoiSize, IppiSize maskSize, int channels, int* pBufferSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>dstRoiSize</code>	Size of the destination ROI in pixels.
<code>maskSize</code>	Size of the mask in pixels.
<code>channels</code>	Number of channels in the image.
<code>pBufferSize</code>	Pointer to the computed value of the external buffer size.

## Description

This function computes the size in bytes of an external memory buffer that is required for the function `ippiFilterWiener`, and stores the result in the `pBufferSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <code>maskSize</code> has a value less than or equal to 1.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1, 3 or 4.

## FilterWiener

*Filters an image using the Wiener algorithm.*

### Syntax

#### Case 1: Operation on one-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f noise[1], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R      16s\_C1R      32f\_C1R

#### Case 2: Operation on multi-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C4R      16s\_C4R      32f\_C4R

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>noise</i>	Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image *pSrc*, the function estimates the local image mean  $\mu$  and variance  $\sigma$  in the rectangular neighborhood (mask) of size *maskSize* with the anchor cell *anchor* centered on the pixel. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the mask.

The following formulas are used in computations:

$$\mu_{i,j} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}$$

$$\sigma_{i,j}^2 = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}^2 - \mu_{i,j}^2$$

Here  $\mu_{i,j}$  and  $\sigma_{i,j}$  stand for local mean and variance for pixel  $x_{i,j}$ , respectively, and  $H, W$  are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel  $y_{i,j}$  is computed as:

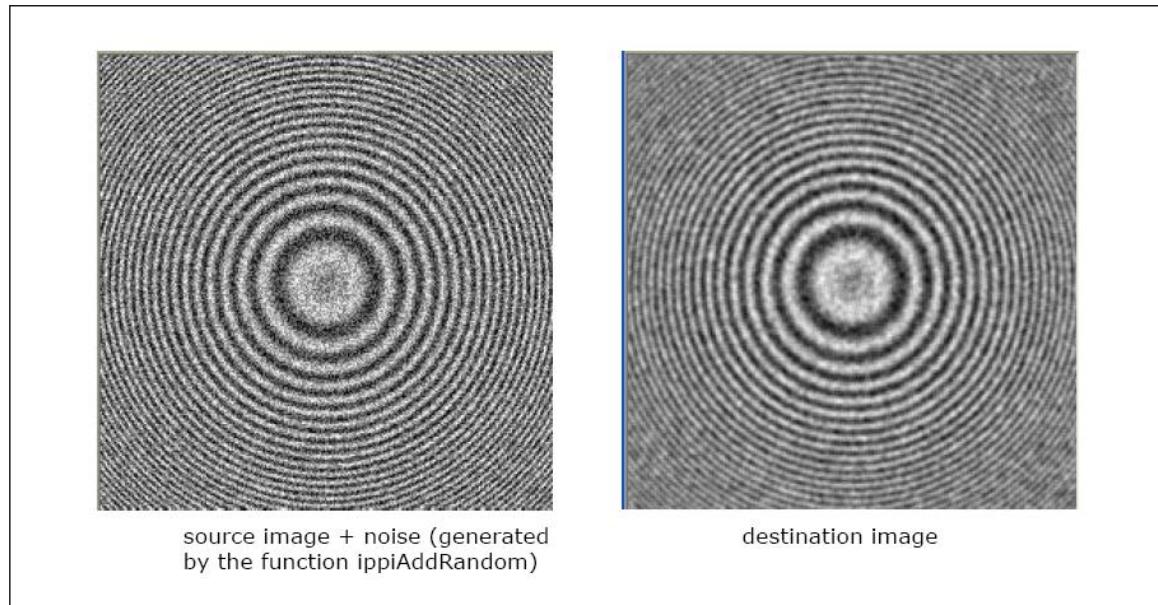
$$Y_{i,j} = \mu_{i,j} + \frac{\sigma_{i,j}^2 - v^2}{\sigma^2} \cdot [X_{i,j} - \mu_{i,j}]$$

and stored in the *pDst*. Here  $v^2$  is the noise variance, specified for each channel by the noise level parameter *noise*. If this parameter is not defined (*noise* = 0), then the function estimates the noise level by averaging through the image of all local variances  $\sigma_{i,j}$ , and stores the corresponding values in the *noise* for further use.

The function `ippiFilterWiener` uses the external work buffer `pBuffer`, which must be allocated before the function call. To determine the required buffer size, the function `ippiFilterWienerGetBufferSize` can be used.

[Figure “Applying the function `ippiFilterWiener`”](#) illustrates the result of using `ippiFilterWiener_32f_C1R` function.

### Applying the function `ippiFilterWiener`



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <code>maskSize</code> has a value less than or equal to 1.
<code>ippStsNoiseRangeErr</code>	Indicates an error condition if one of the <code>noise</code> values is less than 0 or greater than 1.

### Example

To better understand usage of this function, refer to the `FilterWiener.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement. For convenience, any digital image  $f$  is represented here as a matrix with  $M_f$  columns and  $N_f$  rows that contains pixel values  $f[i, j]$ ,  $0 \leq i < M_f$ ,  $0 \leq j < N_f$ .

## ConvGetBufferSize

*Computes the size of the work buffer for the ippiConv function.*

### Syntax

```
IppStatus ippiConvGetBufferSize (IppiSize src1Size, IppiSize src2Size, IppDataType
dataType, int numChannels, IppEnum algType, int* pBufferSize);
```

### Include Files

ippi.h

### Parameters

<i>src1Size, src2Size</i>	Size, in pixels, of the source images.
<i>dataType</i>	Data type for convolution. Possible values are <code>ipp32f</code> , <code>ipp16s</code> , or <code>ipp8u</code> .
<i>numChannels</i>	Number of image channels. Possible values are 1, 3, or 4.
<i>algType</i>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the <code>IppAlgType</code> and <code>IppiROIShape</code> values.
<i>pBufferSize</i>	Pointer to the size of the work buffer.

### Description

The `ippiConvGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional convolution. The result is stored in the `pBufferSize` parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsSizeErr</code>	Indicates an error when the <code>src1Size</code> or <code>src2Size</code> is negative, or equal to zero.
<code>ippStsNullChannelsErr</code>	Indicates an error when the <code>numChannels</code> value differs from 1, 3, or 4.
<code>ippStsDataTypeErr</code>	Indicates an error when the <code>dataType</code> value differs from the <code>ipp32f</code> , <code>ipp16s</code> , or <code>ipp8u</code> .
<code>ippStsAlgTypeErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between <code>algType</code> and <code>ippAlgMask</code> differs from the <code>ippAlgAuto</code>, <code>ippAlgDirect</code>, or <code>ippAlgFFT</code> values;</li> <li>• the result of the bitwise AND operation between <code>algType</code> and <code>ippiROIIMask</code> differs from the <code>ippiROIFull</code> or <code>ippiROIValid</code> values.</li> </ul>
<code>ippStsNullPtrErr</code>	Indicates an error when the <code>pBufferSize</code> is NULL.

### See Also

[Structures and Enumerators](#)

[Conv](#) Performs two-dimensional convolution of two images.

## Conv

*Performs two-dimensional convolution of two images.*

### Syntax

#### Case 1: Operating on integer data

```
IppStatusippiConv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, IppiSize src1Size,
const Ipp<datatype>* pSrc2, int src2Step, IppiSize src2Size, Ipp<datatype>* pDst, int
dstStep, int divisor, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for mod

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R

#### Case 2: Operating on floating-point data

```
IppStatusippiConv_<mod>(const Ipp32f* pSrc1, int src1Step, IppiSize src1Size, const
Ipp32f* pSrc2, int src2Step, IppiSize src2Size, Ipp32f* pDst, int dstStep, IppEnum
algType, Ipp8u* pBuffer);
```

Supported values for mod

32f_C1R
32f_C3R
32f_C4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distance, in bytes, between the starting points of consecutive lines in the source images.
<i>src1Size, src2Size</i>	Size in pixels of the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).
<i>algType</i>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType and IppiROIShape values.
<i>pBuffer</i>	Pointer to the buffer for internal calculations.

## Description

Before using this function, you need to compute the size of the work buffer using the `ippiConvGetBufferSize` function.

The `ippiConv` function operates with ROI. The type of convolution that function performs is defined by the value of the `algType` parameter:

1. If the `ippiROIFull` flag is set, the function performs full two-dimensional finite linear convolution between two source images pointed by the `pSrc1` and `pSrc2` parameters. The resulting destination image `h[i, j]` is computed by the following formula:

$$h[i, j] = \frac{1}{\text{divisor}} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k, l] \times g[i-k, j-l],$$

where

- $M_h = M_f + M_g - 1$

where

- $M_f$  is the number of rows in the first source image matrix  $f$

- $M_g$  is the number of rows in the second source image matrix  $g$

- $N_h = N_f + N_g - 1$

where

- $N_f$  is the number of columns in the first source image matrix  $f$

- $N_g$  is the number of columns in the second source image matrix  $g$

- $0 \leq i < M_h, 0 \leq l < N_h$

•

$$f[k, l] = \begin{cases} f[k, l], & 0 \leq k < M_f; \quad 0 \leq l < N_f \\ 0 & , \text{otherwise} \end{cases}$$

$$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \leq i-k < M_g; \quad 0 \leq j-l < N_g \\ 0 & , \text{otherwise} \end{cases}$$

2. If the `ippiROIValid` flag is set up, the function performs valid two-dimensional finite linear convolution between two source images pointed by the `pSrc1` and `pSrc2` parameters. The destination image `h[i, j]` obtained as a result of the function operation is computed by the following formula:

$$h[i, j] = \frac{1}{\text{divisor}} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1]$$

where

- $M_h = |M_f - M_g| + 1$

where

- $M_f$  is the number of rows in the first source image matrix  $f$

- $M_g$  is the number of rows in the second source image matrix  $g$

- $N_h = |N_f - N_g| + 1$

where

- $N_f$  is the number of columns in the first source image matrix  $f$
- $N_g$  is the number of columns in the second source image matrix  $g$
- $0 \leq i < M_h, 0 \leq j < N_h$

This case assumes that  $M_f \geq M_g$  and  $N_f \geq N_g$ . In case when  $M_f < M_g$  and  $N_f < N_g$ , the subscript index  $g$  in this equation must be replaced with the index  $f$ . For any other combination of source image sizes, the function performs no operation.

---

#### NOTE

The above formula provides the same result as in the case with the `ippiROIFull` flag, but produces only the part of the convolution image that is computed without zero-padded values.

---

Function flavors that accept input data of the `Ipp32f` type use the same summation formula, but without scaling of the result (*divisor* = 1 is assumed).

The following examples illustrate the function operation. For the source images  $f, g$  of size  $3 \times 5$  represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f$$

with  $g = f$ :

- for the `ippiROIFull` case, the resulting convolution image  $h$  is of size  $5 \times 9$  and contains the following data:

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

- for the `ippiROIValid` case, the resulting convolution image  $h$  is of size  $1 \times 1$  and contains the following data:

$$h = [11]$$

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>src1Size</i> or <i>src2Size</i> has a zero or negative value.
ippStsStepErr	Indicates an error when <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.
ippStsDivisorErr	Indicates an error when <i>divisor</i> has a zero value.
ippStsAlgTypeErr	Indicates an error when: <ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between the <i>algType</i> and <i>ippAlgMask</i> differs from the <i>ippAlgAuto</i>, <i>ippAlgDirect</i>, or <i>ippAlgFFT</i> values;</li> <li>• the result of the bitwise AND operation between the <i>algType</i> and <i>ippiROIIMask</i> differs from the <i>ippiROIFull</i> or <i>ippiROIValid</i> values.</li> </ul>

## Example

To better understand usage of this function, refer to the `Conv.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[Structures and Enumerators](#)

[ConvGetBufferSize](#) Computes the size of the work buffer for the `ippiConv` function.

# Deconvolution

---

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implement two methods: the Fourier deconvolution (noniterative method) [see for example, [\[Puettner05\]](#)], and the Richardson-Lucy method (iterative method) [\[Ric72\]](#). Border pixels of a source image are restored before deconvolution.

## DeconvFFTGetSize

Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

## Syntax

```
IppStatusippiDeconvFFTGetSize_32f(int nChannels, int kernelSize, int FFTorder, int* pSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>nChannels</i>	Number of channels in the image. Possible values are 1 or 3.
<i>kernelSize</i>	Size of the kernel.
<i>FFTorder</i>	Order of the created FFT state structure.
<i>pSize</i>	Pointer to the size of the <code>IppiDeconvFFTState_32f_C1R</code> or <code>IppiDeconvFFTState_32f_C3R</code> structure, in bytes.

## Description

This function computes the fast Fourier transform (FFT) deconvolution state structure size that is required to initialize the structure with the `ippiDeconvFFTInit` function. This structure is used by the `ippiDeconvFFT` function, which performs deconvolution of the source image using FFT.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>nChannels</i> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when <i>kernelSize</i> is less than, or equal to 0; or if <i>kernelSize</i> is greater than $2^{FFTorder}$ .

## See Also

`DeconvFFTInit` Initializes the FFT deconvolution state structure.

`DeconvFFT` Performs FFT deconvolution of an image.

## DeconvFFTInit

*Initializes the FFT deconvolution state structure.*

### Syntax

```
IppStatus ippiDeconvFFTInit_32f_C1R(IppiDeconvFFTState_32f_C1R* pDeconvFFTState, const
Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);
IppStatus ippiDeconvFFTInit_32f_C3R(IppiDeconvFFTState_32f_C3R* pDeconvFFTState, const
Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pDeconvFFTState</i>	Pointer to the FFT deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.

<i>FFTorder</i>	Order of the created FFT state structure.
<i>threshold</i>	Value of the threshold level (to exclude dividing by zero).

## Description

This function initializes the FFT deconvolution state structure that is used by the [ippiDeconvFFT](#) function to perform deconvolution of the source image using FFT. Before using the [ippiDeconvFFTInit](#) function, compute the size of the structure using the [ippiDeconvFFTGetSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>kernelSize</code> is less than, or equal to 0; or if <code>kernelSize</code> is greater than $2^{FFTorder}$ .
<code>ippStsBadArgErr</code>	Indicates an error when <code>threshold</code> is less than, or equal to 0.

## See Also

[DeconvFFTGetSize](#) Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

[DeconvFFT](#) Performs FFT deconvolution of an image.

## DeconvFFT

*Performs FFT deconvolution of an image.*

---

## Syntax

```
IppStatusippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R* pDeconvFFTState);
IppStatusippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R* pDeconvFFTState);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.

*pDeconvFFTState* Pointer to the FFT deconvolution state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs deconvolution of the source image *pSrc* using FFT with parameters specified in the FFT deconvolution state structure *pDeconvFFTState* and stores results to the destination image *pDst*. The FFT deconvolution state structure must be initialized by calling the function [DeconvFFTInit](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

## DeconvLRGetSize

*Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.*

## Syntax

```
IppStatusippiDeconvLRGetSize_32f(int numChannels, int kernelSize, IppiSize maxRoi,
int* pSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>numChannels</i>	Number of channels in the image. Possible values are 1 or 3.
<i>kernelSize</i>	Size of the kernel.
<i>maxRoi</i>	Maximum size of the image ROI, in pixels.
<i>pSize</i>	Pointer to the size of the <code>IppiDeconvLRState_32f_C1R</code> or <code>IppiDeconvLRState_32f_C3R</code> structure, in bytes.

## Description

This function computes the Lucy-Richardson (LR) deconvolution state structure size that is required to initialize the structure with the [ippiDeconvLRInit](#) function. This structure is used by the [ippiDeconvLR](#) function, which performs LR deconvolution of the source image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .

ippStsSizeErr	<p>Indicates an error when:</p> <ul style="list-style-type: none"> <li><i>kernelSize</i> is less than, or equal to 0</li> <li><i>kernelSize</i> is greater than <i>maxRoi.height</i> or <i>maxRoi.width</i></li> <li><i>maxRoi.height</i> or <i>maxRoi.width</i> is less than, or equal to zero</li> </ul>
---------------	--

## See Also

[DeconvLRInit](#) Initializes the LR deconvolution state structure.

[DeconvLR](#) Performs LR deconvolution of an image.

## DeconvLRInit

*Initializes the LR deconvolution state structure.*

---

### Syntax

```
IppStatusippiDeconvLRInit_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```

```
IppStatusippiDeconvLRInit_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.
<i>maxRoi</i>	Maximum size of the image ROI, in pixels.
<i>threshold</i>	Value of the threshold level (to exclude dividing by zero).

### Description

This function initializes the LR deconvolution state structure that is used by the [ippiDeconvLR](#) function to perform LR deconvolution of the source image. Before using the [ippiDeconvLRInit](#) function, compute the size of the structure using the [ippiDeconvLRGetSize](#) function.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	<p>Indicates an error when:</p> <ul style="list-style-type: none"> <li><i>kernelSize</i> is less than, or equal to 0</li> <li><i>kernelSize</i> is greater than <i>maxRoi.height</i> or <i>maxRoi.width</i></li> </ul>

- *maxRoi.height* or *maxRoi.width* is less than, or equal to zero
- `ippStsBadArgErr` Indicates an error when *threshold* is less than, or equal to 0.

## See Also

[DeconvLRGetSize](#) Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.

[DeconvLR](#) Performs LR deconvolution of an image.

## DeconvLR

*Performs LR deconvolution of an image.*

### Syntax

```
IppStatusippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, int numIter, IppiDeconvLR_32f_C1R* pDeconvLR);
IppStatusippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, int numIter, IppiDeconvLR_32f_C3R* pDeconvLR);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>numIter</i>	Number of algorithm iterations.
<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs deconvolution of the source image *pSrc* using the Lucy-Richardson algorithm with parameters specified in the state structure *pDeconvLR* and stores results to the destination image *pDst*. The Lucy-Richardson deconvolution state structure must be initialized by calling the function [DeconvLRIInit](#) beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> is less than or equal to 0, or if <i>roi.width</i> is greater than ( <i>maxRoi.width - kernelSize</i> ), or <i>roi.height</i> is greater than ( <i>maxRoi.height - kernelSize</i> ).
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.
ippStsBadArgErr	Indicates an error condition if <i>numIter</i> is less than or equal to 0.

## Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels. The supported fixed filters and their respective kernel sizes are listed in the following table:

### Types of the Fixed Filter Functions

Fixed Filter Type	Kernel Size
Horizontal Prewitt operator	3x3
Vertical Prewitt operator	3x3
Horizontal Scharr operator	3x3
Vertical Scharr operator	3x3
Horizontal Sobel operator	3x3 or 5x5
Vertical Sobel operator	3x3 or 5x5
Second derivative horizontal Sobel operator	3x3 or 5x5
Second derivative vertical Sobel operator	3x3 or 5x5
Second cross derivative Sobel operator	3x3 or 5x5
Horizontal Roberts operator	3x3
Vertical Roberts operator	3x3
Laplacian highpass filter	3x3 or 5x5
Gaussian lowpass filter	3x3 or 5x5
Highpass filter	3x3 or 5x5
Lowpass filter	3x3 or 5x5
Sharpening filter	3x3

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.

---

**NOTE**

The anchor cell is the center cell of the kernel for all fixed filters.

---

## FilterGaussianGetBufferSize

*Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.*

### Syntax

```
IppStatusippiFilterGaussianGetBufferSize(IppiSize maxRoiSize, Ipp32u kernelSize,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

### Platform-aware functions

```
IppStatusippiFilterGaussianGetBufferSize_L(IppiSizeL maxRoiSize, int kernelSize,
IppDataType dataType, IppiBorderType borderType, int numChannels, IppSizeL*
pBufferSize);
```

### Include Files

ippcv.h

Flavors with the \_L suffix: ippcv\_1.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>maxRoiSize</i>	Maximal size of the image ROI in pixels.
<i>kernelSize</i>	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
<i>dataType</i>	Data type of the source and destination images.
<i>borderType</i>	One of border supported types.
<i>numChannels</i>	Number of channels in the images. Possible values are 1 and 3.
<i>pSpecSize</i>	Pointer to the computed size (in bytes) of the Gaussian specification structure.
<i>pBufferSize</i>	Pointer to the computed size (in bytes) of the external buffer.

### Description

This function computes the size of the Gaussian context structure and external work buffer for the [FilterGaussianBorder](#) function or for the platform-aware function [FilterGaussian](#). The results are stored in *pSpecSize* and *pBufferSize*. The buffer with the length *pBufferSize[0]* can be used to filter an image with the width and height less than, or equal to the corresponding fields of *maxRoiSize*, and/or kernel size that is less than, or equal to *kernelSize*.

### NOTE

The platform-aware function [FilterGaussianGetBufferSize\\_L](#) computes only the size of the external work buffer for Gaussian filtering with user-defined borders. To compute the size of the Gaussian specification structure, please use the platform-aware function [FilterGaussianGetSpecSize](#).

Use the computed *pBufferSize* and *pSpecSize* values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the [FilterGaussianBorder](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>maxRoiSize</code> has a field with a zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error when <code>kernelSize</code> is even, or less than 3.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsNumChannelErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[FilterGaussianBorder](#) Performs Gaussian filtering of an image with user-defined borders.

## FilterGaussianGetSpecSize

*Computes the size of the Gaussian specification structure.*

---

## Syntax

```
IppStatusippiFilterGaussianGetSpecSize(int kernelSize, IppDataType dataType, int numChannels, int* pSpecSize, int* pInitBufferSize);
IppStatusippiFilterGaussianGetSpecSize_L(int kernelSize, IppDataType dataType, int numChannels, IppSizeL* pSpecSize, IppSizeL* pInitBufferSize);
```

## Include Files

`ippcv.h`  
`ippcv_1.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>kernelSize</code>	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
<code>dataType</code>	Data type of the source and destination images.
<code>numChannels</code>	Number of channels in the images. Possible values are 1 and 3.
<code>pSpecSize</code>	Pointer to the computed size (in bytes) of the Gaussian specification structure.

*pInitBufferSize* Pointer to the computed size (in bytes) of the external buffer.

## Description

This function computes the size of the Gaussian context structure and the size of the initialization external work buffer for the `FilterGaussianBorder` function or for the platform-aware functions `FilterGaussianInit` and `FilterGaussian`. The results are stored in *pSpecSize* and *pInitBufferSize*. The buffer with the length *pInitBufferSize[0]* can be used to initialize the specification structure for the Gaussian filter.

Use the computed *pInitBufferSize* and *pSpecSize* values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsBadArgErr</code>	Indicates an error when <code>kernelSize</code> is even, or less than 3.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsNumChannelErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

`FilterGaussian` Performs Gaussian filtering of an image with user-defined borders.

## FilterGaussianInit

*Initializes the Gaussian context structure.*

## Syntax

```
IppStatusippiFilterGaussianInit(IppiSize roiSize, Ipp32u kernelSize, Ipp32f sigma,
IppiBorderType borderType, IppDataType dataType, int numChannels,
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

## Platform-aware functions

```
IppStatusippiFilterGaussianInit_L(IppiSizeL roiSize, int kernelSize, Ipp32f sigma,
IppiBorderType borderType, IppDataType dataType, int numChannels,
IppFilterGaussianSpec* pSpec, Ipp8u* pInitBuffer);
```

## Include Files

`ippcv.h`

Flavors with the `_L` suffix: `ippcv_l.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Size of the image ROI in pixels.
<i>kernelSize</i>	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
<i>sigma</i>	Standard deviation of the Gaussian kernel.
<i>borderType</i>	Type of border. Possible values are:
	ippBorderConst Values of all border pixels are set to constant.
	ippBorderRepl Border is replicated from the edge pixels.
	ippBorderInMem Border is obtained from the image pixels in memory.
	ippBorderMirro r Border pixels are mirrored from the source image boundary pixels.
<i>dataType</i>	Data type of the source and destination images.
<i>numChannels</i>	Number of channels in the images. Possible values are 1 and 3.
<i>pSpec</i>	Pointer to the Gaussian specification structure.
<i>pBuffer</i> , <i>pInitBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the Gaussian specification structure *pSpec* for the [FilterGaussianBorder](#) function or for the platform-aware function [FilterGaussian](#). Before using this function, compute the size of the specification structure and the external buffer using the [FilterGaussianGetBufferSize](#) function and [FilterGaussianGetSpecSize](#) for the platform-aware function.

The *roiSize* and *kernelSize* values must be less than, or equal to the corresponding values specified in the [FilterGaussianGetBufferSize](#) function.

The kernel of the Gaussian filter is the matrix of size *kernelSize*×*kernelSize* with the standard deviation *sigma*. The values of the Gaussian kernel elements are calculated by the formula below and then normalized:

$$G(i, j) = \exp \left\{ -\frac{(K/2 - i)^2 + (K/2 - j)^2}{2\sigma^2} \right\}, \quad i, j = 0, \dots, K$$

The anchor cell is the center of the kernel.

For an example on how to use this function, refer to the example provided with the [FilterGaussianBorder](#) function description.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
--------------------	---

---

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
ippStsBadArgErr	Indicates an error when <i>kernelSize</i> is even, or less than 3; or <i>sigma</i> is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsChannelErr	Indicates an error when <i>numChannels</i> has an illegal value.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[FilterGaussianGetBufferSize](#) Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

[FilterGaussianBorder](#) Performs Gaussian filtering of an image with user-defined borders.

## FilterGaussian

*Performs Gaussian filtering of an image with user-defined borders.*

---

### Syntax

#### Case 1: Operating on one-channel data

```
IppStatusippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[1], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operating on multi-channel data

```
IppStatusippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

#### Case 3: Operating on one-channel data with platform-aware functions

```
IppStatusippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[1], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32f_C1R_L
----------	-----------	-----------	-----------

**Case 4: Operating on multi-channel data with platform-aware functions**

```
IppStatusippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R_L	16u_C3R_L	16s_C3R_L	32f_C3R_L
----------	-----------	-----------	-----------

**Include Files**

`ippcv.h`

`ippcv_l.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI, in pixels.
<code>borderType</code>	One of the supported border types.
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pSpec</code>	Pointer to the Gaussian specification structure.
<code>pBuffer</code>	Pointer to the work buffer.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

These functions apply the Gaussian filter to the source image ROI `pSrc`. The kernel of the Gaussian filter is the matrix of size `kernelSize`x`kernelSize` with the standard deviation `sigma`. The values of the Gaussian kernel elements are computed by the [FilterGaussianInit](#) function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.

Before using the `ippiFilterGaussian` function, compute the size of the Gaussian specification structure using the [FilterGaussianGetSpecSize](#) function and the external buffer using the [FilterGaussianGetBufferSize](#) function and initialize the structure using the [FilterGaussianInit](#) function.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width*pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by <i>sizeof(Ipp&lt;dataType&gt;)</i> .
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsBadArgErr	Indicates an error when <i>kernelSize</i> is even, or less than 3.

## Example

To better understand usage of this function, refer to the `FilterGaussianBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterGaussianGetSpecSize](#) Computes the size of the Gaussian specification structure.

[FilterGaussianGetBufferSize](#) Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

[FilterGaussianInit](#) Initializes the Gaussian context structure.

## FilterHipassBorderGetBufferSize

*Computes the size of the work buffer for high-pass filtering with the `ippiFilterHipassBorder` function.*

## Syntax

```
IppStatus ippiFilterHipassBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible values are: <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.

*pBufferSize*      Pointer to the size, in bytes, of the external buffer.

## Description

This function computes the size, in bytes, of the external work buffer for the [ippiFilterHipassBorder](#) function. The result is stored in the *pBufferSize* parameter.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
ippStsNumChannelErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterHipassBorder](#) Filters an image using a high-pass filter.

## FilterHipassBorder

*Filters an image using a high-pass filter.*

---

## Syntax

### Case 1: Operating on one-channel data

```
IppStatusippiFilterHipassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp16s
borderValue, Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);
```

### Case 2: Operating on multi-channel data

```
IppStatusippiFilterHipassBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```

IppStatusippiFilterHipassBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);

```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.						
<i>dstRoiSize</i>	Size of the source and destination ROI, in pixels.						
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> . Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .						
<i>borderType</i>	Type of border. Possible values are: <table> <tr> <td><i>ippBorderConst</i></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><i>ippBorderRepl</i></td><td>Border is replicated from the edge pixels.</td></tr> <tr> <td><i>ippBorderInMem</i></td><td>Border is obtained from the source image pixels in memory.</td></tr> </table>	<i>ippBorderConst</i>	Values of all border pixels are set to constant.	<i>ippBorderRepl</i>	Border is replicated from the edge pixels.	<i>ippBorderInMem</i>	Border is obtained from the source image pixels in memory.
<i>ippBorderConst</i>	Values of all border pixels are set to constant.						
<i>ippBorderRepl</i>	Border is replicated from the edge pixels.						
<i>ippBorderInMem</i>	Border is obtained from the source image pixels in memory.						

Mixed borders are also supported. They can be obtained by the bitwise operation `OR` between any of the `ippBorderRepl`, `ippBorderConst`, or `ippBorderMirror` values and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values.

*borderValue*

Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

*pBorderValue[3], pBorderValue[4]*

Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

*pBuffer*

Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterHipassBorderGetBufferSize` function.

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies a high-pass filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the *mask* value. The kernels have the following values:

		-1	-1	-1	-1	-1		
-1	-1	-1	-1	-1	-1	-1		
-1	8	-1	or	-1	-1	24	-1	-1
-1	-1	-1		-1	-1	-1	-1	-1
				-1	-1	-1	-1	-1

The anchor cell is the center cell of the kernel, highlighted in red.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsBorderErr</code>	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

`FilterHipassBorderGetBufferSize` Computes the size of the work buffer for high-pass filtering with the `ippiFilterHipassBorder` function.

## FilterLaplaceBorderGetBufferSize

*Computes the size of the work buffer for Laplace filtering.*

### Syntax

```
IppStatusippiFilterLaplaceBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible values are: <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external buffer.

### Description

This function computes the size, in bytes, of the external work buffer for the `ippiFilterLaplaceBorder` function. The result is stored in the `pBufferSize` parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>mask</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>srcDataType</code> or <code>dstDataType</code> has an illegal value.
<code>ippStsNumChannelErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

### See Also

[FilterLaplaceBorder](#) Filters an image using a Laplace filter.

## FilterLaplaceBorder

*Filters an image using a Laplace filter.*

---

### Syntax

#### Case 1: Operating on one-channel data

```
IppStatusippiFilterLaplaceBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp16s borderValue, Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

#### Case 2: Operating on multi-channel data

```
IppStatusippiFilterLaplaceBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterLaplaceBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

### Include Files

ippi.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI, in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> . Possible values are <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<code>borderType</code>	Type of border. Possible values are:  <code>ippBorderConst</code> Values of all border pixels are set to constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , or <code>ippBorderMirror</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.  <code>borderValue</code> Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  <code>pBorderValue[3], pBorderValue[4]</code> Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  <code>pBuffer</code> Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `ippiFilterLaplaceBorderGetBufferSize` function.

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies a Laplace filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the *mask* value. The kernels have the following values:

		-1	-3	-4	-3	-1		
-1	-1	-1		-3	0	6	0	-3
-1	8	-1	or	-4	6	20	6	-4
-1	-1	-1		-3	0	6	0	-3
				-1	-3	-4	-3	-1

The anchor cell is the center cell of the kernel, highlighted in red.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsBorderErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterLaplaceBorder](#) Filters an image using a Laplace filter.

## FilterLaplacianGetBufferSize

*Computes the size of the external buffer for the Laplace filter with border.*

---

### Syntax

```
IppStatusippiFilterLaplacianGetBufferSize_<mod>(IppiSizeroiSize, IppiMaskSizemask,  
int*pBufferSize);
```

Supported values for *mod*:

8u16s\_C1R    32f\_C1R

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

This function computes the size of the external buffer that is required for the filter function `ippiFilterLaplacianBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterLaplacianBorder

*Applies Laplacian filter with border.*

### Syntax

```
IppStatusippiFilterLaplacianBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R`    `32f_C1R`

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>mask</i>	Type of the filter kernel.
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible:
	ippBorderConst    Values of all border pixels are set to constant.
	ippBorderRepl    Replicated border is used.
	ippBorderMirro r    Border pixels are mirrored from the source image boundary pixels.
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the laplacian filter to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

			2	4	4	4	2	
2	0	2		4	0	-8	0	4
0	-8	0	or	4	-8	-24	-8	4
2	0	2		4	0	-8	0	4
				2	4	4	4	2

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterLaplacianGetBufferSize](#) beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

---

ippStsBadArgErr	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
ippStsMaskErr	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterLowpassGetBufferSize

Computes the size of the external buffer for the lowpass filter with border.

---

### Syntax

```
IppStatusippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize, IppiMaskSize mask,  
int* pBufferSize);  
  
IppStatusippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize, IppiMaskSize mask,  
int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of IppiMaskSize type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

This function computes the size of the external buffer that is required for the filter function `ippiFilterLowpassBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the <i>pBufferSize</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterLowpassBorder

Applies lowpass filter with border.

---

## Syntax

```
IppStatusippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);

IppStatusippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pDst</i>	Pointer to the destination image ROI.				
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>mask</i>	Type of the filter kernel.				
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); the following values are possible: <table> <tr> <td>ippBorderConst</td> <td>Values of all border pixels are set to constant.</td> </tr> <tr> <td>ippBorderRepl</td> <td>Replicated border is used.</td> </tr> </table>	ippBorderConst	Values of all border pixels are set to constant.	ippBorderRepl	Replicated border is used.
ippBorderConst	Values of all border pixels are set to constant.				
ippBorderRepl	Replicated border is used.				
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).				
<i>pBuffer</i>	Pointer to the working buffer.				

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the lowpass filter (blur operation) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

The 5x5 filter uses the kernel with the following values:

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterLowpassGetBufferSize](#) beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if <i>borderType</i> has a wrong value.
ippStsMaskErr	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterPrewittHorizBorderGetBufferSize

Computes the size of the work buffer for the Prewitt Horizontal filter.

### Syntax

```
IppStatusippiFilterPrewittHorizBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

### Include Files

ippi.h

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
-------------------	--

<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterPrewittHorizBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterPrewittHorizBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterPrewittHorizBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterPrewittHorizBorder](#) Filters and image using a horizontal Prewitt filter.

## FilterPrewittHorizBorder

*Filters and image using a horizontal Prewitt filter.*

### Syntax

```
IppStatusippiFilterPrewittHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R 16s_C1R 32f_C1R`

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderMirror    Mirrored border is used. r ippBorderMirrorR    Mirrored border with replication is used. ippBorderInMem    Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values, and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `FilterPrewittHorizBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Prewitt filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

1	1	1
0	0	0
-1	-1	-1

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilteringConvolutionKernels.c` and `FilterPrewittHorizBorder.c` examples in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterPrewittHorizBorderGetBufferSize](#) Computes the size of the work buffer for the Prewitt Horizontal filter.

## FilterPrewittVertBorderGetBufferSize

*Computes the size of the work buffer for the Prewitt Vertical filter.*

---

## Syntax

```
IppStatusippiFilterPrewittVertBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

## Include Files

`ippi.h`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterPrewittVertBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterPrewittVertBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterPrewittVertBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterPrewittVertBorder](#) Filters and image using a vertical Prewitt kernel.

## FilterPrewittVertBorder

*Filters and image using a vertical Prewitt kernel.*

### Syntax

```
IppStatus ippiFilterPrewittVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R 16s_C1R 32f_C1R`

### Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<code>borderType</code>	Type of border. Possible values are:  <code>ippBorderConst</code> Values of all border pixels are set to constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderMirror</code> Mirrored border is used. <code>r</code> <code>ippBorderMirrorR</code> Mirrored border with replication is used. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterPrewittVertBorderGetBufferSize` function.

This function operates with ROI.

This function applies a vertical Prewitt filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

-1	0	1
-1	0	1
-1	0	1

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilteringConvolutionKernels.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterPrewittVertBorderGetBufferSize](#) Computes the size of the work buffer for the Prewitt Vertical filter.

## FilterRobertsUpBorderGetBufferSize

*Computes the size of the work buffer for the vertical Roberts edge filter.*

## Syntax

```
IppStatusippiFilterRobertsUpBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize  
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*  
pBufferSize);
```

## Include Files

`ippi.h`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterRobertsUpBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterRobertsUpBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterRobertsUpBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>dstRoiSize</code> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>mask</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>srcDataType</code> or <code>dstDataType</code> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[FilterRobertsUpBorder](#) Filters an image using a vertical Roberts edge filter.

## FilterRobertsUpBorder

[Filters an image using a vertical Roberts edge filter.](#)

## Syntax

```
IppStatus ippiFilterRobertsUpBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R 16s_C1R 32f_C1R`

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst    Values of all border pixels are set to constant. ippBorderRepl    Border is replicated from the edge pixels. ippBorderMirro r                Mirrored border is used.  ippBorderMirro rR                Mirrored border with replication is used.  ippBorderInMem    Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.  <i>borderValue</i> Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  <i>pBuffer</i> Pointer to the work buffer.

## Description

This function operates with ROI.

This function applies a vertical Roberts edge filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

0	0	0
0	1	0
-1	0	0

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the vertical direction.

Before using this function, you need to compute the size of the work buffer *pBuffer* using the *FilterRobertsUpBorderGetBufferSize* function.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the *FilteringConvolutionKernels.c* example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[\*\*FilterRobertsUpBorderGetBufferSize\*\*](#) Computes the size of the work buffer for the vertical Roberts edge filter.

## FilterRobertsDownBorderGetBufferSize

*Computes the size of the work buffer for the horizontal Roberts edge filter.*

---

## Syntax

```
IppStatusippiFilterRobertsDownBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

## Include Files

`ippi.h`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterRobertsDownBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterRobertsDownBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterRobertsDownBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

**FilterRobertsDownBorder** Filters an image using a horizontal Roberts edge filter.

## FilterRobertsDownBorder

Filters an image using a horizontal Roberts edge filter.

### Syntax

```
IppStatusippiFilterRobertsDownBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

`8u16s_C1R 16s_C1R 32f_C1R`

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl        Border is replicated from the edge pixels. ippBorderMirror     Mirrored border is used. r ippBorderMirrorR   Mirrored border with replication is used. ippBorderInMem      Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values, and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `FilterRobertsDownBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Roberts edge filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

0	0	0
0	1	0
0	0	-1

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the horizontal direction.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilteringConvolutionKernels.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterRobertsDownBorderGetBufferSize](#) Computes the size of the work buffer for the horizontal Roberts edge filter.

## FilterScharrHorizMaskBorderGetBufferSize

*Computes the size of the work buffer for the Scharr Horizontal filter.*

## Syntax

```
IppStatusippiFilterScharrHorizMaskBorderGetBufferSize(IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

## Include Files

`ippi.h`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterScharrHorizMaskBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterScharrHorizMaskBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterScharrHorizMaskBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterScharrHorizMaskBorder](#) Filters an image using a horizontal Scharr filter.

## FilterScharrHorizMaskBorder

*Filters an image using a horizontal Scharr filter.*

## Syntax

```
IppStatus ippiFilterScharrHorizMaskBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R 16s_C1R 32f_C1R`

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<code>borderType</code>	Type of border. Possible values are:  <code>ippBorderConst</code> Values of all border pixels are set to constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderMirro</code> r Mirrored border is used. <code>ippBorderMirro</code> r <code>R</code> Mirrored border with replication is used. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterScharrHorizMaskBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Scharr filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

3	10	3
0	0	0
-3	-10	-3

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

To better understand usage of this function, refer to the `FilteringConvolutionKernels.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterPrewittHorizBorderGetBufferSize](#) Computes the size of the work buffer for the Prewitt Horizontal filter.

## FilterScharrVertMaskBorderGetBufferSize

*Computes the size of the work buffer for the Scharr Vertical filter.*

## Syntax

```
IppStatusippiFilterScharrVertMaskBorderGetBufferSize(IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

## Include Files

`ippi.h`

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterScharrVertMaskBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterScharrVertMaskBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterScharrVertMaskBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterScharrVertMaskBorder](#) Filters an image using a vertical Scharr kernel.

## FilterScharrVertMaskBorder

*Filters an image using a vertical Scharr kernel.*

## Syntax

```
IppStatus ippiFilterScharrVertMaskBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R 16s_C1R 32f_C1R`

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between starting points of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .
<code>borderType</code>	Type of border. Possible values are:  <code>ippBorderConst</code> Values of all border pixels are set to constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderMirro r</code> Mirrored border is used. <code>ippBorderMirro rR</code> Mirrored border with replication is used. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.  <code>borderValue</code> Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.  <code>pBuffer</code> Pointer to the work buffer.

## Description

This function operates with ROI.

This function applies a vertical Scharr filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

-3	0	3
-10	0	10
-3	0	3

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterScharrVertMaskBorderGetBufferSize` function.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiFilterScharrVertMaskBorderGetBufferSize` and `ippiFilterScharrVertMaskBorder_8u16s_C1R` functions.

```
IppStatus fix_scharrvert_8u16( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s pDst[8*7];
    Ipp8u *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int srcStep = 9 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp16s);
    int bufferSize;
    IppStatus status;
   ippiFilterScharrVertMaskBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
```

```

 pBuffer = ippsMalloc_8u(bufferSize);
 status =ippiFilterScharrVertMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
            borderType, 0, pBuffer);
 ippsFree(pBuffer);
 return status;
}

```

The result is as follows:

```

pDst -->
 16 32 1925 1925 32 -1117 -1127 22
 16 32 1995 1995 32 -1094 -1097 29
 16 32 2016 2016 32 -1088 -1088 32
 10 20 2019 2019 20 -1082 -1082 20
-10 -20 2029 2029 -20 -1062 -1062 -20
-16 -32 2032 2032 -32 -1056 -1056 -32
-16 -32 2032 2032 -32 -1056 -1056 -32

```

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterScharrVertMaskBorderGetBufferSize](#) Computes the size of the work buffer for the Scharr Vertical filter.

## FilterSharpenBorderGetBufferSize

*Computes the size of the work buffer for image sharpening.*

### Syntax

```
IppStatusippiFilterSharpenBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible value is <code>ippMskSize3x3</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible values are 1, 3, or 4.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external buffer.

## Description

This function computes the size, in bytes, of the external work buffer for the `ippiFilterSharpenBorder` function. The result is stored in the `pBufferSize` parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>mask</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>srcDataType</code> or <code>dstDataType</code> has an illegal value.
<code>ippStsNumChannelErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[FilterSharpenBorder](#) Performs image sharpening with a high-pass filter.

## FilterSharpenBorder

Performs image sharpening with a high-pass filter.

## Syntax

### Case 1: Operating on one-channel data

```
IppStatus ippiFilterSharpenBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

### Case 2: Operating on multi-channel data

```
IppStatus ippiFilterSharpenBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatusippiFilterSharpenBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterSharpenBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterSharpenBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterSharpenBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterSharpenBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h, ippvm.h, ipps.h

Libraries:ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.						
<i>dstRoiSize</i>	Size of the source and destination ROI, in pixels.						
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> . Possible value is <code>ippMskSize3x3</code> .						
<i>borderType</i>	Type of border. Possible values are: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Border is replicated from the edge pixels.</td></tr> <tr> <td><code>ippBorderInMem</code></td><td>Border is obtained from the source image pixels in memory.</td></tr> </table> Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , or <code>ippBorderMirror</code> values and the	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Border is replicated from the edge pixels.	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
<code>ippBorderConst</code>	Values of all border pixels are set to constant.						
<code>ippBorderRepl</code>	Border is replicated from the edge pixels.						
<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.						

	ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBorderValue[3], pBorderValue[4]</i>	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `ippiFilterSharpenBorderGetBufferSize` function.

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies a high-pass filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 size. The kernel has the following value:

-1/8	-1/8	-1/8
-1/8	16/8	-1/8
-1/8	-1/8	-1/8

The anchor cell is the center cell of the kernel, highlighted in red.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsBorderErr</code>	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterSharpenBorder](#) Performs image sharpening with a high-pass filter.

## FilterSobelGetBufferSize

Computes the size of the work buffer for the Sobel filter.

## Syntax

```
IppStatusippiFilterSobelGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppNormType normType, IppDataType srcDataType, IppDataType dstDataType, int
numChannels, int* pBufferSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <a href="#">IppiMaskSize</a> type. Possible values are <a href="#">ippMskSize3x3</a> or <a href="#">ippMskSize5x5</a> .
<i>normType</i>	Normalization mode of <a href="#">IppNormType</a> type.
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The [ippiFilterSobel](#) function computes the size, in bytes, of the external work buffer needed for the [ippiFilterSobel](#) function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this functions, see the code example provided with the [ippiFilterSobel](#) function description.

## Return Values

<a href="#">ippStsNoErr</a>	Indicates no error.
<a href="#">ippStsNullPtrErr</a>	Indicates an error when <i>pBufferSize</i> is NULL.
<a href="#">ippStsSizeErr</a>	Indicates an error when <i>dstRoiSize</i> is less than, or equal to zero.
<a href="#">ippStsMaskSizeErr</a>	Indicates an error when <i>mask</i> has an illegal value.
<a href="#">ippStsBadArgErr</a>	Indicates an error when <i>normType</i> has an illegal value.
<a href="#">ippStsDataTypeErr</a>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<a href="#">ippStsNumChannelsError</a>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobel](#) Filters an image using a Sobel filter.

## FilterSobelInit

*Filters an image using a Sobel filter.*

### Syntax

```
IppStatusippiFilterSobel_<mod>_T(IppiSize roiSize, IppiMaskSize maskId, IppNormType  
normType, IppDataType srcDataType, IppDataType dstDataType, int numChannels,  
IppiFilterSobelSpec_T* pSpec);
```

Supported values for *mod*:

8u16s\_C1R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>roiSize</i>	Size of the destination ROI in pixels.
<i>maskId</i>	Predefined mask of the IppiMaskSize type.
<i>normType</i>	Normalization mode if the IppNormType type is specified.
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image.
<i>pSpec</i>	Pointer to the Filter Sobel specification structure.

### Description

This function is used for initialization of the *pSpec* structure for the `ippiFilterSobel` function.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> is negative or equal to zero.
ippStsMaskSizeErr	Indicates an error when <i>maskSize</i> has a wrong value.
ippStsDataTypeErr	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

### See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

**FilterSobelGetBufferSize** Computes the size of the work buffer for the Sobel filter.

## FilterSobel

*Filters an image using a Sobel filter.*

---

### Syntax

```
IppStatusippiFilterSobel_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize maskSize,
IppNormType normType, IppiBorderType borderType, Ipp<srcdatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s\_C1R 16s32f\_C1R 16u32f\_C1R 32f\_C1R

```
IppStatusippiFilterSobel_<mod>_T(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, int
dstStep, IppiBorderType border, const Ipp8u borderValue, IppiFilterSobelSpec_T* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s\_C1R\_T

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the predefined mask. Possible values are <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<i>normType</i>	Normalization mode of <code>IppNormType</code> .
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl        Border is replicated from the edge pixels.

`ippBorderInMem` Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation `OR` between any of the `ippBorderRepl` or `ippBorderConst` values and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values.

`borderValue`

Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

`pBuffer`

Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `ippiFilterSobelBufferSize` function.

Call the `ippiFilterSobelInit_T` function before using the `ippiFilterSobel*_T` function.

This function applies a Sobel filter to the source image with the specified kernel size and normalization type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the `maskSize` value. The formulas below describe the algorithm for the 3x3 and 5x5 Sobel operators.

3x3 Sobel operator:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

5x5 Sobel operator:

$$G_x = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 2 & 8 & 12 & 8 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A$$

where

- `A` is the source image
- `*` is the 2D convolution operator
- $G_x$  and  $G_y$  are horizontal and vertical magnitude of the source image, respectively

Sobel filter output  $G$ , as overall gradient magnitude, is generated through L1 and L2 normalization of  $G_x$  and  $G_y$ .

L1 normalization:

$$\mathbf{G} = |\mathbf{G}_x| + |\mathbf{G}_y|$$

L2 normalization:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when <i>maskSize</i> has an illegal value.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiFilterSobelGetBufferSize` and `ippiFilterSobel_8u16s_C1R` functions.

```
IppStatus filter_sobel_8u16s_c1( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s pDst[8*7];
    IppiSize roiSize = {8, 7};
    IppiMaskSize mask = ippMskSize3x3;
    IppiBorderType borderType = ippBorderConst | ippBorderInMemTop | ippBorderInMemRight;
    int srcStep = 9 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp16s);
    int bufferSize;
    IppStatus status;
    Ipp8u *pBuffer;
    IppNormType normType = ippNormL1;

    ippFilterSobelGetBufferSize(roiSize, mask, normType, ipp8u, ipp16s, 1, &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippFilterSobel_8u16s_C1R(pSrc+srcStep, srcStep, pDst, dstStep, roiSize, mask,
```

```

normType, borderType, 33, pBuffer);
    ippsFree(pBuffer);

    return status;
}

```

The result is as follows:

pDst -->							
132	20	502	516	48	322	330	58
126	20	516	530	48	316	322	58
118	16	512	512	16	280	280	16
118	12	510	512	8	272	276	8
110	4	510	508	8	272	268	8
114	16	516	516	16	272	272	16
162	114	398	652	402	526	394	134

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterSobelGetBufferSize](#) Computes the size of the work buffer for the Sobel filter.

## FilterSobelHorizBorderGetBufferSize

*Computes the size of the work buffer for the Sobel Horizontal filter.*

### Syntax

```

IppStatusippiFilterSobelHorizBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);

IppStatusippiFilterSobelHorizBorderGetBufferSize_T(IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);

```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type. Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.

<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterSobelHorizBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelHorizBorder` function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this functions, see the code example provided with the `ippiFilterSobelHorizBorder` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobelHorizBorder](#) Filters an image using a horizontal Sobel filter.

## FilterSobelHorizBorder

*Filters an image using a horizontal Sobel filter.*

### Syntax

```
IppStatus ippiFilterSobelHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R 16s_C1R 32f_C1R`

```
IppStatus ippiFilterSobelHorizBorder_<mod>_T(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R_T`

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> . Possible values are <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl        Border is replicated from the edge pixels. ippBorderMirror      Mirrored border is used. r ippBorderMirrorR     Mirrored border with replication is used. ippBorderInMem       Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `FilterSobelHorizBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Sobel filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the *mask* value. The kernels have the following values:

		1	4	6	4	1		
1	2	1	2	8	12	8	2	
0	0	0	or	0	0	0	0	
-1	-2	-1		-2	-8	-12	-8	-2
				-1	-4	-6	-4	-1

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiFilterSobelHorizBorderGetBufferSize` and `ippiFilterSobelHorizBorder_8u16s_C1R` functions to filter an image with the Sobel horizontal kernel.

```
IppStatus fix_sobelhoriz_8u16( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s pDst[8*7];
    Ipp8u *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int srcStep = 9 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp16s);
    int bufferSize;
    IppStatus status;
   ippiFilterSobelHorizBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
```

```

&bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status =ippiFilterSobelHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
            borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}

```

The result is as follows:

```

pDst -->
12 12 19 33 40 43 49 52
12 12 19 33 40 42 47 51
 8   8   8   8   8   8   8   8
14   8   5   7   4   2   6   4
 6   0  -3  -1  -4  -6  -2  -4
-8  -8  -8  -8  -8  -8  -8  -8
-4  -4  -4  -4  -4  -4  -4  -4

```

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterSobelHorizBorderGetBufferSize](#) Computes the size of the work buffer for the Sobel Horizontal filter.

## FilterSobelHorizSecondBorderGetBufferSize

*Computes the size of the work buffer for the Sobel Horizontal (second derivative) filter.*

### Syntax

```
IppStatusippiFilterSobelHorizSecondBorderGetBufferSize(IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type. Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.

<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

## Description

The `ippiFilterSobelHorizSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelHorizSecondBorder` function. The result is stored in the *pBufferSize* parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobelHorizSecondBorder](#) Applies horizontal (second derivative) Sobel filter with border.

## FilterSobelHorizSecondBorder

Applies horizontal (second derivative) Sobel filter with border.

---

## Syntax

```
IppStatus ippiFilterSobelHorizSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R    32f\_C1R

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination image ROI.
<i>mask</i>	Type of the filter kernel.
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible:
	ippBorderConst Values of all border pixels are set to constant.
	ippBorderRepl Replicated border is used.
	ippBorderMirro r Mirrored border is used
	ippBorderMirro rR Mirrored border with replication is used
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative horizontal Sobel filter (*y*-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

1    4    6    4    1
1    2    1              0    0    0    0    0
-2    -4    -2          or      -2    -8    -12    -8    -2
1    2    1              0    0    0    0    0
1    4    6    4    1

The function requires the working buffer *pBuffer* which size should be computed by the function [ippiFilterSobelHorizSecondBorderGetBufferSize](#) beforehand.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i>

ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if <i>borderType</i> or <i>divisor</i> has a wrong value.
ippStsMaskErr	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelVertBorderGetBufferSize

Computes the size of the work buffer for the Sobel Vertical filter.

---

### Syntax

```
IppStatusippiFilterSobelVertBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize  
 mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*  
 pBufferSize);
```

```
IppStatusippiFilterSobelVertBorderGetBufferSize_T(IppiSize dstRoiSize, IppiMaskSize  
 mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*  
 pBufferSize);
```

### Include Files

ippi.h

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type. Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

### Description

The *ippiFilterSobelVertBorderGetBufferSize* function computes the size, in bytes, of the external work buffer needed for the *ippiFilterSobelVertBorder* function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the *ippiFilterSobelVertBorder* function description.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.

---

ippStsMaskSizeErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>srcDatatype</i> or <i>dstDatatype</i> has an illegal value.
ippStsNumChannelsError	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobelVertBorder](#) Filters an image using a vertical Sobel filter.

## FilterSobelVertBorder

*Filters an image using a vertical Sobel filter.*

### Syntax

```
IppStatusippiFilterSobelVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C1R 16s\_C1R 32f\_C1R

```
IppStatusippiFilterSobelVertBorder_<mod>_T(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C1R\_T

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> . Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .
<i>borderType</i>	Type of border. Possible values are:

	ippBorderConst	Values of all border pixels are set to constant.
	ippBorderRepl	Border is replicated from the edge pixels.
	ippBorderMirro r	Mirrored border is used.
	ippBorderMirro rR	Mirrored border with replication is used.
	ippBorderInMem	Border is obtained from the source image pixels in memory.
		Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between any of the <code>ippBorderRepl</code> , <code>ippBorderConst</code> , <code>ippBorderMirror</code> , or <code>ippBorderMirrorR</code> values and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>borderValue</i>		Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>		Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer *pBuffer* using the `FilterSobelVertBorderGetBufferSize` function.

This function operates with ROI.

This function applies a vertical Sobel filter to the *pSrc* source image ROI. The size of the source image ROI is equal to the destination image ROI size *dstRoiSize*. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the *mask* value. The kernels have the following values:

		-1 -2 0 2 1
-1 0 1		-4 -8 0 8 4
-2 0 2	or	-6 -12 0 12 6
-1 0 1		-4 -8 0 8 4
		-1 -2 0 2 1

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is <code>NULL</code> .

ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiFilterSobelVertBorderGetBufferSize` and `ippiFilterSobelVertBorder` functions.

```
IppStatus fix_sobel_8u16( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s pDst[8*7];
    Ipp8u *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int srcStep = 9 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp16s);
    int bufferSize;
    IppStatus status;
    ippiFilterSobelVertBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterSobelVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
        borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst ->
-4   -8  -483 -483  -8 279 281  -6
-4   -8  -497 -497  -8 274 275  -7
-4   -8  -504 -504  -8 272 272  -8
-2   -4  -505 -505  -4 270 270  -4
 2    4  -507 -507   4 266 266   4
 4    8  -508 -508   8 264 264   8
 4    8  -508 -508   8 264 264   8
```

## See Also

[Borders in Neighborhood Operations](#)

[Regions of Interest in Intel IPP](#)

## User-defined Border Types

**FilterSobelVertBorderGetBufferSize** Computes the size of the work buffer for the Sobel Vertical filter.

## FilterSobelVertSecondBorderGetBufferSize

*Computes the size of the work buffer for the Sobel vertical (second derivative) filter.*

---

### Syntax

```
IppStatusippiFilterSobelVertSecondBorderGetBufferSize(IppiSize dstRoiSize,  
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,  
int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type. Possible values are <code>ippMskSize3x3</code> or <code>ippMskSize5x5</code> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

### Description

The `ippiFilterSobelVertSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelVertSecondBorder` function. The result is stored in the *pBufferSize* parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobelVertSecondBorder](#) Applies vertical (second derivative) Sobel filter with border.

## FilterSobelNegVertBorderGetBufferSize

Computes the size of the work buffer for the Sobel vertical filter.

### Syntax

```
IppStatusippiFilterSobelNegVertBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize  
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*  
pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type. Possible values are <i>ippMskSize3x3</i> or <i>ippMskSize5x5</i> .
<i>srcDataType</i>	Data type of the source image.
<i>dstDataType</i>	Data type of the destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size of the external work buffer.

### Description

The `ippiFilterSobelVertSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelNegVertBorder` function. The result is stored in the *pBufferSize* parameter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>dstRoiSize</i> is negative, or equal to zero.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>mask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>srcDataType</i> or <i>dstDataType</i> has an illegal value.
<code>ippStsNumChannelsError</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[FilterSobelNegVertSecondBorder](#) Applies vertical Sobel filter with border.

## FilterSobelNegVertBorder

Applies vertical Sobel filter with border.

### Syntax

```
IppStatusippiFilterSobelNegVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R    32f\_C1R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.								
<i>pDst</i>	Pointer to the destination image ROI.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.								
<i>dstRoiSize</i>	Size of the destination image ROI.								
<i>mask</i>	Type of the filter kernel.								
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible: <table> <tr> <td><i>ippBorderConst</i></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><i>ippBorderRepl</i></td><td>Replicated border is used.</td></tr> <tr> <td><i>ippBorderMirro</i> <i>r</i></td><td>Mirrored border is used</td></tr> <tr> <td><i>ippBorderMirro</i> <i>rR</i></td><td>Mirrored border with replication is used</td></tr> </table>	<i>ippBorderConst</i>	Values of all border pixels are set to constant.	<i>ippBorderRepl</i>	Replicated border is used.	<i>ippBorderMirro</i> <i>r</i>	Mirrored border is used	<i>ippBorderMirro</i> <i>rR</i>	Mirrored border with replication is used
<i>ippBorderConst</i>	Values of all border pixels are set to constant.								
<i>ippBorderRepl</i>	Replicated border is used.								
<i>ippBorderMirro</i> <i>r</i>	Mirrored border is used								
<i>ippBorderMirro</i> <i>rR</i>	Mirrored border with replication is used								
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).								
<i>pBuffer</i>	Pointer to the working buffer.								

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply the vertical Sobel filter ( $x$ -derivative) to the source image ROI  $pSrc$  and stores results to the destination image ROI of the same size  $pDst$ . Source image can be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell of the kernel (red).

The function `ippiFilterSobelVertBorde` uses the kernels with the following coefficients:

			-1	-2	0	2	1	
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
			-1	-2	0	2	1	

The function `ippiFilterSobelNegVertBored` uses the kernels which coefficients are the same in magnitude but opposite in sign:

			1	2	0	-2	-1	
1	0	-1		4	8	0	-8	-4
2	0	-2	or	6	12	0	-12	-6
1	0	-1		4	8	0	-8	-4
			1	2	0	-2	-1	

Before using this function, compute the size of the work buffer `pBuffer` using the [FilterSobelNegVertBorderGetBufferSize](#) function.

[Example](#) shows how the function `ippiFilterSobelNegVertBorder_8u16s_C1R` can be used for edge detection.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <code>borderType</code> or <code>divisor</code> has a wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has a wrong value.

## See Also

[User-defined Border Types](#)

## FilterSobelVertSecondBorder

Applies vertical (second derivative) Sobel filter with border.

---

### Syntax

```
IppStatusippiFilterSobelVertSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R    32f\_C1R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.								
<i>pDst</i>	Pointer to the destination image ROI.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.								
<i>dstRoiSize</i>	Size of the destination image ROI.								
<i>mask</i>	Type of the filter kernel.								
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible: <table> <tr> <td><i>ippBorderConst</i></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><i>ippBorderRepl</i></td><td>Replicated border is used.</td></tr> <tr> <td><i>ippBorderMirro r</i></td><td>Mirrored border is used</td></tr> <tr> <td><i>ippBorderMirro rR</i></td><td>Mirrored border with replication is used</td></tr> </table>	<i>ippBorderConst</i>	Values of all border pixels are set to constant.	<i>ippBorderRepl</i>	Replicated border is used.	<i>ippBorderMirro r</i>	Mirrored border is used	<i>ippBorderMirro rR</i>	Mirrored border with replication is used
<i>ippBorderConst</i>	Values of all border pixels are set to constant.								
<i>ippBorderRepl</i>	Replicated border is used.								
<i>ippBorderMirro r</i>	Mirrored border is used								
<i>ippBorderMirro rR</i>	Mirrored border with replication is used								
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).								
<i>pBuffer</i>	Pointer to the working buffer.								

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative vertical Sobel filter ( $x$ -derivative) to the source image  $pSrc$  and stores results to the destination image of the same size  $pDst$ . Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

		1	0	-2	0	1		
1	-2	1	4	0	-8	0	4	
2	<span style="color:red">-4</span>	2	or	6	0	<span style="color:red">-12</span>	0	6
1	-2	1		4	0	-8	0	4
			1	0	-2	0	1	

The function requires the working buffer  $pBuffer$  which size should be computed by the function [ippiFilterSobelVertSecondBorderGetBufferSize](#) beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <code>roiSize</code> has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if <code>borderType</code> or <code>divisor</code> has a wrong value.
ippStsMaskErr	Indicates an error condition if <code>mask</code> has a wrong value.

## Example

To better understand usage of this function, refer to the `FilterSobelVertSecondBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## FilterSobelCrossGetBufferSize

Computes the size of the external buffer for the cross Sobel filter with border.

### Syntax

```
IppStatusippiFilterSobelCrossGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask,  
int* pBufferSize);
```

Supported values for `mod`:

8u16s\_C1R    32f\_C1R

## Include Files

ippcv.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

## Description

This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelCrossBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table "Types of the Fixed Filter Functions"](#)). This buffer *pBufferSize[0]* can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelCrossBorder

*Applies second derivative cross Sobel filter with border.*

## Syntax

```
IppStatusippiFilterSobelCrossBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,  
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s\_C1R    32f\_C1R

## Include Files

ippcv.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>mask</i>	Type of the filter kernel.
<i>borderType</i>	Type of border (see <a href="#">Borders in Neighborhood Operations</a> ); following values are possible: ippBorderConst Values of all border pixels are set to constant. ippBorderRepl Replicated border is used.
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the second derivative cross Sobel filter ( $xy$ -derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

			-1	-2	0	2	1	
-1	0	1		-2	-4	0	4	2
0	0	0	or	0	0	0	0	0
1	0	-1		2	4	0	-4	-2
				1	2	0	-2	-1

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterSobelCrossGetBufferSize](#) beforehand.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with a zero or negative value.

ippStsStepErr	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if <code>borderType</code> or <code>divisor</code> has a wrong value.
ippStsMaskErr	Indicates an error condition if <code>mask</code> has a wrong value.

## GenSobelKernel

*Computes kernel for the Sobel filter.*

---

### Syntax

```
IppStatusippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx, int sign);
IppStatusippiGenSobelKernel_32f(Ipp32f* pDst, int kernelSize, int dx, int sign);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pDst</i>	Pointer to the destination vector.
<i>kernelSize</i>	Size of the Sobel kernel.
<i>dx</i>	Order of derivative.
<i>sign</i>	Specifies signs of kernel elements.

### Description

This function computes the one-dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

$$(1 + x)^{kernelSize - dx - 1} \cdot (x - 1)^{dx}$$

If the `sign` parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by a high order Sobel filter.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the <code>pDst</code> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <code>kernelSize</code> is less than 3 or is even.

---

ippStsBadArgErr	Indicates an error condition if <i>dx</i> is equal to or less than <i>kernelSize</i> , or <i>dx</i> is negative.
-----------------	--

## Deinterlacing Filters

---

This section describes functions that perform image deinterlacing.

### DeinterlaceFilterCAVT

*Performs deinterlacing of two-field image.*

---

#### Syntax

```
IppStatusippiDeinterlaceFilterCAVT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, Ipp16u threshold, IppiSize roiSize);
```

#### Include Files

ippi.h

#### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>threshold</i>	Threshold level value.
<i>roiSize</i>	Size of the source and destination image ROI.

#### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs deinterlacing of a two-field image, pointed to by *pSrc*, using content adaptive vertical temporal (CAVT) filtering.

The field pointed to by *pSrc* is copied to *pDst*, while the other field in the destination image, pointed to by *pDst+dstStep*, is the interpolated one. Note that you can set the pointers to the bottom left corner of the images and use negative steps to have the bottom field unchanged and the top one interpolated.

The *threshold* parameter is the edge detection threshold with the valid range [0-2041] regulating the probability of temporal interpolation: 0 means that all the pixels are interpolated only spatially, from the vertically neighbouring pixels of the copied field, 2041 - that combined spatial-temporal interpolation, involving the pixels from the modified field as well, is applied to all the pixels.

#### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
-------------	---

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize.width</i> is less than or equal to 0, or <i>roiSize.height</i> is odd or less than 8.

## Median

*Creates an image consisting of median values of three source images.*

---

### Syntax

```
IppStatusippiMedian_8u_P3C1R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize size);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>size</i>	Size of the source and destination image ROI.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function sets each pixel in the destination image ROI as the median value of correspondent pixels in the each plane of the source image.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>size</i> has a field with zero or negative value.

# Image Linear Transforms

10

This chapter describes the Intel® IPP image processing functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating-point data only.

## Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see [Function Naming](#) in Chapter 2).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of *flag* argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

**Normalization Factors for Fourier Transform Results**

Value of <i>flag</i> Argument	Normalization Factors	
	Forward Transform	Inverse Transform
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt (MN)	1/sqrt (MN)
IPP_FFT_NODIV_BY_ANY	1	1

In this table, *N* and *M* denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed).

For the FFT, these lengths must be integer powers of 2, that is  $N=2^{\text{orderX}}$ ,  $M=2^{\text{orderY}}$ , where power exponents are known as order of FFT.

For the DFT, *N* and *M* can take on arbitrary integer non-negative values.

## Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin. The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the *N* by *M* transform, the respective FFT and DFT functions actually store real and imaginary parts of the

complex Fourier coefficients  $A(i, j)$  for  $i = 0, \dots, M-1$ ;  $j = 0, \dots, N/2$  in a single real array of dimensions  $(N, M)$ . The RCPack2D storage format is slightly different for odd and even  $M$  and is arranged in accordance with the following tables:

#### RCPack2D Storage for Odd Number of Rows

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0, (N-1)/2)	Im A(0, (N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1, (N-1)/2)	Im A(1, (N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2, (N-1)/2)	Im A(2, (N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/ 2,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2, (N-1)/2)	Im A(M-2, (N-1)/2)	Re A(M/2,N/2)
Im A(M/ 2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1, (N-1)/2)	Im A(M-1, (N-1)/2)	Im A(M/2,N/2)

#### RCPack2D Storage for Even Number of Rows

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0, (N-1)/2)	Im A(0, (N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1, (N-1)/2)	Im A(1, (N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2, (N-1)/2)	Im A(2, (N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/ 2-1,0)	Re A(M-3,1)	Im A(M-3,1)	...	Re A(M-3, (N-1)/2)	Im A(M-3, (N-1)/2)	Re A(M/ 2-1,N/2)
Im A(M/ 2-1,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2, (N-1)/2)	Im A(M-2, (N-1)/2)	Im A(M/ 2-1,N/2)
Re A(M/ 2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1, (N-1)/2)	Im A(M-1, (N-1)/2)	Re A(M/2,N/2)

The shaded columns to the right side of the tables indicate values for even  $N$  only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way: C1-ReA(0,0); C2-ReA(0,0); C3-ReA(0,0); C1-ReA(0,1); C2-ReA(0,1); C3-ReA(0,1); C1-ImA(0,1); C2-ImA(0,1); ...

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

$$A(i,j) = \text{conj}(A(M-i,N-j)), \quad i = 1, \dots, M-1; \quad j = 1, \dots, N-1$$

$$A(0,j) = \text{conj}(A(0,N-j)), \quad j = 1, \dots, N-1$$

$$A(i,0) = \text{conj}(A(M-i,0)), \quad i = 1, \dots, M-1$$

## FFTGetSize

Computes the size of the FFT context structure and the size of the work buffer.

### Syntax

```
IppStatusippiFFTGetSize_R_32f (int orderX, int orderY, int flag, IppHintAlgorithm  
hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatusippiFFTGetSize_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm  
hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>orderX</i> , <i>orderY</i>	Order of the FFT in x- and y- directions, respectively.
<i>flag</i>	Flag to choose the option for results normalization.
<i>hint</i>	This parameter is deprecated. Set the value to <code>ippAlgHintNone</code> .
<i>pSizeSpec</i>	Pointer to the size of the FFT context structure.
<i>pSizeInit</i>	Pointer to the size of the buffer for the FFT initialization function.
<i>pSizeBuf</i>	Pointer to the size of the FFT external work buffer.

## Description

This function computes the following:

- Size of the FFT context structure. The result in bytes is stored in the *pSpecSize* parameter.
- Size of the work buffer for the `ippiFFTInit` functions. The result in bytes is stored in the *pSizeInit* parameter.
- Size of the work buffer for the `ippiFFTFwd` and `ippiFFTInv` functions. The result in bytes is stored in the *pSizeBuf* parameter.

The suffix after the function name indicates the flavors of the FFT functions: `ippiFFTGetSize_C` is for complex flavors and `ippiFFTGetSize_R` is for real flavors.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsFftOrderErr</code>	Indicates an error condition when the FFT order value is illegal.
<code>ippStsFFTFlagErr</code>	Indicates an error condition when the <i>flag</i> value is illegal.

## See Also

[FFTInit](#) Initializes the context structure for the image FFT functions.

[FFTInv](#) Applies an inverse FFT to complex source data and stores results in a destination image.

[FFTFwd](#) Applies forward Fast Fourier Transform to an image.

## FFTInit

*Initializes the context structure for the image FFT functions.*

## Syntax

```
IppStatusippiFFTInit R 32f (int orderX, int orderY, int flag, IppHintAlgorithm hint,  
IppFFTSpec_R_32f* pFFTSPEC, Ipp8u* pMemInit);
```

```
IppStatusippiFFTInit_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm hint,
IppiFFTSpec_C_32fc* pFFTSPEC, Ipp8u* pMemInit);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>orderX</i> , <i>orderY</i>	Order of the FFT in x- and y- directions, respectively.
<i>flag</i>	Flag to choose the option for results normalization.
<i>hint</i>	This parameter is deprecated. Set the value to <code>ippAlgHintNone</code> .
<i>pFFTSPEC</i>	Pointer to the FFT context structure.
<i>pMemInit</i>	Pointer to the temporary work buffer.

## Description

This function initializes the *pFFTSPEC* context structure needed to compute the forward and inverse FFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the `ippiFFTGetSize` function.

The *pMemInit* parameter can be `NULL` only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The `ippiFFTfwd` and `ippiFTInv` functions called with the pointer to the initialized *pFFTSPEC* structure, compute the fast Fourier transform with the following characteristics:

- length  $N=2^{orderX}$  in x-direction by  $M=2^{orderY}$  in y-direction
- results normalization mode as set by *flag* (see [Table “Normalization Factors for Fourier Transform Results”](#))
- computation algorithm indicated by *hint*.

The suffix after the function name indicates the type of the context structure to be initialized:

`ippiFFTInit_C` is for the complex FFT context structure and `ippiFFTInit_R` is for the real FFT context structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsFftOrderErr</code>	Indicates an error condition when the FFT order value is illegal.
<code>ippStsFFTFlagErr</code>	Indicates an error condition when the <i>flag</i> value is illegal.

## Example

The code example below demonstrates how to use the `ippiFFTGetSize` and `ippiFFTInit` functions.

```
/// get sizes for required buffers
ippiFFTGetSize_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec,
&sizeInit, &sizeBuffer );

/// allocate memory for required buffers
pMemSpec = (IppiFFTSpec_R_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = ippMalloc ( sizeBuffer );
}

/// initialize FFT specification structure
ippiFFTInit_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec,
pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward FFT to put source data to frequency domain
ippiFFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );

/// ...

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}

ippFree( pMemSpec );
```

## See Also

[FFTGetSize](#) Computes the size of the FFT context structure and the size of the work buffer.

[FFTInv](#) Applies an inverse FFT to complex source data and stores results in a destination image.

[FFTFwd](#) Applies forward Fast Fourier Transform to an image.

## FFTFwd

Applies forward Fast Fourier Transform to an image.

## Syntax

### Case 1: Not-in-place operation on floating-point data

```
IppStatusippiFFTForward_RToPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

### Case 2: Not-in-place operation on complex data

```
IppStatusippiFFTForward_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

### Case 3: In-place operation on floating-point data

```
IppStatusippiFFTForward_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

### Case 4: In-place operation on complex data

```
IppStatusippiFFTForward_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

---

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pFFTSPEC</i>	Pointer to the FFT specification structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI.

This function performs a forward FFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY*.

The function flavor `ippiFFTwd_RToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with `AC4` descriptor do not process alpha channel.

The function flavor `ippiFFTwd_CToC` that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward FFT functions, you need to compute the size of the work buffer by `ippiFFTGetSize` and initialize the context structure by the `ippiFFTInit` function. The forward FFT functions use the *pFFTSPEC* context structure to set the mode of calculations and retrieve support data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSPEC</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSPEC</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

To better understand usage of these functions, refer to the `FFTwd_CToC.c` and `FFTwd_RToPack.c` examples in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples> :

## See Also

[Regions of Interest in Intel IPP](#)

[FFTGetSize](#) Computes the size of the FFT context structure and the size of the work buffer.

[FFTInit](#) Initializes the context structure for the image FFT functions.

## FFTInv

*Applies an inverse FFT to complex source data and stores results in a destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on floating-point data

```
IppStatusippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiFFTSPEC_R_32f* pFFTSPEC, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

#### Case 2: Not-in-place operation on complex data

```
IppStatusippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFFTSPEC_C_32fc* pFFTSPEC, Ipp8u* pBuffer);
```

#### Case 3: In-place operation on floating-point data

```
IppStatusippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSPEC_R_32f* pFFTSPEC, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1IR

32f\_C3IR

32f\_C4IR

32f\_AC4IR

#### Case 4: In-place operation on complex data

```
IppStatusippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiFFTSPEC_C_32fc* pFFTSPEC, Ipp8u* pBuffer);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pFFTSpec</i>	Pointer to the previously initialized FFT context structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI.

This function performs an inverse FFT on each channel of the source image *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY*.

For the `ippiFFTInv_PackToR`, function flavor, the input buffer must contain data in [RCPack2D format](#).

Before using the inverse FFT functions, you need to compute the size of the work buffer by `ippiFFTGetSize` and initialize the context structure by the `ippiFFTInit` function. The inverse FFT functions use the *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

To better understand usage of these functions, refer to the `FFTInv_CToC.c` and `FFTInv_RToPack.c` examples in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[FFTGetSize](#) Computes the size of the FFT context structure and the size of the work buffer.

[FFTInit](#) Initializes the context structure for the image FFT functions.

## DFTGetSize

*Computes the size of the FFT context structure and the size of the work buffer.*

---

### Syntax

```
IppStatusippiDFTGetSize_R_32f(IppiSize roiSize, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);

IppStatusippiDFTGetSize_C_32fc(IppiSize roiSize, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>flag</i>	Flag to choose the option for results normalization. For more information, see <a href="#">Table "Normalization Factors for Fourier Transform Results"</a>
<i>hint</i>	This parameter is deprecated. Set the value to <code>ippAlgHintNone</code> .
<i>pSizeSpec</i>	Pointer to the size of the DFT context structure.
<i>pSizeInit</i>	Pointer to the size of the buffer for the DFT initialization function.
<i>pSizeBuf</i>	Pointer to the size of the DFT external work buffer.

### Description

This function computes the following:

- Size of the DFT context structure. The result in bytes is stored in the *pSpecSize* parameter.
- Size of the work buffer for the `ippiDFTInit` functions. The result, in bytes, is stored in the *pSizeInit* parameter.
- Size of the work buffer for the `ippiDFTFwd` and `ippiDFTInv` functions. The result, in bytes, is stored in the *pSizeBuf* parameter.

The suffix after the function name indicates the flavors of the DFT functions: `ippiDFTGetSize_C` is for complex flavors and `ippiDFTGetSize_R` is for real flavors.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsFftFlagErr</code>	Indicates an error condition when the <i>flag</i> value is illegal.

---

ippStsFftOrderErr	Indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size <i>roiSize</i> exceeds the limit.
ippStsSizeErr	Indicates an error condition when the <i>roiSize</i> has a field with a zero or negative value.

## See Also

[DFTInit](#) Initializes the context structure for the image DFT functions.

## DFTInit

*Initializes the context structure for the image DFT functions.*

## Syntax

```
IppStatusippiDFTInit_R_32f (IppiSize roiSize, int flag, IppHintAlgorithm hint,
IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pMemInit);

IppStatusippiDFTInit_C_32fc (IppiSize roiSize, int flag, IppHintAlgorithm hint,
IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pMemInit);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>flag</i>	Flag to choose the option for results normalization.
<i>hint</i>	This parameter is deprecated. Set the value to <code>ippAlgHintNone</code> .
<i>pDFTSpec</i>	Pointer to the DFT context structure.
<i>pMemInit</i>	Pointer to the temporary work buffer.

## Description

This function initializes the *pDFTSpec* context structure needed to compute the forward and inverse DFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the `ippiDFTGetSize` function.

The *pMemInit* parameter can be `NULL` only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The `ippiDFTFwd` and `ippiDFTInv` functions called with the pointer to the initialized *pDFTSpec* structure compute the discrete Fourier transform with the following characteristics:

- ROI of the *roiSize* size
- results normalization mode set by *flag* (see [Table "Normalization Factors for Fourier Transform Results"](#))

- computation algorithm indicated by *hint*.

The suffix after the function name indicates the type of the context structure to be initialized: `ippiDFTInit_C` is for the complex DFT context structure and `ippiDFTInit_R` is for the real DFT context structure.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsFftFlagErr</code>	Indicates an error condition when the <i>flag</i> value is illegal.
<code>ippStsFftOrderErr</code>	Indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size <i>roiSize</i> exceeds the limit.
<code>ippStsSizeErr</code>	Indicates an error condition when the <i>roiSize</i> has a field with a zero or negative value.

## Example

The code example below demonstrates how to use the `ippiDFTGetSize` and `ippiDFTInit` functions.

```

/// get sizes for required buffers
ippiDFTGetSize_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec, &sizeInit,
&sizeBuffer );

/// allocate memory for required buffers
pMemSpec = (IppiDFTSpec_R_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = ippMalloc ( sizeBuffer );
}

/// initialize DFT specification structure
ippiDFTInit_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec, pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward DFT to put source data to frequency domain
ippiDFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );

/// ...

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}

```

```

    }

    ippFree( pMemSpec );
}

```

## See Also

[Regions of Interest in Intel IPP](#)

[DFTGetSize](#) Computes the size of the FFT context structure and the size of the work buffer.

[DFTFwd](#) Applies forward discrete Fourier transform to an image.

[DFTInv](#) Applies an inverse DFT to complex source data and stores results in a destination image.

## DFTFwd

*Applies forward discrete Fourier transform to an image.*

### Syntax

#### Case 1: Not-in-place operation on floating-point data

```
IppStatusippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

- 32f\_C1R
- 32f\_C3R
- 32f\_C4R
- 32f\_AC4R

#### Case 2: Not-in-place operation on complex data

```
IppStatusippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

#### Case 3: In-place operation on floating-point data

```
IppStatusippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

- 32f\_C1IR
- 32f\_C3IR
- 32f\_C4IR
- 32f\_AC4IR

#### Case 4: In-place operation on complex data

```
IppStatusippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI.

This function performs a forward DFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors).

The function flavor `ippiDFTFwd_RToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with `AC4` descriptor do not process alpha channel.

The function flavor `ippiDFTFwd_CToC` that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward DFT functions, you need to compute the size of the work buffer by `ippiDFTGetSize` and initialize the context structure by the `ippiDFTInit` function. The forward DFT functions use the *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## See Also

[Regions of Interest in Intel IPP](#)

**DFTGetSize** Computes the size of the FFT context structure and the size of the work buffer.

**DFTInit** Initializes the context structure for the image DFT functions.

## DFTInv

*Applies an inverse DFT to complex source data and stores results in a destination image.*

### Syntax

#### Case 1: Not-in-place operation on floating-point data

```
IppStatusippiDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f\_C1R  
32f\_C3R  
32f\_C4R  
32f\_AC4R

#### Case 2: Not-in-place operation on complex data

```
IppStatusippiDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

#### Case 3: In-place operation on floating-point data

```
IppStatusippiDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

#### Case 5: In-place operation on complex data

```
IppStatusippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI.

This function performs an inverse DFT on each channel of the input buffer *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer *pDst* (*pSrcDst* for in-place flavors).

For function flavor `ippiDFTInv_PackToR`, the input buffer must contain data in [RCPack2D format](#).

Before using the inverse DFT functions, you need to compute the size of the work buffer by `ippiDFTGetSize` and initialize the context structure by the `ippiDFTInit` function. The inverse DFT functions use the *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDFTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## See Also

[Regions of Interest in Intel IPP](#)

[DFTGetSize](#) Computes the size of the FFT context structure and the size of the work buffer.

[DFTInit](#) Initializes the context structure for the image DFT functions.

## MulPack

*Multiples two source images in packed format.*

---

## Syntax

```
IppStatusippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int
src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1R  
32f\_C3R  
32f\_C4R  
32f\_AC4R

```
IppStatusippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

*scaleFactor*Scale factor (see [Integer Result Scaling](#)).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source images, *A* and *B* represented in [RCPack2D format](#) and stores the result into the destination image *C* in packed format also. The multiplying is performed according to the following formulas:

$$\text{Re}_C = \text{Re}_A * \text{Re}_B - \text{Im}_A * \text{Im}_B;$$

$$\text{Im}_C = \text{Im}_A * \text{Re}_B + \text{Im}_B * \text{Re}_A.$$

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

This function can be used in image filtering operations that include FFT transforms.

## Example

To better understand usage of this function, refer to the `MulPack.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## MulPackConj

*Multiples a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.*

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiMulPackConj_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`32f_C1R``32f_C3R``32f_C4R`

---

32f\_AC4R

### Case 2: In-place operation

```
IppStatusippiMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc1, pSrc2</i>	Pointer to the ROI in the source images.
<i>src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrc</i>	Pointer to the first source image ROI for the in-place operation.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values of the source image  $A$  by the corresponding pixel values of the complex conjugate image  $A^*$ , represented in [RCPack2D format](#). The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## Magnitude

*Computes magnitude of elements of a complex data image.*

---

## Syntax

```
IppStatusippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`32fc32f_C1R 32fc32f_C3R`

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

## MagnitudePackGetBufferSize

Computes the size of the work buffer for the [ippiMagnitudePack](#) function.

### Syntax

```
IppStatusippiMagnitudePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize,
int* pSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>numChannels</i>	Number of channels in the image. Possible values are 1 and 3.
<i>dstRoiSize</i>	Size, in pixels, of the destination image ROI.
<i>pSize</i>	Pointer to the computed size of the external work buffer, in bytes.

### Description

The [ippiMagnitudePackGetBufferSize](#) function computes the size, in bytes, of the external work buffer needed for the [ippiMagnitudePack](#) function. The result is stored in the *pSize* parameter.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> has a field with a value less than 1.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

### See Also

[MagnitudePack](#) MODIFIED API. Computes magnitude of elements of an image in packed format.

## MagnitudePack

*MODIFIED API. Computes magnitude of elements of an image in packed format.*

---

### Syntax

```
IppStatusippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R      32f\_C3R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>pBuffer</i>	Pointer to the work buffer. To compute the size of the buffer, use the <a href="#">ippiMagnitudePackGetBufferSize</a> function.

### Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

---

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in [RCPack2D](#) format and stores results in the destination image *pDst*.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error when <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<i>ippStsStepErr</i>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> value is zero or negative.

`ippStsSizeErr` Indicates an error when width or height of images is less than, or equal to zero.

## See Also

[MagnitudePackGetBufferSize](#) Computes the size of the work buffer for the `ippiMagnitudePack` function.

[Regions of Interest in Intel IPP](#)

[Real - Complex Packed \(RCPack2D\) Format](#)

## Phase

*Computes the phase of elements of a complex data image.*

## Syntax

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32fc32f\_C1R 32fc32f\_C3R

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase in radians of elements of a source image `pSrc` given in complex data format, and stores results in the destination image `pDst`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.

ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

## PhasePackGetBufferSize

*Computes the size of the work buffer for the [ippiPhasePack](#) function.*

---

### Syntax

```
IppStatus ippiPhasePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize, int* pSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>numChannels</i>	Number of channels in the image. Possible values are 1 and 3.
<i>dstRoiSize</i>	Size, in pixels, of the destination image ROI.
<i>pSize</i>	Pointer to the computed size of the external work buffer, in bytes.

### Description

The [ippiPhasePackGetBufferSize](#) function computes the size, in bytes, of the external work buffer needed for the [ippiPhasePack](#) function. The result is stored in the *pSize* parameter.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> has a field with a value less than 1.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

### See Also

[PhasePack](#) MODIFIED API. Computes the phase of elements of an image in packed format.

## PhasePack

*MODIFIED API. Computes the phase of elements of an image in packed format.*

---

## Syntax

```
IppStatusippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

32f\_C1R    32f\_C3R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>pBuffer</i>	Pointer to the work buffer. To compute the size of the buffer, use the <a href="#">PhasePackGetBufferSize</a> function.

## Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

---

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase of elements of a source image *pSrc* given in [RCPack2D format](#) and stores the results in the destination image *pDst*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when <i>pSrc</i> or <i>pDst</i> pointer is NULL.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
ippStsSizeErr	Indicates an error when width or height of images is less than, or equal to zero.

## See Also

[PhasePackGetBufferSize](#) Computes the size of the work buffer for the `ippiPhasePack` function.

[Regions of Interest in Intel IPP](#)

[Real - Complex Packed \(RCPack2D\) Format](#)

## PolarToCart

Converts an image in the polar coordinate form to Cartesian coordinate form.

---

### Syntax

```
IppStatusippiPolarToCart_<mod>(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase, int srcStep, IppiSize roiSize, Ipp32fc* pDst, int dstStep);
```

Supported values for mod:

32fc\_C1R    32fc\_C3R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrcMagn</i>	Pointer to the buffer containing magnitudes of the source image.
<i>pSrcPhase</i>	Pointer to the buffer containing phase values of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffers.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the polar coordinate of the source image stored in the arrays of magnitudes *pSrcMagn* and phase values *pSrcPhase* to the destination image *pDst* in complex-data format (in Cartesian coordinate form).

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>srcSize</i> has a field with value less than 1.

## Example

The code example below shows how to use the function `ippiPolarToCart_32fc_C1R`.

```
void func_polartocart()
{
    Ipp32f pSrcMagn[2*2] = {1.0, 0.0, 2.1, 3.2};

    Ipp32f pSrcPhase[2*2] = {0.0, 2.0, 1.6,-1.0};

    Ipp32fc pDst[2*2] = {0};

    int srcStep = 2*sizeof(Ipp32f);

    int dstStep = 2*sizeof(Ipp32fc);

    IppiSize roiSize = {2, 2};

    ippipolarToCart_32fc_C1R(pSrcMagn, pSrcPhase, srcStep, roiSize, pDst, dstStep);

}

Result: pDst - > (1.0, 0.0) (0.0, 0.0) (-0.1, 2.1) (1.7, -2.7)
```

## PackToCplxExtend

*Converts an image in packed format to a complex data image.*

### Syntax

```
IppStatus ippipacktocplxExtend_32f32fc_C1R(const Ipp32f* pSrc, IppiSize srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

### Include Files

`ippi.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcSize</i>	Size in pixels of the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the source image *pSrc* in [RCPack2D format](#) to complex data format and stores the results in *pDst*, which is a matrix with complete set of the Fourier coefficients. Note that if the *pSrc* in RCPack2D format is a real array of dimensions (NxM), then the *pDst* is a real array of dimensions (2xNxM). This should be taken into account when allocating memory for the function operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has field with zero or negative value.

## CpxExtendToPack

*Converts a complex data image to an image in packed format.*

---

## Syntax

```
IppStatusippiCpxExtendToPack_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
IppiSize srcSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for *mod*:

`32fc32f_C1R`

`32fc32f_C3R`

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcSize</i>	Size in pixels of the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the source image *pSrc* in complex data format to [RCPack2D format](#) and stores the results in *pDst*, which is a real array of dimensions (NxM). The *pSrc* is a matrix with complete set of the Fourier coefficients.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has field with zero or negative value.

## Windowing Functions

This section describes Intel IPP windowing functions used in image processing. A window is a mathematical function by which pixel values are multiplied to prepare an image for the subsequent analysis. This procedure is often called 'windowing'. In fact, a window function approaches zero towards the edges of the image avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of window functions:

- Bartlett window function
- Hamming window function

These functions generate the window samples and applied them to the specified image. To obtain the window samples themselves, you should apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is very time consuming, it may be useful if you want to apply the same window to the multiple images. In this case use one of the image multiplication functions (`ippiMul`) to multiply the pixel values of the image by the window samples.

### **WinBartlettGetBufferSize, WinBartlettSepGetBufferSize,**

*Compute the size of the work buffer for the  
ippiWinBartlett or ippiWinBartlettSep function.*

#### Syntax

```
IppStatus ippiWinBartlettGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);

IppStatus ippiWinBartlettSepGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);
```

#### Include Files

`ippi.h`

#### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>dataType</i>	Data type for the Bartlett window function. Possible values are: <code>ipp8u</code> , <code>ipp16u</code> , or <code>ipp32f</code> .
<i>roiSize</i>	Size, in pixels, of the image ROI.
<i>pSize</i>	Pointer to the computed size of the external work buffer, in bytes.

## Description

The `ippiWinBartlettGetBufferSize` and `ippiWinBartlettSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the `ippiWinBartlett` or `ippiWinBartlettSep` function. The result is stored in the *pSize* parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSize</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a value less than 3.
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>dataType</i> has an illegal value.

## See Also

[WinBartlett](#), [WinBartlettSep](#) MODIFIED API. Apply Bartlett window function to the image.

## WinBartlett, WinBartlettSep

MODIFIED API. Apply Bartlett window function to the image.

---

## Syntax

### Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

```
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

## Case 2: In-place operation

```
IppStatusippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize  
roiSize, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      32f\_C1IR

```
IppStatusippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize  
roiSize, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1IR      16u\_C1IR      32f\_C1IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>pBuffer</i>	Pointer to the work buffer. To compute the size of the buffer, use the <a href="#">WinBartlettGetBufferSize</a> <a href="#">WinBartlettSepGetBufferSize</a> or <a href="#">WinBartlettGetBufferSize</a> <a href="#">WinBartlettSepGetBufferSize</a> function.

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Bartlett window function for one-dimensional case with *M* elements is defined as follows:

$$w_{bartlett}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \leq i \leq \frac{M-1}{2} \\ 2 - \frac{2i}{M-1}, & \frac{M-1}{2} < i \leq M-1 \end{cases}$$

The `ippiWinBartlettSep` flavor applies the window function successively to the rows and then to the columns of the image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> or <code>dstStep</code> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error when width or height of images is less than, or equal to zero.

## See Also

[WinBartlettGetBufferSize](#), [WinBartlettSepGetBufferSize](#) Compute the size of the work buffer for the `ippiWinBartlett` or `ippiWinBartlettSep` function.

[Regions of Interest in Intel IPP](#)

## WinHammingGetBufferSize, WinHammingSepGetBufferSize,

*Compute the size of the work buffer for the `ippiWinHamming` or `ippiWinHammingSep` function.*

## Syntax

```
IppStatus ippiWinHammingGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);

IppStatus ippiWinHammingSepGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>dataType</code>	Data type for the Bartlett window function. Possible values are: <code>ipp8u</code> , <code>ipp16u</code> , or <code>ipp32f</code> .
<code>roiSize</code>	Size, in pixels, of the image ROI.
<code>pSize</code>	Pointer to the computed size of the external work buffer, in bytes.

## Description

The `ippiWinHammingGetBufferSize` and `ippiWinHammingSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the `ippiWinHamming` or `ippiWinHammingSep` function. The result is stored in the `pSize` parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a value less than 3.
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>dataType</code> has an illegal value.

## See Also

[WinHamming](#), [WinHammingSep](#) MODIFIED API. Apply Hamming window function to the image.

## WinHamming, WinHammingSep

MODIFIED API. Apply Hamming window function to the image.

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

```
IppStatusippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

### Case 2: In-place operation

```
IppStatusippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1IR</code>	<code>16u_C1IR</code>	<code>32f_C1IR</code>
----------------------	-----------------------	-----------------------

```
IppStatusippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

<code>8u_C1IR</code>	<code>16u_C1IR</code>	<code>32f_C1IR</code>
----------------------	-----------------------	-----------------------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>pBuffer</i>	Pointer to the work buffer. To compute the size of the buffer, use the <a href="#">ippiWinHammingGetBufferSize</a> or <a href="#">ippiWinHammingSepGetBufferSize</a> function.

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Hamming (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Hamming window function for one-dimensional case with *M* elements is defined as follows:

$$w_{\text{hamming}}(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{M-1}\right)$$

The `ippiWinHammingSep` flavor applies the window function successively to the rows and then to the columns of the image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error when width or height of images is less than, or equal to zero.

**See Also**

[WinHammingGetBufferSize](#), [WinHammingSepGetBufferSize](#) Compute the size of the work buffer for the `ippiWinHamming` or `ippiWinHammingSep` function.

[Regions of Interest in Intel IPP](#)

## Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions `ippiDCTFwd` and `ippiDCTInv` need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [Rao90].

The DCT functions `ippiDCT8x8Fwd` and `ippiDCT8x8Inv` working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions `ippiDCT8x8Inv` meet IEEE-1180 standard requirements (see [IEEE]).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([Feig92]) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [AP922].

### Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

## DCTFwdGetSize, DCTInvGetSize

*Compute the size of the DCT context structure and the size of the required work buffers.*

### Syntax

```
IppStatus ippiDCTFwdGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippiDCTInvGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>roiSize</code>	Size of the source and destination ROI, in pixels.
<code>pSizeSpec</code>	Pointer to the size of the DCT context structure.
<code>pSizeInit</code>	Pointer to the size of the buffer for the DCT initialization function.

*pSizeBuf* Pointer to the size of the DCT external work buffer.

## Description

These functions compute the following:

1. Size of the DCT context structure. The result, in bytes, is stored in the *pSizeSpec* parameter.
2. Size of the work buffer for the [ippiDCTFwdInit](#) and [ippiDCTInvInit](#) functions. The result, in bytes, is stored in the *pSizeInit* parameter.
3. Size of the work buffer for the [ippiDCTFwd](#) and [ippiDCTInv](#) functions. The result, in bytes, is stored in the *pSizeBuf* parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.

## See Also

[DCTFwdInit](#) [DCTInvInit](#) Initialize the context structure for the forward or inverse DCT operation.  
[DCTFwd](#) Applies a forward discrete cosine transform to an image.  
[DCTInv](#) Applies an inverse discrete cosine transform to an image.

## DCTFwdInit, DCTInvInit

*Initialize the context structure for the forward or inverse DCT operation.*

---

## Syntax

```
IppStatusippiDCTFwdInit_32f(IppiDCTFwdSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u*pMemInit);  
IppStatusippiDCTInvInit_32f(IppiDCTInvSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u*pMemInit);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pDCTSpec</i>	Pointer to the forward or inverse DCT context structure for initialization.
<i>roiSize</i>	Size of the source and destination ROI, in pixels.
<i>pMemInit</i>	Pointer to the temporary work buffer.

## Description

These functions initialize the `pDCTSpec` context structure to apply the forward or inverse DCT to two-dimensional image data. The `ippiDCTFwd` and `ippiDCTInv` functions use the pointer to the initialized DCT context structure as an argument to compute the forward or inverse DCT for points in the ROI of size `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.

## See Also

- `DCTFwd` Applies a forward discrete cosine transform to an image.  
`DCTInv` Applies an inverse discrete cosine transform to an image.

## DCTFwd

*Applies a forward discrete cosine transform to an image.*

## Syntax

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

```
32f_C1R  
32f_C3R  
32f_C4R  
32f_AC4R
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.

<i>pDCTSpc</i>	Pointer to the previously initialized forward DCT context structure.
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the [ippiDFTInit](#) function.

This function performs a forward DCT on each channel of the source image *pSrc* and writes the result into the corresponding channel of the destination image buffer *pDst*. Note that the function flavor with *AC4* descriptor does not process alpha channel. This function uses the previously initialized *pDCTSpc* context structure to set the mode of calculations and retrieve support data.

You can use this function with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

Before using the forward DCT functions, you need to compute the size of the required buffers and the external work buffer using the [ippiDCTFwdGetSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpc</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpc</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

The code example below demonstrates how to use the [ippiDCTFwdGetSize](#), [ippiDCTFwdInit](#), and [ippiDCTFwd](#) functions.

```
void DCT_example( void )
{
    Ipp32f Src[8*8] = {
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0
    };
    Ipp32f Dst[8*8];
    IppiSize roiSize = {8, 8};
    int srcStep;
    int dstStep;
```

```

int sizeSpec;
int sizeInit;
int sizeBuffer;

IppiDCTFwdSpec_32f *pMemSpec;
Ipp8u *pMemInit = 0;
Ipp8u *pMemBuffer = 0;

srcStep = dstStep = 8 * sizeof(Ipp32f);

/// get sizes for required buffers
ippiDCTFwdGetSize_32f( roiSize, &sizeSpec, &sizeInit, &sizeBuffer );

/// allocate memory for required buffers
pMemSpec = (IppiDCTFwdSpec_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = (Ipp8u*) ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = (Ipp8u*) ippMalloc ( sizeBuffer );
}

/// initialize DCT specification structure
ippiDCTFwdInit_32f( pMemSpec, roiSize, pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward DCT
ippiDCTFwd_32f_C1R( Src, srcStep, Dst, dstStep, pMemSpec, pMemBuffer );

/// ...

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}

ippFree( pMemSpec );
}

```

## Result:

## DCTInv

Applies an inverse discrete cosine transform to an image.

---

### Syntax

```
IppStatusippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,  
const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for `mod` :

```
32f_C1R  
32f_C3R  
32f_C4R  
32f_AC4R
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>pDCTSpec</code>	Pointer to the previously initialized inverse DCT context structure.
<code>pBuffer</code>	Pointer to the external work buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the `ippiDCTInvInit` function.

This function performs an inverse DCT on each channel of the input image `pSrc` and writes the result into the corresponding channel of the output image buffer `pDst`. Note that the function flavor with `AC4` descriptor does not process alpha channel. This function uses the previously initialized `pDCTSpec` context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

Before using the inverse DCT functions, you need to compute the size of the required buffers and the external work buffer using the `ippiDCTInvGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCT8x8Fwd

Performs a forward DCT on a 2D buffer of 8x8 size.

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for *mod*:

16s\_C1      32f\_C1

#### Case 2: Not-in-place operation with ROI

```
IppStatusippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
Ipp<dstDatatype>* pDst);
```

Supported values for *mod*:

16s\_C1R      8u16s\_C1R

#### Case 3: In-place operation

```
IppStatusippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for *mod*:

16s\_C1I      32f\_C1I

### Include Files

`ippi.h`

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image for in-place operations.

## Description

Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> value is zero or negative.

## Example

The code example below illustrates the use of `ippiDCT8x8Fwd` function.

```
IppStatus dct16s( void ) {
    Ipp16s x[64] = {0};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
       ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
        --roi.width;
        --roi.height;
    }
    returnippiDCT8x8Fwd_16s_C1I( x );
}
```

The destination image `x` contains:

```
18 -9 -2 -1 -1 0 0 0
-9 7 0 0 0 0 0 0
-2 0 2 0 0 0 0 0
-1 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

## DCT8x8Inv, DCT8x8Inv\_A10

*Performs an inverse DCT on a 2D buffer of 8x8 size.*

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for `mod`:

```
16s_C1      32f_C1
```

```
IppStatusippiDCT8x8Inv_A10_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

#### Case 2: Not-in-place operation with ROI

```
IppStatusippiDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for `mod`:

```
16s_C1R     16s8u_C1R
```

#### Case 3: In-place operation

```
IppStatusippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for `mod`:

```
16s_C1I      32f_C1I
```

```
IppStatusippiDCT8x8Inv_A10_16s_C1I(Ipp16s* pSrcDst);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
<i>pSrcDst</i>	Pointer to the source and destination image for in-place operations.

## Description

Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the inverse discrete cosine transform of data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

### Caution

Source data for 16s functions must be the result of the forward discrete cosine transform of data from the range [-512, 511] for flavors with A10 modifier (`ippiDCT8x8Inv_A10`), and from the range [-256, 255] for flavors without A10 modifier (`ippiDCT8x8Inv`); they cannot be arbitrary data from the range [-32768, 32767].

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> value is zero or negative.

## DCT8x8FwdLS

*Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.*

---

## Syntax

```
IppStatusippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst,
Ipp16s addVal);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image buffer.
-------------	-------------------------------------

---

<i>pDst</i>	Pointer to the destination buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>addVal</i>	The level shift value.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer *pSrc* from unsigned `Ipp8u` type to the signed `Ipp16s` type and then performs level shift operation by adding the constant value *addVal* to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is zero or negative.

## DCT8x8InvLSClip

*Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.*

---

## Syntax

```
IppStatusippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep,
Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>addVal</i>	The level shift value.
<i>clipDown</i>	The lower bound for the range of output values.
<i>clipUp</i>	The upper bound for the range of output values.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer *pSrc*. After completing the DCT, this function performs level shift operation by adding the constant value *addVal* to each sample. Finally, the function converts data from the signed `Ipp16s` type to the unsigned `Ipp8u` type. The output data are clipped to the range `[clipDown..clipUp]`. The result is stored in the destination buffer *pDst*.

---

### **Caution**

Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> value is zero or negative.

## DCT8x8Inv\_2x2, DCT8x8Inv\_4x4

Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.

---

## Syntax

```
IppStatusippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatusippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatusippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);
IppStatusippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for in-place operations.

## Description

These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

### Caution

Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

## DCT8x8To2x2Inv, DCT8x8To4x4Inv

*Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size.*

### Syntax

```
IppStatusippiDCT8x8To2x2Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatusippiDCT8x8To4x4Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatusippiDCT8x8To2x2Inv_16s_C1I(Ipp16s* pSrcDst);
IppStatusippiDCT8x8To4x4Inv_16s_C1I(Ipp16s* pSrcDst);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for in-place operations.

## Description

These functions compute the inverse discrete cosine transform of the 2D buffer *pSrc* of 8x8 size. Then the functions perform downsampling of the result by averaging to the destination buffer *pDst* of size 2x2 or 4x4.

In-place flavors of the functions perform operations on the source and destination buffer *pSrcDst*.

**Caution**

Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

---

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

# Image Statistics Functions

11

This chapter describes the Intel® IPP image processing functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity,  $L_1$ , and  $L_2$  norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity,  $L_1$ , and  $L_2$  norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

## Sum

*Computes the sum of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pSum);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R
--------	---------	---------

#### Case 2: Operation on one-channel floating-point data

```
IppStatusippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pSum, IppHintAlgorithm hint);
```

#### Case 3: Operation on multi-channel integer data

```
IppStatusippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f sum[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
--------	---------	---------

```
IppStatusippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f sum[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 4: Operation on multi-channel floating-point data**

```
IppStatusippiSum_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
```

```
IppStatusippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[4], IppHintAlgorithm hint);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pSum</i>	Pointer to the computed sum of pixel values.
<i>sum</i>	Array containing computed sums of channel values of pixels in the source buffer.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the sum of pixel values *pSum* for the source image *pSrc* using algorithm indicated by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In case of a multi-channel image, the sum is computed over each channel and stored in the array *sum*.

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the <i>pSrc</i> or <i>pSum</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below demonstrates the use of `ippiSum` function:

```
IppStatus sum( void ) {
    Ipp64f sum;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiset_8u_C1R( 1, x, 5, roi );
    return ippisum_8u_C1R( x, 5, roi, &sum);
}
```

## Integral

*Transforms an image to the integral representation.*

### Syntax

```
IppStatus ippintegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize srcRoiSize, Ipp32s val);
IppStatus ippintegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize srcRoiSize, Ipp32f val);
IppStatus ippintegral_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize srcRoiSize);
```

### Include Files

`ippcv.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>srcRoiSize</i>	Size of source and destination image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transforms a source image  $pSrc$  to the integral image  $pDst$ . Pixel values of the destination image  $pDst$  are computed using pixel values of the source image  $pSrc$  and the specified value  $val$  in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

where  $i, j$  are coordinates of the destination image pixels (see Figure 11-1) varying in the range  $i = 1, \dots, srcRoiSize.height$ ,  $j = 0, \dots, srcRoiSize.width$ . Pixel values of zero row and column of  $pDst$  ( $i=0$ ) is set to  $val$ .

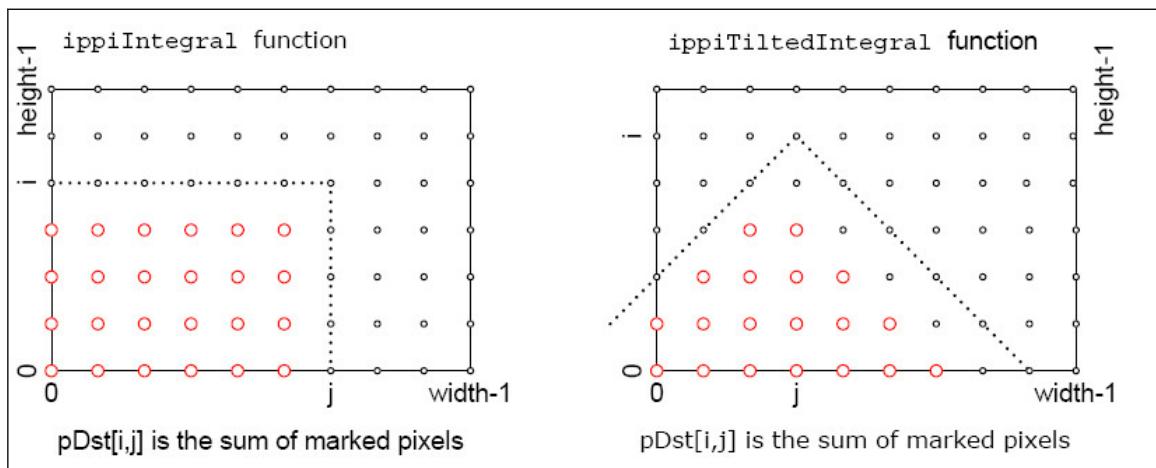
For the `ippiIntegral_32f_C1` function flavor the value of  $val$  is considered to be equal to zero.

The size of the destination images is  $(srcRoiSize.width + 1) \times (srcRoiSize.height + 1)$ .

[Figure “Operation of the Integral and TiltedIntegral functions”](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

For large images the result of summation can exceed the upper bound of the output data type. [Table “Maximum Image Size for Integral Functions”](#) lists the maximum image size for different function flavors and values.

## Operation of the Integral and TiltedIntegral functions



## Maximum Image Size for Integral Functions

Function Flavor	Value $val$	Maximum Image Size
<code>ippiIntegral_8u32s_C1R</code>	0	$(2^{31}-1)/255$
	$-2^{31}$	$2^{32}/255$
<code>ippiIntegral_8u32f_C1R</code>	0	$2^{24}/255$
	$-2^{24}$	$(2^{25}+1)/255$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if $pSrc$ or $pDst$ is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if $srcRoiSize$ has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>(srcRoiSize.width+1) * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is not divisible by <i>&lt;pixelSize&gt;</i> .

## Example

The code example below demonstrates how to use the `ippiIntegral_8u32s_C1R` function:

```
void func_integral_8u32s_C1R()
{
    Ipp8u pSrc[5*4];
    Ipp32s pDst[6*5];
    IppiSize ROI = {5,4};
   ippiSet_8u_C1R(1,pSrc,5,ROI);
    Ipp32s val = 1;
   ippiIntegral_8u32s_C1R(pSrc, 5, pDst, 6*sizeof(Ipp32s), ROI, val);
}
```

Result:

pSrc ->	1 1 1 1 1	pDst ->	1 1 1 1 1
	1 1 1 1 1		1 2 3 4 5 6
	1 1 1 1 1		1 3 5 7 9 11
	1 1 1 1 1		1 4 7 10 13 16
			1 5 9 13 17 21

The code example below demonstrates how to use the `ippiIntegral_32f_C1R` function:

```
void func_integral_32f_C1R()
{
    Ipp32f pSrc[5*4];
    Ipp32f pDst[6*5];
    IppiSize ROI = {5, 4};

   ippiSet_32f_C1R(1, pSrc, 5*sizeof(Ipp32f), ROI);
   ippiIntegral_32f_C1R(pSrc, 5*sizeof(Ipp32f), pDst, 6*sizeof(Ipp32f), ROI);
}
```

Result:

pSrc ->	1.0 1.0 1.0 1.0 1.0	pDst ->	0.0 0.0 0.0 0.0 0.0 0.0
	1.0 1.0 1.0 1.0 1.0		0.0 1.0 2.0 3.0 4.0 5.0
	1.0 1.0 1.0 1.0 1.0		0.0 2.0 4.0 6.0 8.0 10.0
	1.0 1.0 1.0 1.0 1.0		0.0 3.0 6.0 9.0 12.0 15.0
			0.0 4.0 8.0 12.0 16.0 20.0

## SqrIntegral

Transforms an image to integral and integral of pixel squares representations.

### Syntax

```
IppStatusippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);

IppStatusippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

```
IppStatusippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSqr</i>	Pointer to the ROI of the destination integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.
<i>valSqr</i>	The value to add to <i>pSqr</i> image pixels

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination images: integral image *pDst* and integral image of pixel squares *pSqr*. Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{k < i} \sum_{l < j} pSrc[k, l]^2$$

where *i,j* are coordinates of the destination image pixels (see [Figure "Operation of the Integral and TiltedIntegral functions"](#)) varying in the range *i* = 1, ..., *roiSize.height*, *j* = 0,..., *roiSize.width*. Pixel values of zero row and column are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 1) x (*roiSize.height* + 1).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the *i,j* coordinates.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> or <code>sqrStep</code> is less than <code>(roiSize.width+1) * &lt;pixelSize&gt;</code> .
ippStsNotEvenStepErr	Indicates an error condition if <code>dstStep</code> is not divisible by 4, or <code>sqrStep</code> is not divisible by 8.

## TiltedIntegral

Transforms an image to the tilted integral representation.

### Syntax

```
IppStatusippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize roiSize, Ipp32f val);
IppStatusippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, IppiSize roiSize, Ipp32s val);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ipcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ipcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination integral image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of source image ROI in pixels.
<code>val</code>	The value to add to pDst image pixels

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transforms a source image `pSrc` to the tilted integral image `pDst`. Pixel values of the destination image `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, 1]$$

where  $i, j$  are coordinates of the destination image pixels (see [Figure "Operation of the Integral and TiltedIntegral functions"](#)) varying in the range  $i = 2, \dots, roiSize.height + 1$ ,  $j = 0, \dots, roiSize.width + 1$ . Pixel values of rows 0 and 1 of the destination image  $pDst$  ( $i=0$ ) is set to  $val$ .

The size of the destination images is  $(roiSize.width + 2) \times (roiSize.height + 2)$ .

[Figure "Operation of the Integral and TiltedIntegral functions"](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if $pSrc$ or $pDst$ is NULL.
ippStsSizeErr	Indicates an error condition if $roiSize$ has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if $srcStep$ is less than $roiSize.width * <pixelSize>$ , or $dstStep$ is less than $(roiSize.width+2) * <pixelSize>$ .
ippStsNotEvenStepErr	Indicates an error condition if one $dstStep$ is not divisible by 4.

## TiltedSqrIntegral

---

Transforms an image to tilted integral and tilted integral of pixel squares representations.

---

### Syntax

```
IppStatusippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);
IppStatusippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);
IppStatusippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

*pSrc* Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSqr</i>	Pointer to the ROI of the destination integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.
<i>valSqr</i>	The value to add to <i>pSqr</i> image pixels

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination image: tilted integral image *pDst* and tilted integral image of pixel squares *pSqr*.

Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]^2$$

where *i,j* are coordinates of the destination image pixels (see [Figure "Operation of the Integral and TiltedIntegral functions"](#)) varying in the range *i* = 2, ..., *roiSize.height*, *j* = 0, ..., *roiSize.width*. Pixel values of zero and first rows (*i*=0,1) are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 2) x (*roiSize.height* + 2).

[Figure "Operation of the Integral and TiltedIntegral functions"](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the *i,j* coordinates.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> or <i>sqrStep</i> is less than ( <i>roiSize.width</i> +2) * <i>&lt;pixelSize&gt;</i> .
<i>ippStsNotEvenStepErr</i>	Indicates an error condition if <i>dstStep</i> is not divisible by 4, or <i>sqrStep</i> is not divisible by 8.

## Mean

*Computes the mean of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pMean);
```

Supported values for mod:

8u\_C1R        16u\_C1R        16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatusippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pMean, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatusippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for mod:

8u\_C1MR        16u\_C1MR        32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatusippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f mean[3]);
```

Supported values for mod:

8u\_C3R        16u\_C3R        16s\_C3R

```
IppStatusippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f mean[4]);
```

Supported values for mod:

8u\_C4R        16u\_C4R        16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatusippiMean_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[3], IppHintAlgorithm hint);
```

Supported values for mod:

32f\_C3R

```
IppStatusippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[4], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatusippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,  
int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);
```

Supported values for `mod`:

`8u_C3CMR`      `16u_C3CMR`      `32f_C3CMR`

### Include Files

`ippi.h`

`ippcv.h`

### Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels.
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pMean</code>	Pointer to the computed mean of pixel values.
<code>mean</code>	Array containing computed mean values for each channel of a multi-channel image.
<code>hint</code>	Option to select the algorithmic implementation of the function.

### Description

The flavors of the function `ippiMean` that perform masked operations are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the mean (average) of pixel values `pMean` for the source image `pSrc`. Computation algorithm is specified by the `hint` argument (see [Table "Hint Arguments for Image Moment Functions"](#)). For non-masked operations on a multi-channel image (Case 4, 5), the mean is computed over each channel and stored in the array `mean`. In the mask multi-channel mode (Case 6), the mean is computed for a single channel of interest specified by `coi`.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below shows how to use the `ippiMean` function.

```
IppStatus mean( void ) {
    Ipp64f mean;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
   ippiSet_8u_C1R( 3, x, 5, roi );
    returnippiMean_8u_C1R( x, 5, roi, &mean );
}
```

## Mean\_StdDev

*Computes the mean and standard deviation of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize  
roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

8u\_C1R      16u\_C1R      32f\_C1R

#### Case 2: Masked operation on one-channel data

```
IppStatusippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*  
pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

8u\_C1MR      16u\_C1MR      32f\_C1MR

### Case 3: Operation on multi-channel data

```
IppStatusippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize
roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

`8u_C3CR`      `16u_C3CR`      `32f_C3CR`

### Case 4: Masked operation on multi-channel data

```
IppStatusippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

`8u_C3CMR`      `16u_C3CMR`      `32f_C3CMR`

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pMean</code>	Pointer to the computed mean of pixel values.
<code>pStdDev</code>	Pointer to the computed standard deviation of pixel values in the image.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the mean and standard deviation of pixel values in the ROI of the source image `pSrc`. In the mask mode, the computation is done over pixels selected by nonzero mask values. In the multi-channel mode, the mean is computed for a single channel of interest specified by `coi`. If any of the parameters `pMean` or `pStdDev` is not required, the zero pointer is to be passed to the corresponding parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> or <code>pMask</code> pointer is <code>NULL</code> .

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## RectStdDev

---

*Computes the standard deviation of the integral images.*

---

### Syntax

```
IppStatusippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr,
int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
IppStatusippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f* pSqr,
int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
IppStatusippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr,
int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source integral image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source integral image.
<i>pSqr</i>	Pointer to the ROI in the source integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of destination image ROI in pixels.
<i>rect</i>	Rectangular window.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window *rect* using the integral image *pSrc* and integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i, j* are coordinates of the destination image pixels varying in the range *i* = 0,..., *roiSize.height* - 1, *j* = 0,..., *roiSize.width* - 1;

*sum* = *pSrc*[ *i* + *rect.y* + *rect.height*, *j* + *rect.x* + *rect.width*] - *pSrc*[ *i* + *rect.y*, *j* + *rect.x* + *rect.width*] - *pSrc*[ *i* + *rect.y* + *rect.height*, *j* + *rect.x*] + *pSrc*[ *i* + *rect.y*, *j* + *rect.x*];

*sumSqr* = *pSqr*[ *i* + *rect.y* + *rect.height*, *j* + *rect.x* + *rect.width*] - *pSqr*[ *i* + *rect.y*, *j* + *rect.x* + *rect.width*] - *pSqr*[ *i* + *rect.y* + *rect.height*, *j* + *rect.x*] + *pSqr*[ *i* + *rect.y*, *j* + *rect.x*];

*numPix* = *rect.height* \* *rect.width*.

The minimum size of each source images *pSrc* and *pSqr* should be (*roiSize.width* + *rect.x* + *rect.width*) x (*roiSize.height* + *rect.y* + *rect.height*).

The source images *pSrc* and *pSqr* can be obtained by using the functions [ippiIntegral](#) or [ippiSqrIntegral](#).

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsSizeErr	Indicates an error condition if <i>rect.width</i> or <i>rect.height</i> is less than or equal to zero, or if <i>rect.x</i> or <i>rect.y</i> is less than zero.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>sqrStep</i> is less than ( <i>roiSize.width</i> + <i>rect.x</i> + <i>rect.width</i> +1) * <pixelSize>, or <i>dstStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if <i>sqrStep</i> is not divisible by 8, or one of <i>pSrc</i> and <i>dstStep</i> is not divisible by 4.

## TiltedRectStdDev

Computes the standard deviation of the tilted integral images.

### Syntax

```
IppStatusippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
IppStatusippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
```

```
IppStatusippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const
Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the source integral image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source integral image.
<i>pSqr</i>	Pointer to the ROI in the source integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of destination image ROI in pixels.
<i>rect</i>	Rectangular window.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window *rect* using the tilted integral image *pSrc* and tilted integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{\text{sumSqr} \cdot \text{numPix} - \text{sum}^2}{\text{numPix}^2}\right)}$$

where *i, j* are coordinates of the destination image pixels varying in the range *i* = 0,..., *roiSize.height* - 1, *j* = 0,..., *roiSize.width* - 1;

*sum* = *pSrc*[ *i* + *rect.x* - *rect.y* + *rect.height* + *rect.width*, *j* + *rect.x* + *rect.y* - *rect.height* + *rect.width*] - *pSrc*[ *i* + *rect.x* - *rect.y* + *rect.width*, *j* + *rect.x* + *rect.y* + *rect.width*] - *pSrc*[ *i* + *rect.x* - *rect.y* + *rect.height*, *j* + *rect.x* - *rect.y* - *rect.height*] + *pSrc*[ *i* + *rect.x* - *rect.y*, *j* + *rect.x* + *rect.y*];

*sumSqr* = *pSqr*[ *i* + *rect.x* - *rect.y* + *rect.height* + *rect.width*, *j* + *rect.x* + *rect.y* - *rect.height* + *rect.width*] - *pSqr*[ *i* + *rect.x* - *rect.y* + *rect.width*, *j* + *rect.x* + *rect.y* + *rect.width*] - *pSqr*[ *i* + *rect.x* - *rect.y* + *rect.height*, *j* + *rect.x* - *rect.y* - *rect.height*] + *pSqr*[ *i* + *rect.x* - *rect.y*, *j* + *rect.x* + *rect.y*];

*numPix* = 2 \* *rect.height* \* *rect.width*.

The minimum size of each source images *pSrc* and *pSqr* should be  $(roiSize.width + rect.height + rect.width - 2) \times (roiSize.height + rect.x + rect.y + rect.height + rect.width - 2)$ .

The source images *pSrc* and *pSqr* can be obtained by using the functions [ippiTiltedIntegral](#) or [ippiTiltedSqrIntegral](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>rect.width</i> or <i>rect.height</i> is less than or equal to zero, or if <i>rect.x</i> or <i>rect.y</i> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>sqrStep</i> is less than $(roiSize.width+rect.x+rect.width+1) * <\text{pixelSize}>$ , or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>sqrStep</i> is not divisible by 8, or one of <i>pSrc</i> and <i>dstStep</i> is not divisible by 4.

## HistogramGetBufferSize

Computes the size of the specification structure and work buffer for the `ippiHistogram` function.

### Syntax

```
IppStatusippiHistogramGetBufferSize(IppDataType dataType, IppiSize roiSize, const int nLevels[], int numChannels, int uniform, int* pSpecSize, int* pBufferSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>dataType</i>	Data type of the source image. Supported values are: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>nLevels</i>	Number of level values. Each channel has a separate number of levels.
<i>numChannels</i>	Number of image channels. Supported values are: 1, 3, and 4.
<i>uniform</i>	Type of levels distribution: 0 - with random step, 1 - with uniform step.

<i>pSpecSize</i>	Pointer to the computed value of the specification structure size, in bytes.
<i>pBufferSize</i>	Pointer to the computed value of the external buffer size.

## Description

The `ippiHistogramGetBufferSize` function computes the size of the histogram specification structure and the size of the external work buffer (in bytes) needed for the [Histogram](#) function.

For an example on how to use this function, refer to the example provided with the [Histogram](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> is less than, or equal to zero.
<code>ippStsHistoNofLevelsErr</code>	Indicates an error when the number of levels is less than 2.
<code>ippStsNumChannelsErr</code>	Indicates an error when the <code>numChannels</code> value differs from 1, 3, or 4.
<code>ippStsDataTypeErr</code>	Indicates an error when the <code>dataType</code> value differs from <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , or <code>ipp32f</code> .

## See Also

[Histogram](#) Computes the intensity histogram of an image.

## HistogramGetLevels

---

*Returns the array with level values stored in the specification structure.*

## Syntax

```
IppStatusippiHistogramGetLevels(const IppiHistogramSpec* pSpec, Ipp32f* pLevels[]);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSpec</i>	Pointer to the specification structure.
<i>pLevels</i>	Pointer to the array of pointers to the level values vectors for each channel.

## Description

The `ippiHistogramGetLevels` function returns the level values stored in the histogram specification structure.

For an example on how to use this function, refer to the example provided with the [Histogram](#) function description.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is <code>NULL</code> .
ippStsBadArgErr	Indicates an error when the <code>pSpec</code> object is not initialized.

## See Also

[Histogram](#) Computes the intensity histogram of an image.

# HistogramInit, HistogramUniformInit

*Initializes the specification structure for the ippiHistogram function.*

## Syntax

```
IppStatus ippiHistogramInit(IppDataType dataType, const Ipp32f* pLevels[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
IppStatus ippiHistogramUniformInit(IppDataType dataType, Ipp32f lowerLevel[], Ipp32f upperLevel[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<code>dataType</code>	Data type of the source image. Supported values are: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<code>pLevels</code>	Pointer to the array of pointers to the level values vectors for each channel.
<code>lowerLevel</code>	Lower levels for uniform histogram, separate for each channel.
<code>upperLevel</code>	Upper levels for uniform histogram, separate for each channel.
<code>nLevels</code>	Number of level values. Each channel has a separate number of levels.
<code>numChannels</code>	Number of image channels. Supported values are: 1, 3, and 4.
<code>pSpec</code>	Pointer to the specification structure.

## Description

The `ippiHistogramInit` function initializes the specification structure for the [Histogram](#) function.

For an example on how to use these functions, refer to the example provided with the [Histogram](#) function description.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> is less than, or equal to zero.
ippStsHistoNofLevelsErr	Indicates an error when the number of levels is less than 2.
ippStsNumChannelsErr	Indicates an error when the <i>numChannels</i> value differs from 1, 3, or 4.
ippStsDataTypeErr	Indicates an error when the <i>dataType</i> value differs from <i>ipp8u</i> , <i>ipp16u</i> , <i>ipp16s</i> , or <i>ipp32f</i> .

## See Also

[Histogram](#) Computes the intensity histogram of an image.

# Histogram

*Computes the intensity histogram of an image.*

## Syntax

### Case 1: One-channel data

```
IppStatusippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist, const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Three-channel data

```
IppStatusippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist[3], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

### Case 3: Four-channel data

```
IppStatusippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist[4], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>pHist</i>	Pointer to the computed histogram. In case of multi-channel data, <i>pHist</i> is an array of pointers to the histogram for each channel.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The `ippiHistogram` function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the intensity histogram for each channel of the source image and stores the result in the *pHist* array.

Before calling this function, initialize the specification structure using the [HistogramInit](#) or [HistogramUniformInit](#) functions. The specification structure defines the following parameters for histogram calculation:

- Histogram type: with uniform or random levels step
- Number of levels
- Level values

Length of the *pHist* array is defined by the *nLevels* parameter passed to the [HistogramInit](#) or [HistogramUniformInit](#) function.

As *nLevels* is the number of levels, the number of values in the *pHist* array, which is the number of histogram bins, is *nLevels* - 1. The meaning of the *pHist* and *pLevels* values can be illustrated by the following example: *pHist*[*k*] is the number of the source image pixels *pSrc*(*x*, *y*) that satisfy the condition *pLevels*[*k*] ≤ *pSrc*(*x*, *y*) < *pLevels*(*k*+1).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> is less than <i>roiSize.width*sizeOf(*pSrc)*nChannels</i> .
<code>ippStsBadArgErr</code>	Indicates an error when the <i>pSpec</i> object is not initialized.

## Example

The code example below demonstrates how to use the [HistogramGetSize](#), [HistogramUniformInit](#), [HistogramGetLevels](#), and [Histogram](#) functions.

```
void HistogramExample()
{
    const int HEIGHT = 8;
```

```
const int WIDTH = 8;
Ipp8u pImg[WIDTH*HEIGHT];
IppiSize roi = {WIDTH, HEIGHT};
int i;
IppStatus sts;

{ // fill image with random values in [0..255] range with uniform distribution.
    IppsRandUniState_8u* pRndObj;
    int sizeRndObj;

    // get spec size
    ippsRandUniformGetSize_8u( &sizeRndObj );
    pRndObj = (IppsRandUniState_8u*)ippsMalloc_8u( sizeRndObj );
    // initialize rnd spec
    ippsRandUniformInit_8u(pRndObj, 0/*low*/, 255/*high*/, 0/*seed*/ );

    // fill image
    for ( i=0; i<HEIGHT; i++ ) {
        sts = ippsRandUniform_8u(pImg + i*WIDTH, WIDTH, pRndObj);
    }

    ippsFree( pRndObj );
}

printf_8u_2D("pImg:", pImg, roi, WIDTH, sts);

{
    const int nBins = 5;
    int nLevels[] = { nBins+1 };
    Ipp32f lowerLevel[] = {0};
    Ipp32f upperLevel[] = {256};
    Ipp32f pLevels[nBins+1], *ppLevels[1];
    int sizeHistObj, sizeBuffer;

    IppiHistogramSpec* pHistObj;
    Ipp8u* pBuffer;
    Ipp32u pHistVec[nBins];

    // get sizes for spec and buffer
   ippiHistogramGetBufferSize(ipp8u, roi, nLevels, 1/*nChan*/, 1/*uniform*/, &sizeHistObj,
&sizeBuffer);

    pHistObj = (IppiHistogramSpec*)ippsMalloc_8u( sizeHistObj );
    pBuffer = (Ipp8u*)ippsMalloc_8u( sizeBuffer );
    // initialize spec
   ippiHistogramUniformInit( ipp8u, lowerLevel, upperLevel, nLevels, 1, pHistObj );

    // check levels of bins
    ppLevels[0] = pLevels;
    sts = ippiHistogramGetLevels( pHistObj, ppLevels );
    printf_32f( "pLevels:", pLevels, nBins+1, sts );

    // calculate histogram
    sts = ippiHistogram_8u_C1R( pImg, WIDTH, roi, pHistVec, pHistObj, pBuffer );

    ippsFree( pHistObj );
    ippsFree( pBuffer );
}
```

```

        printf_32u( "Histogram:", pHistVec, nBins, sts );
    }
}

```

**Output:**

```

pImg:
0 33 53 102 90 188 210 60
195 137 247 137 7 15 65 244
149 44 210 20 170 140 183 144
133 61 191 32 212 108 178 89
86 30 54 93 168 93 2 114
30 145 216 42 86 113 148 205
148 181 217 99 219 31 156 156
237 36 74 80 208 121 118 106

pLevels:
0.0 51.0 102.0 153.0 204.0 255.0

Histogram:
13 14 16 10 11

```

**See Also**[Regions of Interest in Intel IPP](#)

**Histogram** Computes the intensity histogram of an image.

**HistogramGetSize** Computes the size of the specification structure and work buffer for the `ippiHistogram` function.

**HistogramGetLevels** Returns the array with level values stored in the specification structure.

**HistogramInit, HistogramUniformInit** Initializes the specification structure for the `ippiHistogram` function.

## CountInRange

*Computes the number of pixels within the given intensity range.*

**Syntax****Case 1: Operation on one-channel data**

```
IppStatusippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, int* counts, Ipp<datatype> lowerBound, Ipp<datatype> upperBound);
```

Supported values for `mod`:

8u\_C1R      32f\_C1R

**Case 2: Operation on multi-channel data**

```
IppStatusippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, int counts[3], Ipp<datatype> lowerBound[3], Ipp<datatype> upperBound[3]);
```

Supported values for `mod`:

8u\_C3R      32f\_C3R

8u\_AC4R      32f\_AC4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>counts</i>	The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.
<i>lowerBound</i>	Lower limit of the intensity range.
<i>upperBound</i>	Upper limit of the intensity range.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the number of pixels in the image which have intensity values in the range between *lowerBound* and *upperBound* (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array *counts* of three resulting values is returned. The alpha channel values, if present, are not processed.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> has a zero or negative value.
ippStsRangeErr	Indicates an error condition if <i>lowerBound</i> exceeds <i>upperBound</i> .

## BlockMinMax

*Finds minimum and maximum values for blocks of an image.*

---

## Syntax

```
IppStatusippiBlockMinMax_<dataType>_C1R(const Ipp<dataType>* pSrc, int srcStep,
IppSize srcSize, Ipp<dataType>* pDstMin, int dstMinStep, Ipp<dataType>* pDstMax, int
dstMaxStep, IppSize blockSize, Ipp<dataType>* pGlobalMin, Ipp<dataType>* pGlobalMax);
```

Supported values for dataType:

8u	16u	16s	32f
----	-----	-----	-----

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcSize</i>	Size, in pixels, of the source image.
<i>pDstMin</i>	Pointer to the destination image to store minimum values per block.
<i>dstMinStep</i>	Distance, in bytes, between the starting points of consecutive lines in the <i>pDstMin</i> image.
<i>pDstMax</i>	Pointer to the destination image to store maximum values per block.
<i>dstMaxStep</i>	Distance, in bytes, between the starting points of consecutive lines in the <i>pDstMax</i> image.
<i>blockSize</i>	Size, in pixels, of the image block.
<i>pGlobalMin</i>	Destination pointer to the minimum value for the entire source image.
<i>pGlobalMax</i>	Destination pointer to the maximum value for the entire source image.

## Description

This function operates with ROI.

This function finds minimum and maximum values for blocks of the source image, which are defined by the *blockSize* parameter. Minimum and maximum values for blocks are stored in the *pDstMin* and *pDstMax* images, respectively. Minimum and maximum values for the entire image are stored in the *pGlobalMin* and *pGlobalMax* pointers, respectively.

If *pDstMin* or *pDstMax* pointer is NULL, the corresponding component (minimum or maximum value) is not calculated.

The size of the *pDstMin* and *pDstMax* images is calculated by the following formulae:

- if *srcWidth* is divisible by *blockWidth*, the destination width is equal to:

*dstWidth=srcWidth/blockWidth*

otherwise:

$dstWidth = srcWidth / blockHeight + 1$

- if  $srcHeight$  is divisible by  $blockHeight$ , the destination height is equal to:

$dstHeight = srcHeight / blockHeight$

otherwise:

$dstHeight = srcHeight / blockHeight + 1$

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when $pSrc$ , $pDstMin$ , and $pDstMax$ pointers are NULL.
ippStsStepErr	Indicates an error when: <ul style="list-style-type: none"> <li>• <math>srcStep</math> is less than <math>srcSize.width * &lt;pixelSize&gt;</math></li> <li>• <math>dstMinStep</math> or <math>dstMaxStep</math> is less than <math>dstSize.width * &lt;pixelSize&gt;</math></li> </ul>
ippStsSizeErr	Indicates an error when $srcSize$ or $blockSize$ has a zero or negative value.

## See Also

[Regions of Interest in Intel IPP](#)

# Min

---

*Computes the minimum of image pixel values.*

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Operation on multi-channel data

```
IppStatusippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pMin</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiMin`.

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u min;

ippiMin_8u_C1R ( src, 4, roiSize, &min );

result: min = 20
```

## MinIdx

*Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiMinIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Operation on multi-channel data

```
IppStatusippiMinIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3], int indexX[3], int indexY[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiMinIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4], int indexX[4], int indexY[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with minimum value.

*indexX, indexY*

Arrays containing the x and y coordinates of pixels with minimum channel values.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*. The function also retrieves the *x* and *y* coordinates of pixels on which the minimum is reached. If several pixels have equal minimum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX[k]* and *indexY[k]* are the *x* and *y* coordinates of the pixel that has the minimal intensity value of the *k*-th channel, *k* = 1,2,3,4.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of <i>pSrc</i> , <i>pMin</i> , <i>pIndexX</i> , or <i>pIndexY</i> pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Max

*Computes the maximum of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMax);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------	----------------------

#### Case 2: Operation on multi-channel data

```
IppStatusippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3]);
```

Supported values for *mod*:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

```
IppStatusippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4]);
```

Supported values for *mod*:

<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
---------------------	----------------------	----------------------	----------------------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSrc</i> or <i>pMax</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## MaxIdx

---

*Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.*

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      16s\_C1R      32f\_C1R

## Case 2: Operation on multi-channel data

```
IppStatusippiMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3], int indexX[3], int indexY[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4], int indexX[4], int indexY[4]);
```

Supported values for mod:

8u_C4R	16s_C4R	16u_C4R	32f_C4R
--------	---------	---------	---------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with maximum value.
<i>indexX</i> , <i>indexY</i>	Arrays containing the x and y coordinates of pixels with maximum channel values.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*. The function also retrieves the *x* and *y* coordinates of pixels on which the maximum is reached. If several pixels have equal maximum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX[k]* and *indexY[k]* are the *x* and *y* coordinates of the pixel that has the maximal intensity value of the *k-th* channel, *k* = 1,2,3,4.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
-------------	--

ippStsNullPtrErr	Indicates an error condition if any of <i>pSrc</i> , <i>pMax</i> , <i>pIndexX</i> , or <i>pIndexY</i> pointers is <b>NULL</b> .
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Example

The code example below demonstrates how to use the function `ippiMaxIdx`.

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };

IppiSize roiSize = { 4, 1 };

Ipp8u max;

int IndexX;

int IndexY;

ippiMaxIdx_8u_C1R ( src, 4, roiSize, &max, &IndexX, &IndexY );

result: max = 80  IndexX = 3 IndexY = 0
```

## MinMax

---

*Computes the minimum and maximum of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp<datatype>* pMin, Ipp<datatype>* pMax);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      16s\_C1R      32f\_C1R

#### Case 2: Operation on multi-channel data

```
IppStatusippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

Supported values for *mod*:

8u\_C3R      16u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatusippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

Supported values for *mod*:

8u\_C4R      16u\_C4R      16s\_C4R      32f\_C4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin, pMax</i>	Pointers to the minimum and maximum pixel values (for one-channel data).
<i>min, max</i>	Arrays containing minimum and maximum channel values of pixels in the source buffer (for multi-channel data).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum and maximum pixel values *pMin* and *pMax* for the source image *pSrc*. In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays *min* and *max*.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if <i>pSrc</i> , <i>pMin</i> , or <i>pMax</i> pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## MinMaxIdx

*Calculates minimum and maximum pixel values and their indexes in selected image rectangle.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiMinMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize  
roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      32f\_C1R

**Case 2: Masked operation on one-channel data**

```
IppStatusippiMinMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*  
pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*  
pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C1MR      16u\_C1MR      32f\_C1MR

**Case 3: Operation on multi-channel data**

```
IppStatusippiMinMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize  
roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint*  
pMaxIndex);
```

Supported values for mod:

8u\_C3CR      16u\_C3CR      32f\_C3CR

**Case 4: Masked operation on multi-channel data**

```
IppStatusippiMinMaxIdx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*  
pMask, int maskStep, IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,  
IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMinVal</i>	Pointer to the variable that returns the value of the minimum pixel.
<i>pMaxVal</i>	Pointer to the variable that returns the value of the maximum pixel.

---

<i>pMinIndex</i>	Pointer to the variable that returns the index of the minimum value found.
<i>pMaxIndex</i>	Pointer to the variable that returns the index of the maximum value found.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values. If there are several minima and maxima in the selected area, the function returns the top leftmost positions. If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns  $\{minIndex, maxIndex\} = \{0, 0\}$ ,  $minVal=maxVal=0$ . If any of the parameters *pMinVal*, *pMaxVal*, *pMinIndex*, or *pMaxIndex* is not required, the zero pointer is to be passed to the corresponding parameter.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if <i>pSrc</i> or <i>pMask</i> pointer is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error for masked operations when <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<i>ippStsNotEvenStepErr</i>	Indicates an error when steps for floating-point images cannot be divided by 4.
<i>ippStsCOIErr</i>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## MaxEvery

*Computes maximum value for each pair of pixels of two images.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiMaxEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int
src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiMaxEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2,
int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiMaxEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation

```
IppStatusippiMaxEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointer to the first and second source image, respectively (for not-in-place operation).
<i>src1Step</i> , <i>src2Step</i>	Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).
<i>pDst</i>	Pointer to the destination image (for not-in-place operation).
<i>pSrc</i>	Pointer to the first source image ROI (for in-place operation).
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the first source image (for in-place operation).
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI (for in-place operation).
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the second source and destination image (for in-place operation).
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

### Not-in-place operation:

This function computes the maximum value for each pair of the corresponding pixels of two source images (*pSrc1* and *pSrc2* for not-in-place operation or *pSrc* and *pSrcDst* for in-place), and stores the result in *pDst*.

### In-place operation:

This function computes the maximum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in *pSrcDst*:

$$pSrcDst(i, j) = \max(pSrc(i, j), pSrcDst(i, j)).$$

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
-------------	--

---

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i> or <i>srcDstStep</i> is less than <i>roiSize.width*pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

## MinEvery

*Computes minimum value for each pair of pixels of two images.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatusippiMinEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int
src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiMinEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2,
int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatusippiMinEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation

```
IppStatusippiMinEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*>
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointer to the first and second source image, respectively (for not-in-place operation).
-----------------------------	--

<i>src1Step</i> , <i>src2Step</i>	Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).
<i>pDst</i>	Pointer to the destination image (for not-in-place operation).
<i>pSrc</i>	Pointer to the first source image ROI (for in-place operation).
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the first source image (for in-place operation).
<i>pSrcDst</i>	Pointer to the second source and destination image ROI (for in-place operation).
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the second source and destination image (for in-place operation).
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

### Not-in-place operation:

This function computes the minimum value for each pair of the corresponding pixels of two source images (*pSrc1* and *pSrc2* for not-in-place operation or *pSrc* and *pSrcDst* for in-place), and stores the result in *pDst*.

### In-place operation:

This function computes the minimum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in *pSrcDst*:

$$pSrcDst(i, j) = \min(pSrc(i, j), pSrcDst(i, j)).$$

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i> or <i>srcDstStep</i> is less than <i>roiSize.width*pixelSize</i> .
<i>ippStsNotEvenStepErr</i>	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

## FindPeaks3x3GetBufferSize

---

Computes the size of the working buffer for the peak search.

---

### Syntax

```
IppStatusippiFindPeaks3x3GetBufferSize_32s_C1R(int roiWidth, int* pBufferSize);
```

---

```
IppStatusippiFindPeaks3x3GetBufferSize_32f_C1R(int roiWidth, int* pBufferSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiWidth</i>	Maximum width of the image, in pixels.
<i>pBufferSize</i>	Pointer to the size of the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the function [ippiFindPeaks3x3](#). The buffer with the length *pBufferSize[0]* can be used to filter images with width that is less than or equal to *roiWidth*.

[Example 11-8](#) shows how to use the function [ippiFindPeaks3x3GetBufferSize\\_32f\\_C1R](#).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiWidth</i> is less than 1.

## FindPeaks3x3

---

*Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value.*

## Syntax

```
IppStatusippiFindPeaks3x3_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize roiSize,
Ipp32s threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm,
int border, Ipp8u* pBuffer);
```

```
IppStatusippiFindPeaks3x3_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32f threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm,
int border, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the first source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the first source image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>threshold</i>	Threshold value.
<i>pPeak</i>	Pointer to the coordinates peaks [ <i>maxPeakCount</i> ].
<i>maxPeakCount</i>	Maximum number of peaks.
<i>pPeakCount</i>	Pointer to the number of the detected peaks.
<i>border</i>	Border value, only pixel with distance from the edge of the image greater than <i>border</i> are processed.
<i>norm</i>	Specifies type of the norm to form the mask for extremum search:  ippiNormInf                  Infinity norm (8-connectivity, 3x3 rectangular mask); ippiNormL1                  L1 norm (4-connectivity, 3x3 cross mask).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function detects local maximum and minimum pixels in the source image:

$$pSrc(i_m, j_m) = \max_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad pSrc(i_m, j_m) \geq threshold$$

$$pSrc(i_m, j_m) = \min_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad |pSrc(i_m, j_m)| \geq threshold$$

and stores their coordinates in the *pPeak* array *pPeak[m].x = jm*, *pPeak[m].y = im*, *m = 0, ..., pPeakCount[0]*, *pPeakCount[0] ≤ maxPeakCount*

The neighborhood  $O(i, j)$  for the extremum search is defined by the parameter *norm*. The number of detected extrema is returned in *pPeakCount[0]*. The operation is stopped when the *maxPeakCount* extrema are found.

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFindPeaks3x3GetBufferSize](#) beforehand.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.

---

ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or if <i>maxPeakCount</i> is less than or equal to 0; or if <i>border</i> is less than 1 or greater than one of $0.5 * \text{roiSize}.width$ or of $0.5 * \text{roiSize}.height$ .
ippStsStepErr	Indicates an error condition if <i>srcStep</i> is less than $\text{roiSize}.width * \langle \text{pixelSize} \rangle$ .
ippStsNotEvenStepErr	Indicates an error condition if <i>srcStep</i> is not divisible by 4.

## Example

To better understand usage of this function, refer to the `FindPeaks3x3.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Image Moments

Spatial and central moments are important statistical properties of an image. The spatial moment  $M_U(m, n)$  of order  $(m, n)$  is defined as follows:

$$M_U(m, n) = \sum_{j, k} x_k^m y_j^n p_{j, k}$$

where the summation is performed for all rows and columns in the image;  $p_{j, k}$  are pixel values;  $x_k$  and  $y_j$  are pixel coordinates;  $m$  and  $n$  are integer power exponents that define the moment order.

The central moment  $U_U(m, n)$  is the spatial moment computed relative to the "center of gravity"  $(x_0, y_0)$ :

$$U_U(m, n) = \sum_{j, k} (x_k - x_0)^m (y_j - y_0)^n p_{j, k}$$

where  $x_0 = M_U(1, 0) / M_U(0, 0)$  and  $y_0 = M_U(0, 1) / M_U(0, 0)$ .

The normalized spatial moment  $M(m, n)$  and central moment  $U(m, n)$  are defined as follows:

$$M(m, n) = \frac{M_U(m, n)}{M_U(0, 0)^{\frac{m+n+2}{2}}}$$

$$U(m, n) = \frac{U_U(m, n)}{U_U(0, 0)^{\frac{m+n+2}{2}}}$$

The Intel IPP functions support moments of order  $(m, n)$  with  $0 \leq m + n \leq 3$ . The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type `IppiMomentState_64s` (for integer versions) or `IppiMomentState_64f` (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the *hint* argument to one of the following values that are listed in [Table "Hint Arguments for Image Moment Functions"](#):

## Hint Arguments for Image Moment Functions

Value	Description
ippAlgHintNone	The computation algorithm will be chosen by the internal function logic.
ippAlgHintFast	Fast algorithm must be used. The output results will be less accurate.
ippAlgHintAccurate	High accuracy algorithm must be used. The function will need more time to execute.

## MomentGetSize

*Computes the size of the external buffer for the moment context structure.*

---

### Syntax

```
IppStatusippiMomentGetSize_64f(IppHintAlgorithm hint, int* pSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSize</i>	Pointer to the computed value of the buffer size.
<i>hint</i>	Option to select the algorithmic implementation of the function.

### Description

Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function [ippiMomentInit](#). Computation algorithm is specified by *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)).

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pSize</i> pointer is NULL.

## MomentInit

*Initializes the moment context structure.*

---

### Syntax

```
IppStatusippiMomentInit_64f(IppiMomentState_64f* pState, IppHintAlgorithm hint);
```

### Include Files

ippi.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pState</i>	Pointer to the structure for storing moment values.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

This function initializes the structure that is needed for the function [ippiMoments](#) to store the computed image moments. Computation algorithm is specified by *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)).

The structure is allocated in the external buffer. The size of this buffer can be computed by the function [ippiMomentGetStateSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer is NULL.

## Moments

*Computes all image moments of order 0 to 3 and Hu moment invariants.*

## Syntax

### Case 1: Computation of floating-point results

```
IppStatusippiMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize  
roiSize, IppiMomentState_64f* pCtx);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_AC4R	16u_AC4R	32f_AC4R

## Include Files

ippi.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pCtx</i>	Pointer to the structure that stores image moments.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes all spatial and central moments of order 0 to 3 for the source image *pSrc*. The seven Hu moment invariants are also computed. Different functions, `ippiMoments64s` and `ippiMoments64f`, are used to compute image moments in integer and floating-point formats, respectively.

The `ippiMoments` function computes spatial moment values relative to the image point referred to by *pSrc*. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use [`ippiGetSpatialMoment`](#) functions to recalculate them.

The moments' values are stored in the *pCtx* structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pCtx</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.

## GetSpatialMoment

*Retrieves image spatial moment of the specified order, computed by `ippiMoments`.*

## Syntax

```
IppStatusippiGetSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pState</i>	Pointer to the structure that stores image moments.
<i>mOrd, nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>roiOffset</i>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<i>pValue</i>	Pointer to the retrieved moment value.

## Description

This function returns the pointer *pValue* to the spatial moment that was previously computed by the `ippiMoments` function. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin. You may also obtain spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

The moment order is specified by the integer exponents *mOrd, nOrd*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>mOrd + nOrd</i> is greater than 3, or <i>nChannel</i> has an illegal value.

## GetCentralMoment

*Retrieves image central moment computed by `ippiMoments`.*

## Syntax

```
IppStatusippiGetCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib`

## Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd, nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .

<i>nChannel</i>	The channel for which the moment is returned.
<i>pValue</i>	Pointer to the returned moment value.

## Description

This function returns the pointer *pValue* to the central moment previously computed by the [ippiMoments](#) function. The moment order is specified by the integer exponents *mOrd*, *nOrd*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>mOrd + nOrd</i> is greater than 3, or <i>nChannel</i> has an illegal value.

## GetNormalizedSpatialMoment

*Retrieves the normalized value of the image spatial moment computed by [ippiMoments](#).*

---

## Syntax

```
IppStatusippiGetNormalizedSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>roiOffset</i>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<i>pValue</i>	Pointer to the returned normalized moment value.

## Description

This function normalizes the spatial moment value that was previously computed by the [ippiMoments](#) function, and returns the pointer *pValue* to the normalized moment. See [Image Moments](#) for details of moments normalization. The moment order (*mOrd*, *nOrd*) is specified by integer power exponents. All

spatial moment values are computed by [ippiMoments](#) relative to the image ROI origin. You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if $M(0, 0)$ value is close to zero.
ippStsSizeErr	Indicates an error condition if <i>mOrd + nOrd</i> is greater than 3, or <i>nChannel</i> has an illegal value.

## GetNormalizedCentralMoment

*Retrieves the normalized value of the image central moment computed by [ippiMoments](#).*

### Syntax

```
IppStatusippiGetNormalizedCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>pValue</i>	Pointer to the returned moment value.

### Description

This function normalizes the central moment value that was previously computed by the [ippiMoments](#) function, and returns the pointer *pValue* to the normalized moment. The moment order (*mOrd*, *nOrd*) is specified by the integer power exponents. See [Image Moments](#) for details of moments normalization.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.

ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if $M(0, 0)$ value is close to zero.

## GetHuMoments

*Retrieves image Hu moment invariants computed by ippiMoments function.*

---

### Syntax

```
IppStatus ippiGetHuMoments_64f(const IppiMomentState_64f* pState, int nChannel,
IppiHuMoment_64f pHm);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pState</i>	Pointer to the structure that stores image moments.
<i>nChannel</i>	The channel for which the moment is returned.
<i>pHm</i>	Pointer to the array containing the Hu moment invariants.

### Description

This function returns the pointer *pHm* to the array of seven Hu moment invariants previously computed by the [ippiMoments](#) function.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pState</i> or <i>pHm</i> pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if $M(0, 0)$ value is close to zero.

### Example

To better understand usage of this function, refer to the `GetHuMoments.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Image Norms

---

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value)

- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values).

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

## **Norm\_Inf**

*Computes the infinity norm of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Masked operation on one-channel data

```
IppStatusippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*  
pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

#### Case 3: Operation on multi-channel data

```
IppStatusippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

```
IppStatusippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,  
Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

#### Case 4: Masked operation on multi-channel data

```
IppStatusippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*  
pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

## Include Files

ippcv.h

ippi.h

## Domain Dependencies

Flavors declared in `ippcv.h`:

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels.
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pValue</code>	Pointer to the computed infinity norm of pixel values.
<code>value</code>	An array containing the computed infinity norms of channel values in case of multi-channel data.
<code>pNorm</code>	Pointer to the computed norm value in the mask mode.

## Description

The flavors of the function `ippiNorm_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see [Regions of Interest in Intel IPP](#)) and computes the infinity norm `pValue` (`pNorm` for the mask mode) for the source image `pSrc`. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by non-zero mask values.

For non-masked operations on a multi-channel image (*Case 3*), the norm is computed separately for each channel and stored in the array `value`.

In the mask multi-channel mode (*Case 4*), the norm is computed for a single channel of interest specified by `coi`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>srcStep</code> or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .

---

ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Norm\_L1

Computes the L1- norm of image pixel values.

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for mod:

8u\_C1R      16u\_C1R      16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatusippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatusippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatusippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R      16u\_C3R      16s\_C3R

```
IppStatusippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R      16u\_C4R      16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatusippiNorm_L1_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[3], IppHintAlgorithm hint);
```

Supported values for mod:

```
IppStatusippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatusippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of pixel values.
<i>value</i>	An array containing the computed L1- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

The flavors of the function `ippiNorm_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (*Case 4, 5*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below demonstrates how an image norm can be computed.

```
IppStatus norm( void ){
    Ipp64f sum, normL1;

    Ipp8u x[5*4];
    IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   ippiSum_8u_C1R( x, 5, roi, &sum);
    returnippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

## Norm\_L2

*Computes the L2- norm of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for *mod*:

8u\_C1R        16u\_C1R        16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatusippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatusippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
--------	---------	---------

```
IppStatusippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatusippiNorm_L2_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[3], IppHintAlgorithm hint);
```

```
IppStatusippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatusippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of pixel values.
<i>value</i>	An array containing the computed L2- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNorm_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (*Case 4, 5*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormDiff\_Inf

*Computes the infinity norm of differences between pixel values of two images.*

## Syntax

### Case 1: Operation on one-channel data

```
IppStatusippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

### Case 2: Masked operation on one-channel data

```
IppStatusippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

### Case 3: Operation on multi-channel data

```
IppStatusippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

```
IppStatusippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

### Case 4: Masked operation on multi-channel data

```
IppStatusippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

## Include Files

ippcv.h

ippi.h

## Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared in `ippi.h`:

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc1</code> , <code>pSrc2</code>	Pointers to the source images ROI.
<code>src1Step</code> , <code>src2Step</code>	Distance in bytes between starts of consecutive lines in the source images.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels.
<code>pValue</code>	Pointer to the computed infinity norm of difference between pixel values.
<code>value</code>	An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pNorm</code>	Pointer to the computed norm value in the mask mode.

## Description

The flavors of the function `ippiNormDiff_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm `pValue` (`pNorm` in the mask mode) of differences between pixel values of the two source images `pSrc1` and `pSrc2`. This norm is defined as the largest absolute value of differences:

$$\text{norm} = \max |pSrc1 - pSrc2|$$

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 3*), the norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (*Case 4*), the norm is computed for a single channel of interest specified by `coi`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.

ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.
--------------	---

## NormDiff\_L1

Computes the L1- norm of differences between pixel values of two images.

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u\_C1R      16u\_C1R      16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatusippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatusippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatusippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R      16u\_C3R      16s\_C3R

```
IppStatusippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R      16u\_C4R      16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatusippiNormDiff_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

```
IppStatusippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

## Case 6: Masked operation on multi-channel data

```
IppStatusippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

## Include Files

ippcv.h

ippi.h

## Domain Dependencies

Flavors declared in ippcv.h:

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of difference between pixel values.
<i>value</i>	An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormDiff_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute values of differences:

$$\text{norm} = \sum |pSrc1 - pSrc2|$$

Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 4, 5*), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width * pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Example

The code example below shows how to use the function `ippiNormDiff_L1_8u_C1R`.

```
void func_normdiff_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

   ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
   ippiSet_8u_C1R(2, pSrc2, src2Step, roi);

   ippiNormDiff_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
```

Result -> 20.0

## NormDiff\_L2

*Computes the L2- norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u\_C1R        16u\_C1R        16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatusippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatusippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u\_C1MR        16u\_C1MR        32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatusippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u\_C3R        16u\_C3R        16s\_C3R

```
IppStatusippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u\_C4R        16u\_C4R        16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatusippiNormDiff_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);
```

```
IppStatusippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatusippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of difference between pixel values.
<i>value</i>	An array containing the computed L2- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

The flavors of the function `ippiNormDiff_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*. This norm is defined as the square root of the sum of squared differences:

$$\text{norm} = \sqrt{\sum |p_{src1} - p_{src2}|^2} .$$

Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Case 4, 5*), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width * pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormRel\_Inf

*Computes the relative error for the infinity norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Masked operation on one-channel data

```
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

**Case 3: Operation on multi-channel data**

```
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

```
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

**Case 4: Masked operation on multi-channel data**

```
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValue</i>	Pointer to the computed relative error value.

---

<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.

## Description

The flavors of the function `ippiNormRel_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the largest absolute pixel value in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer *pSrc2*. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 3), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (Case 4), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width * pixelSize</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the infinity norm of <i>pSrc2</i> has a zero value.

## NormRel\_L1

*Computes the relative error for the L1 norm of differences between pixel values of two images.*

## Syntax

### Case 1: Operation on one-channel integer data

```
IppStatusippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod*:

8u\_C1R

16u\_C1R

16s\_C1R

**Case 2: Operation on one-channel floating-point data**

```
IppStatusippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm
hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatusippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatusippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
--------	---------	---------

```
IppStatusippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatusippiNormRel_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);
```

```
IppStatusippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatusippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

**Headers:**ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared inippi.h:

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormRel_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Cases 4, 5*), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .

ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when <i>coi</i> is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L1 norm of <i>pSrc2</i> has a zero value.

## Example

The code example below shows how to use the function `ippiNormRel_L1_8u_C1R`.

```
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiset_8u_C1R(1, pSrc1, src1Step, roi);
    ippiset_8u_C1R(2, pSrc2, src2Step, roi);
    ippinormrel_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.5
```

## NormRel\_L2

*Computes the relative error for the L2 norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatusippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
--------	---------	---------

**Case 2: Operation on one-channel floating-point data**

```
IppStatusippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm
hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatusippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR
---------	----------	----------

**Case 4: Operation on multi-channel integer data**

```
IppStatusippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
--------	---------	---------

```
IppStatusippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R
--------	---------	---------

**Case 5: Operation on multi-channel floating-point data**

```
IppStatusippiNormRel_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);
```

```
IppStatusippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatusippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR
----------	-----------	-----------

**Include Files**

ippcv.h

ippi.h

**Domain Dependencies**

Flavors declared in ippcv.h:

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared in `ippi.h`:

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<code>pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>src1Step, src2Step</code>	Distance in bytes between starts of consecutive lines in the source images.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>roiSize</code>	Size of the source ROI in pixels.
<code>coi</code>	Channel of interest (for color images only); can be 1, 2, or 3.
<code>pValue</code>	Pointer to the computed relative error value.
<code>value</code>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<code>pNorm</code>	Pointer to the computed relative norm value in the mask mode.
<code>hint</code>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormRel_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm of differences between pixel values of two source buffers `pSrc1` and `pSrc2`. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error `pValue` (`pNorm` in the mask mode) is then formed by dividing the computed norm of differences by the L2- norm of the second source image buffer `pSrc2`. Computation algorithm is specified by the `hint` argument (see [Table "Hint Arguments for Image Moment Functions"](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (*Cases 4, 5*), the relative norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (*Case 6*), the relative norm is computed for a single channel of interest specified by `coi`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .

---

ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIERR	Indicates an error when <i>coi</i> is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L2 norm of <i>pSrc2</i> has a zero value.

## Example

The code example below shows how to use the function `ippiNormRel_L2_8u_C1R`.

```
void func_normrel_11()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiset_8u_C1R(1, pSrc1, src1Step, roi);
    ippiset_8u_C1R(10, pSrc2, src2Step, roi);
    ippinormrel_L2_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.9
```

## Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images  $f$  and  $g$  are considered as matrices with  $M$  column and  $N$  rows containing pixel values  $f[i, j]$ ,  $g[i, j]$ , respectively ( $0 \leq i < M$ ,  $0 \leq j < N$ ), the universal image quality index  $Q$  may be calculated as a product of three components:

$$Q = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \cdot \frac{2\bar{f}\bar{g}}{(\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\sigma_f \sigma_g}{\sigma_f^2 + \sigma_g^2}$$

where

$$\bar{f} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f[i,j] \quad \bar{g} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g[i,j]$$

$$\sigma_{fg} = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})(g[i,j] - \bar{g})$$

$$\sigma_f^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})^2 \quad \sigma_g^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g[i,j] - \bar{g})^2$$

The first component is the correlation coefficient, which measures the degree of linear correlation between images  $f$  and  $g$ . It varies in the range  $[-1, 1]$ . The best value 1 is obtained when  $f$  and  $g$  are linearly related, which means that  $g[i,j] = af[i,j] + b$  for all possible values of  $i$  and  $j$ . The second component, with a value range of  $[0, 1]$ , measures how close the mean luminance is between images. Since  $\sigma_f$  and  $\sigma_g$  can be considered as estimates of the contrast of  $f$  and  $g$ , the third component measures how similar the contrasts of the images are. The value range for this component is also  $[0, 1]$ .

The range of values for the index  $Q$  is  $[-1, 1]$ . The best value 1 is achieved if and only if the images are identical.

## QualityIndexGetBufferSize

*Computes the size of the work buffer for the ippiQualityIndex function.*

---

### Syntax

```
IppStatus ippiQualityIndexGetBufferSize(IppDataType srcType, IppChannels ippChan,
IppiSize roiSize, int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>srcType</i>	Data type of the source images. Possible values: ipp8u, ipp16u, or ipp32f.
<i>ippChan</i>	Number of channels in the source images. Possible values: ippC1, ippC3, or ippAC4.
<i>roiSize</i>	Size, in pixels, of the source images.
<i>pBufferSize</i>	Pointer to the computed value of the buffer size, in bytes.

### Description

The function computes the size of the work buffer, in bytes, for the [ippiQualityIndex](#) function and stores the result in the *pBufferSize* parameter.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when <i>srcType</i> has an illegal value.
ippStsChannelErr	Indicates an error when <i>ippChan</i> has an illegal value.

## See Also

[QualityIndex](#) Computes the universal image quality index.

## QualityIndex

*Computes the universal image quality index.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatusippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype>
pQualityIndex[1], Ipp8u* pBuffer);
```

Supported values for *mod*:

8u32f_C1R	16u32f_C1R	32f_C1R
-----------	------------	---------

#### Case 2: Operation on multi-channel data

```
IppStatusippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype>
pQualityIndex[3], Ipp8u* pBuffer);
```

Supported values for *mod*:

8u32f_C3R	16u32f_C3R	32f_C3R
8u32f_AC4R	16u32f_AC4R	32f_AC4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.

<i>roiSize</i>	Size of the source ROI in pixels.
<i>pQualityIndex</i>	Pointer to the computed quality index value.
<i>pBuffer</i>	Pointer to the buffer for internal calculations. To compute the size of the buffer, use the <a href="#">QualityIndexGetBufferSize</a> function.

## Description

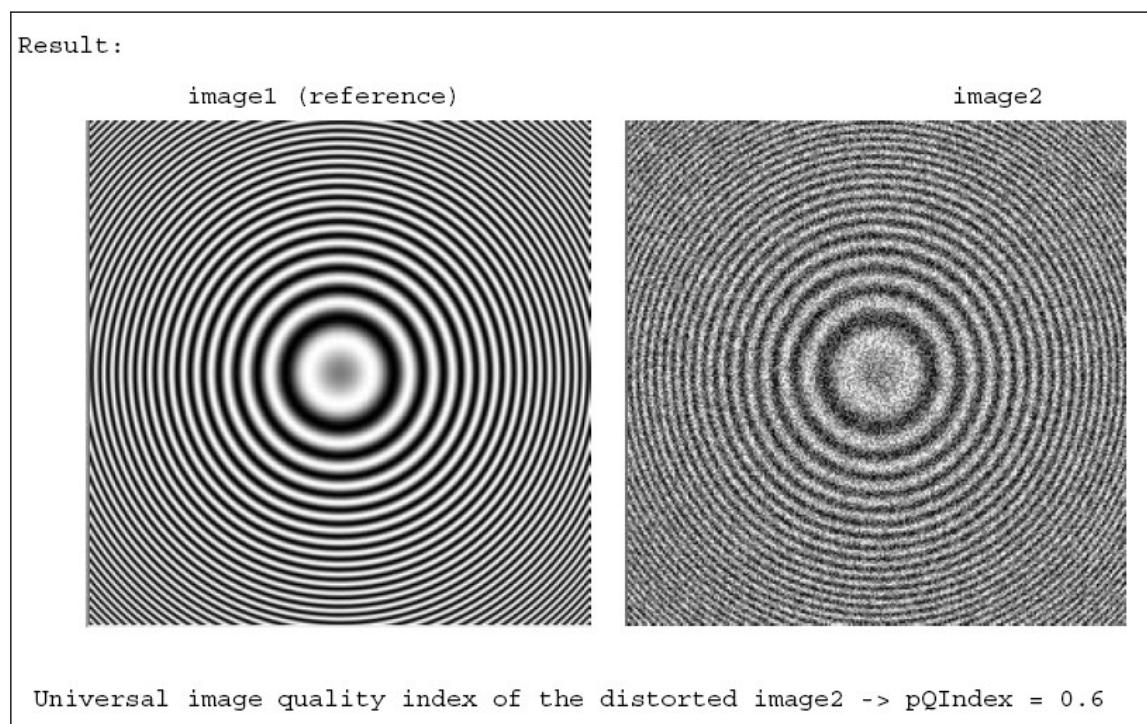
This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the universal image quality index for two images *pSrc1* and *pSrc2* according to the formula in the introduction section above. The computed value of the index is stored in *pQualityIndex*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value.
<code>ippStsQualityIndexErr</code>	Indicates an error condition if pixel values of one of the images are constant.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Example

To better understand usage of this function, refer to the `QualityIndex.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.



**See Also**

[QualityIndexGetBufferSize](#) Computes the size of the work buffer for the `ippiQualityIndex` function.

## Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance  $s_{tx}(r, c)$  between a template and an image for the pixel in row  $r$  and column  $c$  is given by the equation:

$$s_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[ t(j, i) - x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right) \right]^2$$

where  $x(r, c)$  is the image pixel value in row  $r$  and column  $c$ , and  $t(j, i)$  is the template pixel value in row  $j$  and column  $i$ ; template size is  $tplCols$  by  $tplRows$  and its center is positioned at  $(r, c)$ .

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation  $R_{tx}(r, c)$  between a template and an image at the pixel in row  $r$  and column  $c$  is computed by the equation :

$$R_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j, i) \cdot x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right)$$

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

$$G_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j, i) - \bar{t}] \cdot \left[ x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right) - \bar{x}(r, c) \right]$$

where  $\bar{t}$  with the overline is the mean of the template, and  $\bar{x}$  with the overline is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute *normalized* values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD:  $\sigma_{tx}(r, c)$

$$\sigma_{tx}(r, c) = \frac{s_{tx}(r, c)}{\sqrt{R_{xx}(r, c)R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

normalized cross-correlation  $\rho_{tx}(r, c)$ :

$$\rho_{tx}(r, c) = \frac{R_{tx}(r, c)}{\sqrt{R_{xx}(r, c)R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $R_{xx}$  and  $R_{tt}$  denote the autocorrelation of the image and the template, respectively:

$$R_{xx}(r, c) = \sum_{\substack{j=r-\frac{tplRows-1}{2} \\ i=c-\frac{tplCols-1}{2}}}^{\substack{r+\frac{tplRows-1}{2} \\ c+\frac{tplCols-1}{2}}} x_{j, i} x_{j, i}$$

$$R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j, i} t_{j, i}$$

Normalized correlation coefficient  $\gamma_{tx}(r, c)$ :

$$\gamma_{tx}(r, c) = \frac{G_{tx}(r, c)}{\sqrt{G_{xx}(r, c) G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $G_{xx}$  and  $G_{tt}$  denote the autocorrelations of the image and the template without constant brightness component, respectively:

$$G_{xx}(r, c) = \sum_{\substack{j=r-\frac{tplRows-1}{2} \\ i=c-\frac{tplCols-1}{2}}}^{\substack{r+\frac{tplRows-1}{2} \\ c+\frac{tplCols-1}{2}}} [x_{j, i} - \bar{x}(r, c)]^2$$

$$G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j, i} - \bar{t})^2$$

## SqrDistanceNormGetBufferSize

*Computes the size of the work buffer for the ippiSqrDistanceNorm function.*

---

### Syntax

```
IppStatus ippiSqrDistanceNormGetBufferSize (IppiSize srcRoiSize, IppiSize tplRoiSize,
IppEnum algType, int* pBufferSize);
```

### Include Files

ippi.h

### Parameters

<i>srcRoiSize, tplRoiSize</i>	Size of the source/template ROI in pixels.
<i>algType</i>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the <i>IppAlgType</i> , <i>IppiROIShape</i> , and <i>IppiNormOp</i> values.
<i>pBufferSize</i>	Pointer to the size of the work buffer.

## Description

The `ippiSqrDistanceNormGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that computes the Euclidean distance between an image and a template. The result is stored in the `pBufferSize` parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsSizeErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>srcRoiSize</code> or <code>tplRoiSize</code> is negative, or equal to zero</li> <li>• the value of <code>srcRoiSize</code> is less than the corresponding value of <code>tplRoiSize</code></li> </ul>
<code>ippStsAlgTypeErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippAlgMask</code> values differs from the <code>ippAlgAuto</code>, <code>ippAlgDirect</code>, or <code>ippAlgFFT</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiROIIMask</code> values differs from the <code>ippiROIFull</code>, <code>ippiROISame</code>, or <code>ippiROIValid</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiNormMask</code> values differs from the <code>ippiNormNone</code> or <code>ippiNorm</code> values.</li> </ul>
<code>ippStsNullErr</code>	Indicates an error when the <code>pBufferSize</code> is NULL.

## See Also

[Structures and Enumerators](#)

[SqrDistanceNorm](#) Computes Euclidean distance between an image and a template.

## SqrDistanceNorm

Computes Euclidean distance between an image and a template.

### Syntax

#### Case 1: Operating with integer output

```
IppStatusippiSqrDistanceNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

#### Case 2: Operating on data with floating-point output

```
IppStatusippiSqrDistanceNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f_C1R	8u32f_C1R	16u32f_C1R
---------	-----------	------------

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>srcRoiSize</code>	Size of the source ROI in pixels.
<code>pTpl</code>	Pointer to the template image.
<code>tplStep</code>	Distance, in bytes, between the starting points of consecutive lines in the template image.
<code>tplRoiSize</code>	Size of the template ROI in pixels.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>scaleFactor</code>	Scale factor.
<code>algType</code>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the <code>IppAlgType</code> , <code>IppiROIShape</code> , and <code>IppiNormOp</code> values.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

Before using this function, you need to compute the size of the work buffer using the `ippiSqrDistanceNormGetBufferSize` function.

This function operates with ROI.

Depending on the `IppiNormOp` value set to the `algType` parameter, the function calculates the following results:

IppiNormOp Value	Result
<code>ippiNormNone</code>	Squared Euclidean distances $s_{tx}(r, c)$
<code>ippiNorm</code>	Normalized squared Euclidean distances $\sigma_{tx}(r, c)$

For more information on how each value is calculated, see [Image Proximity Measures](#).

The size of the resulting matrix depends on the `IppiROIShape` value:

IppiROIShape Value	Matrix Size
<code>ippiROIFull</code>	$(W_S + W_t - 1) * (H_S + H_t - 1)$
<code>ippiROISame</code>	$W_S * H_S$
<code>ippiROIValid</code>	$(W_S - W_t + 1) * (H_S - H_t + 1)$

where

- $W_S$ ,  $H_S$  is the width and height of the source image

- $W_t, H_t$  is the width and height of the template image

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsStepErr	Indicates an error when the value of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> is negative, or equal to zero.
ippStsSizeErr	Indicates an error when: <ul style="list-style-type: none"> <li>• <i>srcRoiSize</i> or <i>tplRoiSize</i> is negative, or equal to zero</li> <li>• the value of <i>srcRoiSize</i> is less than the corresponding value of <i>tplRoiSize</i></li> </ul>
ippStsAlgTypeErr	Indicates an error when: <ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between the <i>algType</i> and <i>ippAlgMask</i> values differs from the <i>ippAlgAuto</i>, <i>ippAlgDirect</i>, or <i>ippAlgFFT</i> values;</li> <li>• the result of the bitwise AND operation between the <i>algType</i> and <i>ippiROIIMask</i> values differs from the <i>ippiROIIFull</i>, <i>ippiROISame</i>, or <i>ippiROIValid</i> values;</li> <li>• the result of the bitwise AND operation between the <i>algType</i> and <i>ippiNormMask</i> values differs from the <i>ippiNormNone</i> or <i>ippiNorm</i> values;</li> </ul>

## Example

The code example below demonstrates how to use the *ippiSqrDistanceNormGetBufferSize* and *ippiSqrDistanceNorm* functions.

```
IppStatus SqrDistanceNormExample()
{
    IppStatus status;
    Ipp32f pSrc[5*4];
    Ipp32f pTpl[5*4];
    Ipp32f pDst[9*7];//(5+5-1) x (4+4-1)

    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {5,4};
    IppiSize dstRoiSize = {9,7};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 5*sizeof(Ipp32f);
    int dstStep = 9*sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto | ippNorm | ippROIIFull);
    Ipp8u *pBuffer;
    int bufSize=0;

   ippiSet_32f_C1R(2.0, pSrc, srcStep, srcRoiSize);
   ippiSet_32f_C1R(1.0, pTpl, tplStep, tplRoiSize);

    status =ippiSqrDistanceNormGetBufferSize(srcRoiSize, tplRoiSize, ipp32f, funCfg, &bufSize);
    if (status != ippStsNoErr) return status;

    pBuffer = ippsMalloc_8u( bufSize );
}
```

```

status =ippiSqrDistanceNorm_32f_C1R( pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize,
pDst, dstStep, funCfg, pBuffer);
printf_2D_32f("pDst", pDst, dstRoiSize);

ippsFree( pBuffer );
return status;
}

```

The result is as follows:

```

pDst ->
2.24 1.58 1.29 1.12 1.00 1.12 1.29 1.58 2.24
1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.29
1.12 0.79 0.65 0.56 0.50 0.56 0.65 0.79 1.12
1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.29
1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
2.24 1.58 1.29 1.12 1.00 1.12 1.29 1.58 2.24

```

## See Also

[Integer Result Scaling](#)

[Image Proximity Measures](#)

[Regions of Interest in Intel IPP](#)

[Structures and Enumerators](#)

[SqrDistanceNormGetBufferSize](#) Computes the size of the work buffer for the `ippiSqrDistanceNorm` function.

## CrossCorrNormGetBufferSize

*Computes the size of the work buffer for the `ippiCrossCorrNorm` function.*

### Syntax

```
IppStatusippiCrossCorrNormGetBufferSize (IppiSize srcRoiSize, IppiSize tplRoiSize,
IppEnum algType, int* pBufferSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>srcRoiSize, tplRoiSize</code>	Size of the source/template ROI in pixels.
<code>algType</code>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the <code>IppAlgType</code> , <code>IppiROIShape</code> , and <code>IppiNormOp</code> values.
<code>pBufferSize</code>	Pointer to the size of the work buffer.

## Description

The `ippiCrossCorrNormGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional cross-correlation. The result is stored in the `pBufferSize` parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsSizeErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li>• <code>srcRoiSize</code> or <code>tplRoiSize</code> is negative, or equal to zero</li> <li>• the value of <code>srcRoiSize</code> is less than the corresponding value of <code>tplRoiSize</code>.</li> </ul>
<code>ippStsAlgTypeErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippAlgMask</code> differs from the <code>ippAlgAuto</code>, <code>ippAlgDirect</code>, or <code>ippAlgFFT</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiROIIMask</code> differs from the <code>ippiROIIFull</code>, <code>ippiROISame</code>, or <code>ippiROIValid</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiNormMask</code> differs from the <code>ippiNormNone</code>, <code>ippiNorm</code>, or <code>ippiNormCoefficient</code> values.</li> </ul>
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pBufferSize</code> is NULL.

## See Also

[Structures and Enumerators](#)

[CrossCorrNorm](#) Computes a normalized cross-correlation between an image and a template.

## CrossCorrNorm

*Computes a normalized cross-correlation between an image and a template.*

## Syntax

### Case 1: Operating on data with integer output

```
IppStatusippiCrossCorrNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int
dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

### Case 2: Operating on data with floating-point output

```
IppStatusippiCrossCorrNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f*
pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`

`8u32f_C1R`

`16u32f_C1R`

**Case 3: Operating on data with integer output with TL functions**

```
IppStatus ippsCrossCorrNorm_ 8u_C1RSfs_T (const Ipp8u* pSrc, int srcStep,
IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
int dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

**Case 4: Operating on data with floating-point output with TL functions**

```
IppStatus ippsCrossCorrNorm_ <mod>_T (const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize,
Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`

`8u32f_C1R`

`16u32f_C1R`

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>srcRoiSize</code>	Size of the source ROI in pixels.
<code>pTpl</code>	Pointer to the template image.
<code>tplStep</code>	Distance, in bytes, between the starting points of consecutive lines in the template image.
<code>tplRoiSize</code>	Size of the template ROI in pixels.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>scaleFactor</code>	Scale factor.
<code>algType</code>	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the <code>IppAlgType</code> , <code>IppiROIShape</code> , and <code>IppiNormOp</code> values.
<code>pBuffer</code>	Pointer to the work buffer.

**Description**

This function operates with ROI.

Depending on the `IppiNormOp` value set to the `algType` parameter, the function calculates the following results:

IppiNormOp Value	Result
ippiNormNone	Cross-correlation values $R_{tx}(r, c)$
ippiNorm	Normalized cross-correlation values $\rho_{tx}(r, c)$
ippiNormCoefficient	Normalized correlation coefficients $\gamma_{tx}(r, c)$

For more information about how each value is calculated, see [Image Proximity Measures](#).

The size of the resulting matrix depends on the IppiROIShape value:

IppiROIShape Value	Matrix Size
ippiROIFull	$(W_S + W_t - 1) * (H_S + H_t - 1)$
ippiROISame	$W_S * H_S$
ippiROIValid	$(W_S - W_t + 1) * (H_S - H_t + 1)$

where

- $W_S, H_S$  is the width and height of the source image
- $W_t, H_t$  is the width and height of the template image

Before using this function, you need to compute the size of the work buffer using the `ippiCrossCorrNormGetBufferSize` function.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsStepErr	Indicates an error when the value of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> is negative, or equal to zero.
ippStsSizeErr	Indicates an error when: <ul style="list-style-type: none"> <li>• the <code>srcRoiSize</code> or <code>tplRoiSize</code> is negative, or equal to zero.</li> <li>• the value of <code>srcRoiSize</code> is less than the corresponding value of the <code>tplRoiSize</code>.</li> </ul>
ippStsAlgTypeErr	Indicates an error when: <ul style="list-style-type: none"> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippAlgMask</code> differs from the <code>ippAlgAuto</code>, <code>ippAlgDirect</code>, or <code>ippAlgFFT</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiROIIMask</code> differs from the <code>ippiROIIFull</code>, <code>ippiROISame</code>, or <code>ippiROIValid</code> values.</li> <li>• the result of the bitwise AND operation between the <code>algType</code> and <code>ippiNormMask</code> differs from the <code>ippiNormNone</code>, <code>ippiNorm</code>, or <code>ippiNormCoefficient</code> values.</li> </ul>

## Example

The code example below demonstrates how to use the `ippiCrossCorrNormGetBufferSize` and `ippiCrossCorrNorm` functions.

```
IppStatus CrossCorrNormExample()
{
    IppStatus status;
    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {3,3};
    IppiSize dstRoiSize = {5,4}; // same as src
    Ipp32f pSrc[5*4] = { 1.0f, 2.0f, 1.5f, 4.1f, 3.6f,
                          0.2f, 3.2f, 2.5f, 1.5f, 10.0f,
                          5.0f, 6.8f, 0.5f, 4.1f, 1.1f,
                          7.1f, 4.2f, 2.2f, 8.7f, 10.0f };
    Ipp32f pTpl[3*3] = {2.1f, 3.5f, 7.7f,
                        0.4f, 2.3f, 5.5f,
                        1.4f, 2.8f, 3.1f };
    Ipp32f pDst[5*4];
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 5*sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto|ippiROISame|ippiNorm);
    Ipp8u *pBuffer;
    int bufSize;

    status = ippiCrossCorrNormGetBufferSize(srcRoiSize, tplRoiSize, ipp32f, funCfg, &bufSize);
    if (status != ippStsNoErr) return status;

    pBuffer = ippsMalloc_8u( bufSize );

    status = ippiCrossCorrNorm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize,
    pDst, dstStep, funCfg, pBuffer);
    printf_2D_32f("pDst", pDst, dstRoiSize);

    ippsFree( pBuffer );
    return status;
}
```

The result is as follows:

```
pDst ->
0.53 0.54 0.58 0.50 0.30
0.68 0.62 0.68 0.83 0.38
0.77 0.55 0.60 0.81 0.42
0.81 0.46 0.70 0.62 0.24
```

## See Also

[Integer Result Scaling](#)  
[Regions of Interest in Intel IPP](#)  
[Structures and Enumerators](#)  
[Image Proximity Measures](#)

## SADGetBufferSize

*Computes the size of the work buffer for the `ippiSAD` function.*

## Syntax

```
IppStatusippiSADGetBufferSize(IppiSize srcRoiSize, IppiSize tplRoiSize, IppDataType dataType, int numChannels, IppiROIShape shape, int* pBufferSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>shape</i>	Enumeration that defines the shape of the result of the SAD operation (see <a href="#">Structures and Enumerators</a> ).
<i>dataType</i>	Type of the input data.
<i>numChannels</i>	Number of channels in the images.
<i>pBufferSize</i>	Pointer to the computed value of the external buffer size.

## Description

The `ippiSADGetBufferSize` function computes the size of the external work buffer (in bytes) needed for the `ippiSAD` function and stores the result in the `*pBufferSize` parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with a zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if either <code>shape</code> does not equal <code>ippiROIValid</code> or <code>numChannels</code> does not equal 1.

## SAD

*Computes sums of absolute differences (SAD) for a template image and different locations within a source image where the template image can fit.*

## Syntax

```
IppStatusippiSAD_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiROIShape shape, int scaleFactor, Ipp8u* pBuffer);
```

```
IppStatusippiSAD_32f_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppROIshape shape, Ipp8u* pBuffer);
```

Supported values for mod:

8u32s_C1RSfs	16u32s_C1RSfs	16s32s_C1RSfs
--------------	---------------	---------------

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image ROI.
<i>tplStep</i>	Distance, in bytes, between the starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starts of consecutive lines in the destination image.
<i>shape</i>	Enumeration that defines the shape of the result of the SAD operation (see <a href="#">Structures and Enumerators</a> ).
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the buffer for internal calculations.

## Description

The ippiSAD function operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function only supports the `ippiROIValid` value of *shape*, and so the sizes of the destination ROI are different from the sizes of both the source ROI and template ROI and depend on those sizes as follows:

$$\text{dstRoiSize.width} = W = \text{srcRoiSize.width} - \text{tplRoiSize.width} + 1$$

$$\text{dstRoiSize.height} = H = \text{srcRoiSize.height} - \text{tplRoiSize.height} + 1$$

So there are exactly  $W*H$  unique locations within the source image where the template image can fit. The top-left pixel determines each of these locations. For each location, the function computes absolute differences between corresponding pixel values of the source and template images, sums these differences to make the SAD value of the top-left pixel, and assigns the SAD value to the appropriate pixel in the destination buffer.

For integer flavors, the resulting values are also scaled by *scaleFactor*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the <i>pSrc</i> , <i>pTpl</i> , <i>pDst</i> , or <i>pBuffer</i> pointer is <b>NULL</b> .
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with a zero or negative value or if <i>srcRoiSize</i> in any direction is less than <i>tplRoiSize</i> .
ippStsStepErr	Indicates an error condition if <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images)
ippStsBadArgErr	Indicates an error condition if <i>scaleFactor</i> < 0.
ippStsNotSupportedModeErr	Indicates an error condition if intersection of the source and destination ROI is detected or if <i>shape</i> differs from <i>ippiROIValid</i> .

## Example

The code example below shows how to use the `ippiSAD_8u32s_C1RSfs` function.

```
Ipp8u src[8*8] = {1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8};

IppiSize srcRoi = { 7, 7 };

Ipp8u template[3*3] = {
10, 10, 10,
10, 10, 10,
10, 10, 10};

IppiSize tplRoi = { 3, 3 };

Ipp32s dst[8*8] = {0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0};

IppiSize srcRoi = { 3, 3 };

ippiSADGetBufferSize( srcRoi, tplRoi, ippiROIValid , 1, &bufferSize);
Ipp8u* pBuffer = new Ipp8u [bufferSize];
ippiSAD_8u32s_C1RSfs(src, 8, srcRoi, template, 3, tplRoi, dst, 8, ippiROIValid , 1, pBuffer);

Result:
```

1 2 3 4 8 8 8 8	10 10 10	36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8 src	10 10 10 tpl	36 32 22 15 9 0 0 0 dst
1 2 3 4 8 8 8 8	10 10 10	36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8		36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8		36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8		0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8		0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8		0 0 0 0 0 0 0 0

# Image Geometry Transforms

12

This chapter describes the Intel® IPP image processing functions that perform geometric operations of resizing, rotating, warping and remapping an image.

Most functions performing geometric transform of an image use an interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the *interpolation* parameter for rotate, warp, and remap. For resize transform, the interpolation type is part of the function name.

The following interpolation algorithms are used:

- nearest neighbor
- linear interpolation
- cubic convolution
- supersampling
- interpolation using Lanczos window function
- interpolation with two-parameter cubic filters
- optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

<code>IPPI_INTER_NN/ippNearest</code>	Nearest neighbor interpolation.
<code>IPPI_INTER_LINEAR/ippLinear</code>	Linear interpolation.
<code>IPPI_INTER_CUBIC</code>	Cubic interpolation.
<code>IPPI_INTER_LANCZOS/ippLanczos</code>	Interpolation using 3-lobed Lanczos window function.
<code>IPPI_INTER_CUBIC2P_BSPLINE</code>	Interpolation using B-spline.
<code>IPPI_INTER_CUBIC2P_CATMULLROM</code>	Interpolation using Catmull-Rom spline.
<code>IPPI_INTER_CUBIC2P_B05C03</code>	Interpolation using special cubic filter.
<code>ippSuper</code>	Supersampling interpolation.
<code>ippCubic</code>	Interpolation with two-parameter cubic filters.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of `IPPI_SMOOTH_EDGE` or `IPPI_SUBPIXEL_EDGE` and the desired interpolation mode, or use the special function flags.

## Caution

You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

See [appendix B "Interpolation in Image Geometric Transform Functions"](#) for more information on the interpolation algorithms that are used in the library.

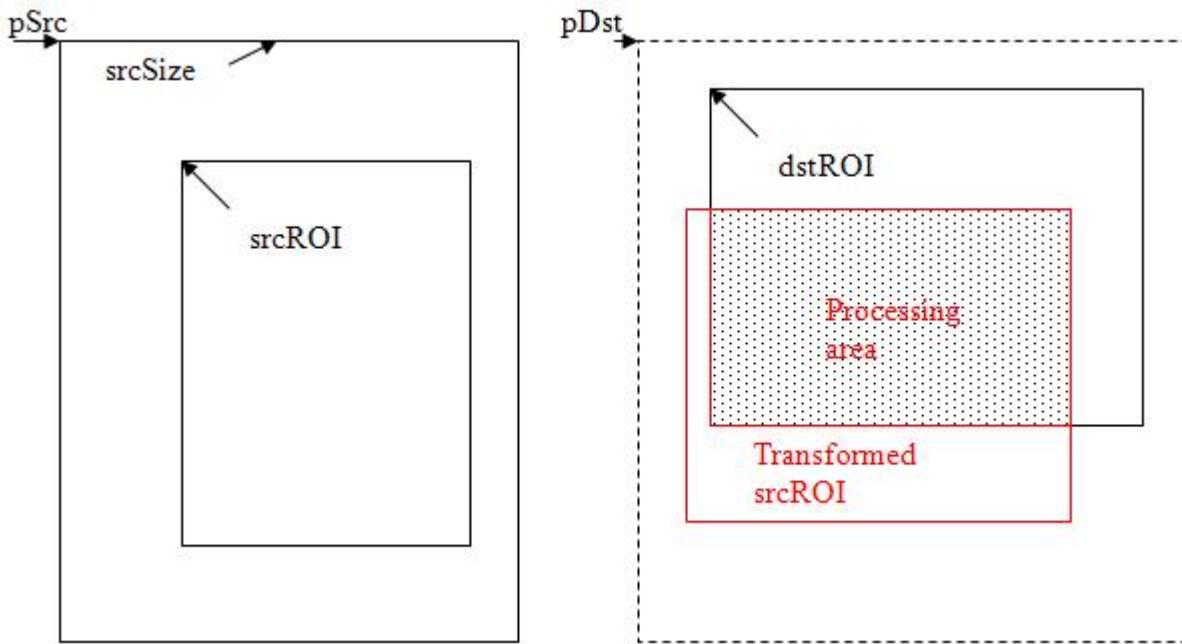
Super Sampling mode of resize transform has several limitations. It can be used only:

- for reducing image size
- for full images resize, while other interpolation modes can be used for full and tiled images for better speed/memory performance.

## ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images. The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see [Regions of Interest in Intel IPP](#) in chapter 2). The main difference is that operations take place in the intersection of the *transformed* source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations (see figure below):

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.



The coordinates in the source and destination images must have the same origin.

When using functions with ROI, every scan line of a source image has a stride. It is padded with zeroes for alignment, so the actual size of a scan line in bytes is often greater than the image width. The size of each row of image data in bytes is determined by the value of `srcStep` parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by `srcRoi` parameter of `IppiRect` type, meaning that all four values describing the rectangular ROI are given explicitly.

The destination image ROI for different functions can be specified either by `dstRoi` parameter of `IppiRect` type, or `dstRoiSize` parameter of `IppiSize` type. In the latter case, only the destination ROI size is passed, while its origin is referenced by `pDst` pointer.

The destination image origin ROI for different functions can be specified by `dstOffset` parameter of `IppiPoint` type and `dstSize` parameter of `IppiSize` type. In this case, the processed destination image corresponds to the processed ROI of the destination image origin.

## Geometric Transform Functions

---

### ResizeYCbCr422GetBufSize

*Computes the size of the external buffer for the NV12 resize transform.*

#### Syntax

```
IppStatusippiResizeYCbCr422GetBufSize(IppiRect srcROI, IppiSize dstRoiSize, int interpolation, int* pSize);
```

#### Include Files

ippi.h

#### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

#### Parameters

<i>srcROI</i>	Region of interest of the source image.
<i>dstRoiSize</i>	Region of interest of the destination image.
<i>interpolation</i>	Type of interpolation to apply to the source image:
	IPP_INTER_NN                  Nearest neighbor interpolation
	IPP_INTER_LINEAR              Linear interpolation
	IPP_INTER_CUBIC               Cubic interpolation
	IPPI_INTER_CUBIC2P_          Catmull-Rom cubic filter CATMULLROM
	IPP_INTER_LANCZOS            Lanczos filter with size 6x6
<i>pSize</i>	Pointer to the size, in bytes, of the external buffer.

#### Description

This function computes the size of the external buffer for the YCbCr resize transform.

#### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if <i>pSize</i> is NULL.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• width of either source or destination ROI is less than 2</li> <li>• height of either source or destination ROI is less than 1</li> </ul>
ippStsDoubleSize	Indicates a warning if width or wither source or destination ROI is not a multiple of 2.

`ippStsInterpolationErr`Indicates an error if *interpolation* has an illegal value.

## ResizeYCbCr422

*Performs resizing of a 4:2:2 two-channel image.*

---

### Syntax

```
IppStatusippiResizeYCbCr422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int interpolation,
Ipp8u* pBuffer);
```

### Include Files

`ippi.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcSize</i>	Size, in pixels, of the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>srcROI</i>	Region of interest in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>interpolation</i>	Type of interpolation to apply to the source image:
	IPP_INTER_NN                  Nearest neighbor interpolation
	IPP_INTER_LINEAR               Linear interpolation
	IPP_INTER_CUBIC                Cubic interpolation
	IPPI_INTER_CUBIC2P_           Catmull-Rom cubic filter CATMULLROM
	IPP_INTER_LANCZOS             Lanczos filter with size 6x6
<i>pBuffer</i>	Pointer to the external buffer.

### Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI to the destination image ROI origin. The function performs resampling of the result using the interpolation mode specified by the *interpolation* parameter, and stores it in the destination image ROI.

The source image is a two-channel image in the 4:2:2 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV422 or YCrCb422.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if <i>pSrc</i> , <i>pDst</i> , or <i>pBuffer</i> is NULL.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• width of either source or destination ROI is less than 2</li> <li>• height of either source or destination ROI is less than 1</li> </ul>
ippStsDoubleSize	Indicates a warning if width of either source or destination ROI is not a multiple of 2.
ippStsInterpolationErr	Indicates an error if <i>interpolation</i> has an illegal value.
ippStsWrongIntersectROI	Indicates a warning if <i>srcROI</i> does not intersect with the source image; no operation is required.

## Resize Functions with Prior Initialization

This section describes the Intel® IPP resize functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

### Using Intel® IPP Resize Functions with Prior Initialization

You can use one of the following approaches to image resizing:

- [Resizing the whole image](#)
- [Resizing a tiled image with one prior initialization](#)
- [Resizing a tiled image with prior initialization for each tile](#)

Interpolation algorithms of the Lanczos, Linear, and Cubic types use edge pixels of the source image that are out of the image origin. When calling the `ippiResize<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels;
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory;
- Mixed borders: a combined approach is applied.

#### NOTE

If you want to resize an image with antialiasing, follow the same instructions as provided below, but use `ippiResizeAntialiasing<Filter>Init` instead of `ippiResize<Filter>Init` for initialization, and `ippiResizeAntialiasing<Filter>` instead of `ippiResize<Filter>`, as a processing function.

## Resizing the Whole Image

You can apply the approach described below to resize when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

To resize the whole image:

1. Call the `ippiResizeGetSize` function with the appropriate interpolation type. This function uses source and destination image sizes to calculate how much memory must be allocated for the `IppResizeSpec` structure and initialization work buffer.

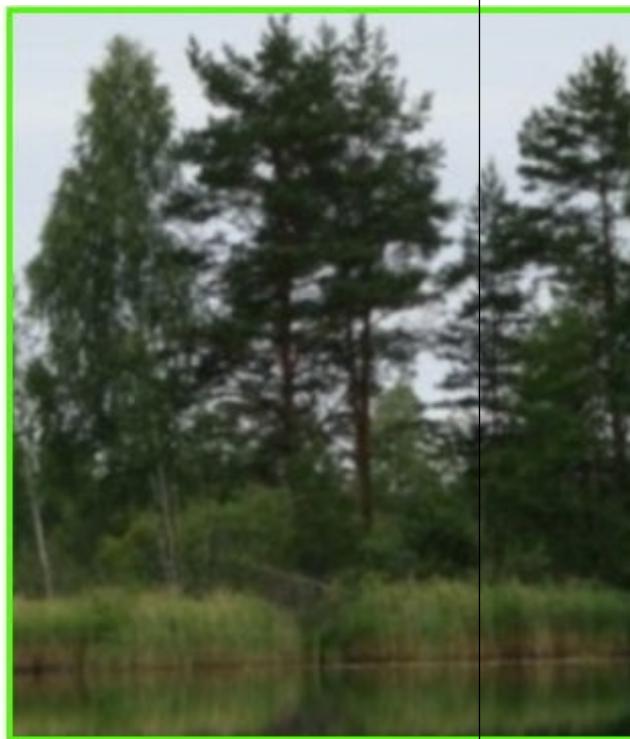
2. Initialize the `IppResizeSpec` structure by calling the `ippiResize<Filter>Init`, where `<Filter>` can take one of the following values: Nearest, Linear, Cubic, Lanczos, and Super. These prerequisite steps allow `resize` to be called multiple times without recalculations.
3. Call the `ippiResizeGetBufferSize` function for the initialized `IppResizeSpec` structure. This function uses the destination image size to calculate how much memory must be allocated for the resize work buffer.
4. Call `ippiResize<Filter>` with the appropriate image type.
5. If you call the `ippiResize<Filter>` function with a `ippBorderInMem` border or any mixed border type, the applied interpolation algorithm uses weighted values from edge pixels of the source image when outside the image boundaries. To obtain the size of the border required for correct edge calculation, call the `ippiResizeGetBorderSize` function for the appropriate flavor. In case of mixed border type, out of image pixels are used only behind the non-replicated edge.
6. You can use mixed borders by using the bitwise OR operation between the `ippBorderRepl` type and the following border types: `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight`.

Figure *Simple Image Resize* shows a simple image resizing example, in which image resolution is increased by 1.5x.

#### Simple Image Resize



a) source image



b) image after resize transform

## Example

The code example below demonstrates whole image resizing with the Lanczos interpolation method:

```
IppStatus resizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize
dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType border = ippBorderRepl;

    /* Spec and init buffer sizes */
    status =ippiResizeGetSize_8u(srcSize, dstSize, ippLanczos, 0, &specSize, &initSize);

    if (status != ippStsNoErr) return status;

    /* Memory allocation */
    pInitBuf = ippsMalloc_8u(initSize);
    pSpec     = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);

    if (pInitBuf == NULL || pSpec == NULL)
    {
        ippsFree(pInitBuf);
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }

    /* Filter initialization */
    status =ippiResizeLanczosInit_8u(srcSize, dstSize, 3, pSpec, pInitBuf);
    ippsFree(pInitBuf);

    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    /* work buffer size */
    status =ippiResizeGetBufferSize_8u(pSpec, dstSize, numChannels, &bufSize);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    pBuffer = ippsMalloc_8u(bufSize);
    if (pBuffer == NULL)
    {
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }

    /* Resize processing */
    status =ippiResizeLanczos_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset, dstSize, border,
```

```

0, pSpec, pBuffer);

    ippsFree(pSpec);
    ippsFree(pBuffer);

    return status;
}

```

## Resizing a Tiled Image with One Prior Initialization

You can apply the approach described below to resize when source and destination images are not fully accessible in memory, or to improve the performance of resizing by external threading.

The main difference between this approach and whole image resizing is that the processing is split into sections of the image called tiles. Each call of the `Resize<Filter>` function works with the destination image origin region of interest (ROI) that is defined by `dstOffset` and `dstSize` parameters. The destination and source ROI must be fully accessible in memory.

To resize an image with the tiled approach:

1. Call the `ippiResizeGetSize` function with the appropriate interpolation type. This function uses the source and destination image sizes to calculate how much memory must be allocated for the `IppResizeSpec` structure and initialization work buffer.
2. Initialize the `IppResizeSpec` structure by calling `ippiResize<Filter>Init`, where `<Filter>` can take one of the following values: Nearest, Cubic, Linear, and Lanczos.
3. Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is often sufficient.
4. Obtain the source ROI for the defined destination tile by calling the `ippiResizeGetSrcRoi` function for the corresponding flavor. The algorithm uses edge pixels that are out of the source ROI to calculate edge pixels of the destination ROI. These out of the source ROI edge pixels must be accessible in memory.
5. If the source ROI is an interior field of the source image origin, obtain the border ROI size by calling the `ippiResizeGetBorderSize` function for the corresponding flavor.
6. If the source ROI is an edge tile, the algorithm can interpolate pixels beyond the image boundary as in the previous method.
7. If the source and destination images are fully accessible in memory, you can use the source ROI offset for the `pSrc` calculation. To obtain the offset, call the `ippiResizeGetSrcOffset` function for the corresponding flavor.
8. Call the `ippiResizeGetBufferSize` function to obtain the size of the resize work buffer required for each tile processing. The `dstSize` parameter must be equal to the tile size.
9. Call `ippiResize<Filter>` for each tile (ROI). The `dstOffset` parameter must specify the image ROI offset with respect to the destination image origin. The `dstSize` parameter must be equal to the ROI size. Parameters `pSrc` and `pDst` must point to the beginning of the source and destination ROI in memory respectively. The source and destination ROIs must be fully accessible in memory.

You can process tiles in any order. When using multiple threads you can process all tiles simultaneously.

---

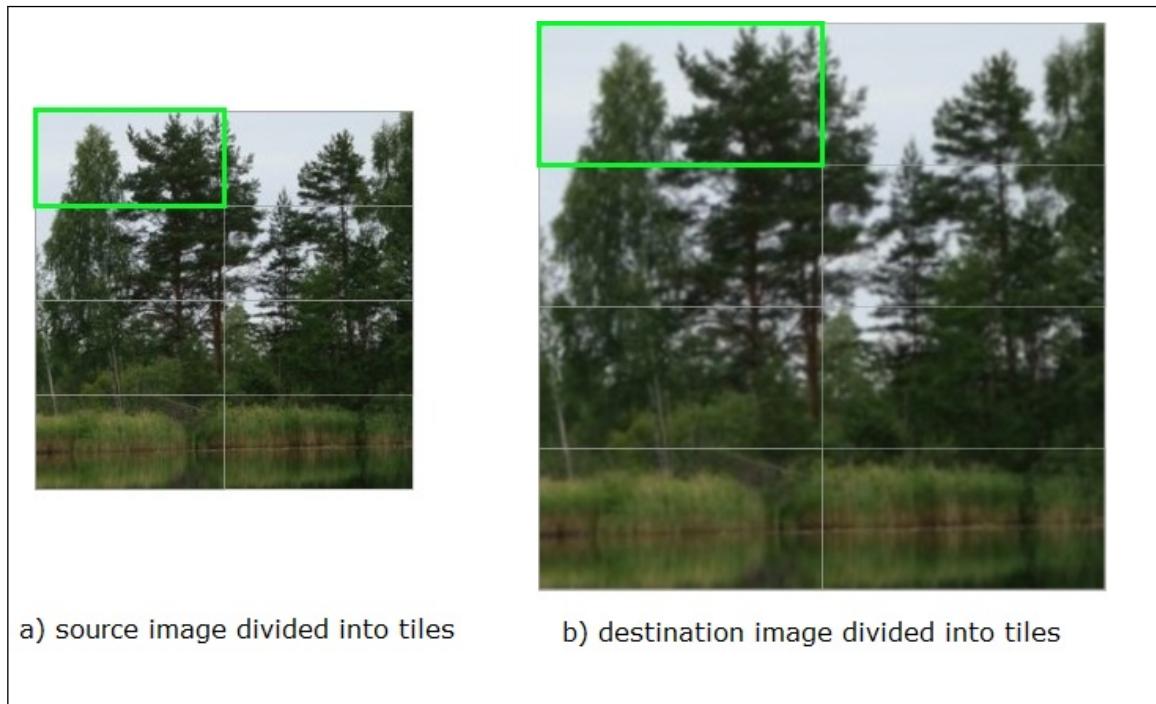
### NOTE

If you resize a tiled image with the Super Sampling algorithm, and the source image width to destination image width ratio is  $m/n$ , you can reach better performance of resize operation if all destination tiles have width that is a multiple of  $n$ .

---

Figure *Tiling Image Resize* shows the resize of the image divided into tiles.

### Tiling Image Resize



### Example

The code example below demonstrates a multithreading resize operation using OpenMP\* with parallelization only in the y direction:

```
#define MAX_NUM_THREADS 16

IppStatus tileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderSize borderSize = {0, 0, 0, 0};
    IppiBorderType border = ippBorderRepl;
    int numThreads, slice, tail;
    int bufSize1, bufSize2;
    IppiSize dstTileSize, dstLastTileSize;
    IppStatus pStatus[MAX_NUM_THREADS];

    /* Spec and init buffer sizes */
    status =ippiResizeGetSize_8u(srcSize, dstSize, ippLinear, 0, &specSize, &initSize);

    if (status != ippStsNoErr) return status;
```

```

/* Memory allocation */
pInitBuf = ippsMalloc_8u(initSize);
pSpec     = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);

if (pInitBuf == NULL || pSpec == NULL)
{
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    return ippStsNoMemErr;
}

/* Filter initialization */
status =ippiResizeLinearInit_8u(srcSize, dstSize, pSpec);
ippsFree(pInitBuf);

if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

status =ippiResizeGetBorderSize_8u(pSpec, &borderSize);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* General transform function */
/* Parallelized only by Y-direction here */
#pragma omp parallel num_threads(MAX_NUM_THREADS)
{
    #pragma omp master
    {
        numThreads = omp_get_num_threads();
        slice = dstSize.height / numThreads;
        tail   = dstSize.height % numThreads;

        dstTileSize.width = dstLastTileSize.width = dstSize.width;
        dstTileSize.height = slice;
        dstLastTileSize.height = slice + tail;

        ippิResizeGetBufferSize_8u(pSpec, dstTileSize, ippC3, &bufSize1);
        ippิResizeGetBufferSize_8u(pSpec, dstLastTileSize, ippC3, &bufSize2);

        pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
    }

    #pragma omp barrier
    {
        if (pBuffer)
        {
            Ipp32u i;
            Ipp8u *pSrcT, *pDstT;
            Ipp8u *pOneBuf;
            IppiPoint srcOffset = {0, 0};
            IppiPoint dstOffset = {0, 0};
            IppiSize srcSizeT = srcSize;

```

```

IppiSize dstSizeT = dstTileSize;

i = omp_get_thread_num();
dstSizeT.height = slice;
dstOffset.y += i * slice;

if (i == numThreads - 1) dstSizeT = dstLastTileSize;

pStatus[i] =ippiResizeGetSrcRoi_8u(pSpec, dstOffset, dstSizeT, &srcOffset,
&srcSizeT);

if (pStatus[i] == ippStsNoErr)
{
    pSrcT = (Ipp8u*)((char*)pSrc + srcOffset.y * srcStep);
    pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);

    pOneBuf = pBuffer + i * bufSize1;

    pStatus[i] =ippiResizeLinear_8u_C3R (pSrcT, srcStep, pDstT, dstStep,
dstOffset, dstSizeT, border, 0, pSpec, pOneBuf);
}
}
}

ippsFree(pSpec);

if (pBuffer == NULL) return ippStsNoMemErr;

ippsFree(pBuffer);

for (Ipp32u i = 0; i < numThreads; ++i)
{
    /* Return bad status */
    if(pStatus[i] != ippStsNoErr) return pStatus[i];
}

return status;
}

```

## Resizing a Tiled Image with Prior Initialization for Each Tile

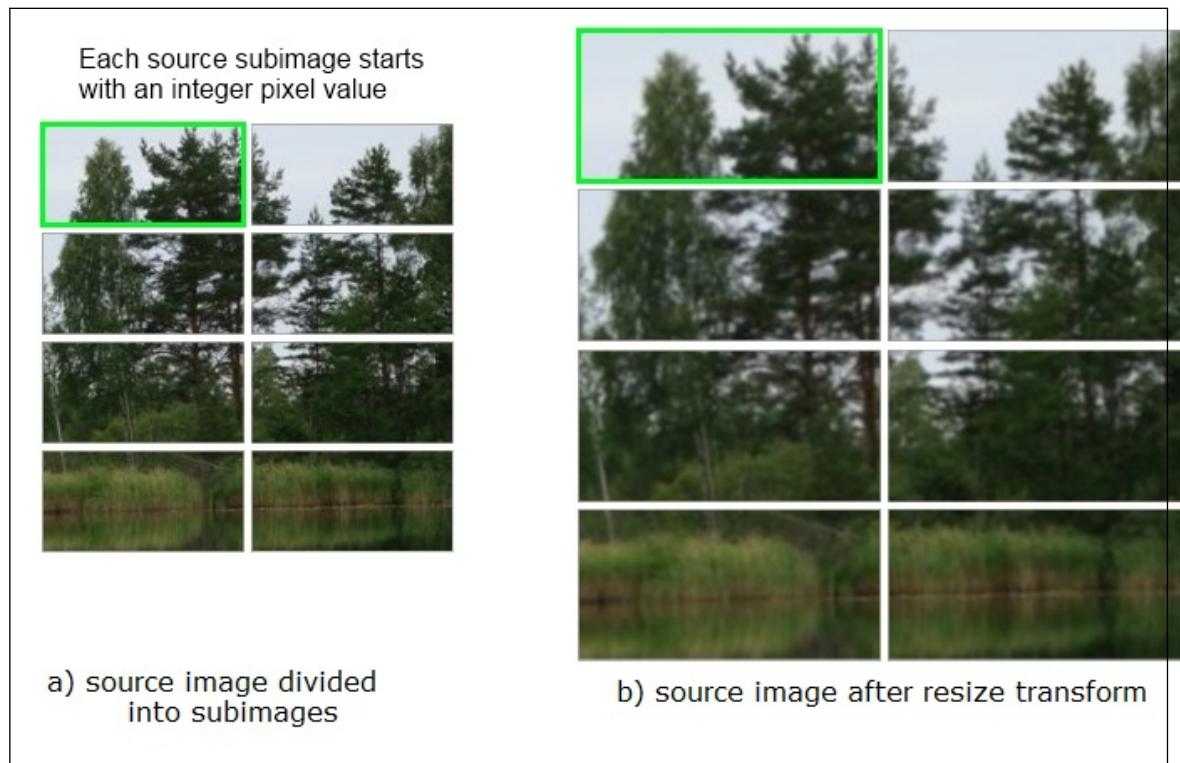
You can apply this approach only in cases when the destination image can be divided into tiles so that each destination tile corresponds to a source image tile that starts with an integer pixel value origin. For example, if the ratio of the source and destination images sizes is 2/3, the destination image can be divided into 3x3 tiles, each of which corresponds to the source image tile 2x2.

This approach is useful if there are restrictions on memory size when processing an image, or if the image size is large and `ippiResizeGetBufferSize` function returns `ippStsSizeErr` error. The initialization data for a tile is less than the same data for the whole image.

Each tile of the source image can be considered as an independent image that can be resized. For interior tile processing, the border must be always of the `ippBorderInMem` type. If you need to replicate any borders of the source image origin, you should combine the border type of the outer tiles so that interior tiles edges have border in memory and external tile borders are of the specified border type. This approach enables the right linking order of tiles.

Figure *Resize of the Image Divided into Subimages* shows the approach, when the source image is divided into several subimages that are resized independently.

### Resize of the Image Divided into Subimages



### Example

The code example below divides the source image into tiles and resizes each image independently:

```
IppStatus separateTileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcTileSize, Ipp32s srcStep,
Ipp8u* pDst, IppiSize dstTileSize, Ipp32s dstStep, Ipp32s xNumTiles, Ipp32s yNumTiles)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppStatus status = ippStsNoErr;

    /* tiles cicle */
    for (Ipp32s j = 0; j < xNumTiles; j++)
    {
        for (Ipp32s i = 0; i < yNumTiles; i++)
        {
            /* calculation of the destination image ROI offset */
            IppiPoint dstOffset = {j * dstTileSize.width, i * dstTileSize.height};
            Ipp8u* pDstT = pDst + dstStep * dstOffset.y + dstOffset.x * numChannels *
sizeof(Ipp8u);

            /* calculation of the source image ROI offset */
            IppiPoint srcOffset = {j * srcTileSize.width, i * srcTileSize.height};
            Ipp8u* pSrcT = pSrc + srcStep * srcOffset.y + srcOffset.x * numChannels *
sizeof(Ipp8u);
        }
    }
}
```

```
IppiBorderType borderT = ippBorderRepl;

IppiPoint dstOffsetZero = {0, 0};

/* correction of the border type for the tile processing */
if (j > 0) /* the processed tile is not on the left image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemLeft);
}
if (j < xNumTiles - 1) /* the processed tile is not on the right image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemRight);
}
if (i > 0) /* the processed tile is not on the top image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemTop);
}
if (i < yNumTiles - 1) /* the processed tile is not on the bottom image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemBottom);
}

/* Spec and init buffer sizes */
status =ippiResizeGetSize_8u(srcTileSize, dstTileSize, ippLanczos, 0, &specSize,
&initSize);

if (status != ippStsNoErr) return status;

/* Memory allocation */
pInitBuf = ippsMalloc_8u(initSize);
pSpec     = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);

if (pInitBuf == NULL || pSpec == NULL)
{
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    return ippStsNoMemErr;
}

/* Filter initialization */
status =ippiResizeLanczosInit_8u(srcTileSize, dstTileSize, 3, pSpec, pInitBuf);
ippsFree(pInitBuf);

if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* work buffer size */
status =ippiResizeGetBufferSize_8u(pSpec, dstTileSize, numChannels, &bufSize);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}
```

```

pBuffer = ippsMalloc_8u(bufSize);
if (pBuffer == NULL)
{
    ippsFree(pSpec);
    return ippStsNoMemErr;
}

/* Resize processing */
status =ippiResizeLanczos_8u_C3R(pSrcT, srcStep, pDstT, dstStep, dstOffsetZero,
dstTileSize, borderT, 0, pSpec, pBuffer);

ippsFree(pSpec);
ippsFree(pBuffer);

if (status != ippStsNoErr) return status;
}

return ippStsNoErr;
}

```

## See Also

[User-defined Border Types](#)

### ResizeGetSize

*Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.*

### Syntax

#### Case 1: Processing images of 32-bit sizes

```
IppStatus ippiResizeGetSize_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, int*
pInitBufSize);
```

Supported values for mod:

8u	16u	32f	64f
----	-----	-----	-----

```
IppStatus ippiResizeGetSize_16s(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, Ipp32s*
pInitBufSize);
```

#### Case 2: Processing images with platform-aware functions

```
IppStatus ippiResizeGetSize_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL*
pSpecSize, IppSizeL* pInitBufSize);
```

#### Case 3: Processing images with threading layer (TL) functions

```
IppStatus ippiResizeGetSize_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL*
pSpecSize, IppSizeL* pInitBufSize);
```

### Include Files

ippi.h

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>srcSize</code>	Size, in pixels, of the source image.
<code>dstSize</code>	Size, in pixels, of the destination image.
<code>interpolation</code>	Interpolation method. Supported values: <code>ippNearest</code> , <code>ippLinear</code> , <code>ippCubic</code> , <code>ippLanczos</code> , and <code>ippSuper</code> .
<code>antialiasing</code>	Supported values: 1 - resizing with antialiasing, 0 - resizing without antialiasing.
<code>dataType</code>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> (only for linear interpolation).
<code>pSpecSize</code>	Pointer to the size, in bytes, of the specification structure.
<code>pInitBufSize</code>	Pointer to the size, in bytes, of the temporary buffer required for initialization of the specification structure.

## Description

This function computes the size of the specification structure and the size of the external buffer for the following functions depending on the `interpolation` parameter value:

- `ippiResizeAntialiasingCubicInit`, `ippiResizeAntialiasingLanczosInit`, or `ippiResizeAntialiasingLinearInit` for resizing with antialiasing
- `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, or `ippiResizeSuperInit` for resizing without antialiasing

Interpolation algorithms have the following filter sizes:

Nearest Neighbor	1x1
Linear	2x2
Cubic	4x4
2-lobed Lanczos	4x4

---

**NOTE** The `ippiResizeGetSize` function always returns non-zero value for the `pInitBufSize` parameter, even if the temporary buffer is not required for the specification structure initialization. The temporary buffer is only required when initializing the specification structure for the following functions: `ippiResizeAntialiasingCubic`, `ippiResizeAntialiasingLanczos`, `ippiResizeAntialiasingLanczos`, `ippiResizeLanczos`, and `ippiResizeCubic`.

---

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsResizeNoOperation	Indicates an error if width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>If the source image size is less than the filter size of the chosen interpolation method (except <code>ippSuper</code>).</li> <li>If one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for <code>ippSuper</code> method only).</li> <li>If the width or height of the source or destination image is negative.</li> </ul>
ippStsExceededSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>If at least one of the computed values exceeds the maximum of the data type positive value pointed by <code>pSpecSize</code> or <code>pInitBufSize</code> correspondingly (the size of one of the processed images is too large).</li> <li>If width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).</li> </ul>
ippStsInterpolationErr	Indicates an error if <code>interpolation</code> has an illegal value.
ippStsNoAntialiasing	Indicates a warning if the specified interpolation method does not support antialiasing.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.
ippStsDataTypeErr	Indicates an error if <code>dataType</code> has an illegal value.

## ResizeGetBufferSize

Computes the size of the external buffer for image resizing.

---

### Syntax

#### Case 1: Single precision

```
IppStatusippiResizeGetBufferSize_<mod>(const IppiResizeSpec_32f* pSpec, IppiSize dstSize, Ipp32u numChannels, int* pBufSize);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

#### Case 2: Double precision

```
IppStatusippiResizeGetBufferSize_64f(const IppiResizeSpec_64f* pSpec, IppiSize dstSize, Ipp32u numChannels, int* pBufSize);
```

#### Case 3: Processing images with platform-aware functions

```
IppStatusippiResizeGetBufferSize_L(const IppiResizeSpec* pSpec, IppiSizeL dstSize, Ipp32u numChannels, IppSizeL* pBufSize);
```

### Case 4: Processing images with threading layer (TL) functions

```
IppStatusippiResizeGetBufferSize_LT(const IppiResizeSpec_LT* pSpec, IppSizeL* pBufSize);
```

#### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

#### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

#### Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstSize</i>	Size in pixels of the destination image.
<i>numChannels</i>	Number of channels, possible values: 1, 3, or 4.
<i>pBufSize</i>	Pointer to the size, in bytes, of the external buffer.

#### Description

This function computes the size of the external buffer for image resizing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetBufferSize` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, and `ippiResizeSuperInit`.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pBufferSize</i> pointer is NULL.
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsNumChannelErr</code>	Indicates an error if the value of <i>numChannels</i> is illegal.
<code>ippStsSizeErr</code>	Indicates an error condition If width or height of the destination image is negative.
<code>ippStsSizeWrn</code>	Indicates a warning if the destination image size is more than the destination image origin size.
<code>ippStsExceededSizeErr</code>	Indicates an error If at least one of the computed values exceeds maximum of <code>IppSizeL</code> type positive value (the size of one of the processed images is too large).

## ResizeGetBorderSize

*Computes the size of possible borders for the resize transform.*

### Syntax

#### Case 1: Interpolation with single precision

```
IppStatusippiResizeGetBorderSize_<mod>(const IppiResizeSpec_32f* pSpec,  
IppiBorderSize* borderSize);
```

Supported values for mod:

8u	16u	16s	32f
----	-----	-----	-----

#### Case 2: Interpolation with double precision

```
IppStatusippiResizeGetBorderSize_64f(const IppiResizeSpec_64f* pSpec, IppiBorderSize*  
borderSize);
```

#### Case 3: Processing images with platform-aware functions

```
IppStatusippiResizeGetBorderSize_L(const IppiResizeSpec* pSpec, IppiBorderSize*  
borderSize);
```

#### Case 4: Processing images with threading layer (TL) functions

```
IppStatusippiResizeGetBorderSize_LT(const IppiResizeSpec_LT* pSpec, IppiBorderSize*  
borderSize);
```

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

### Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>borderSize</i>	Size in pixels of necessary borders.

### Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetBorderSize` function, you need to initialize the *pSpec* parameter by calling one of the following functions: [ippiResizeNearestInit](#), [ippiResizeLinearInit](#), [ippiResizeCubicInit](#), [ippiResizeLanczosInit](#), and [ippiResizeSuperInit](#).

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsBorderErr	Indicates an error if <code>border</code> has an illegal value.

## ResizeGetSrcOffset

*Computes the offset of the source image for resizing by tile processing.*

### Syntax

#### Single precision

```
IppStatusippiResizeGetSrcOffset_<mod>(const IppiResizeSpec_32f* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

#### Double precision

```
IppStatusippiResizeGetSrcOffset_64f(const IppiResizeSpec_64f* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

#### Processing images with platform-aware functions

```
IppStatusippiResizeGetSrcOffset_L(const IppiResizeSpec* pSpec, IppiPointL dstOffset, IppiPointL* srcOffset);
```

### Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSpec</code>	Pointer to the spec structure for the resize filter.
<code>dstOffset</code>	Offset of the tiled destination image with respect to the destination image origin.
<code>srcOffset</code>	Offset of the source image.

## Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcOffset` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, or `ippiResizeSuperInit`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.

## ResizeGetSrcRoi

*Computes the ROI of the source image for resizing by tile processing.*

---

## Syntax

### Single precision

```
IppStatus ippiResizeGetSrcRoi_<mod>(const IppiResizeSpec_32f* pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

### Double precision

```
IppStatus ippiResizeGetSrcRoi_64f(const IppiResizeSpec_64f* pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

### Processing images with platform-aware functions

```
IppStatus ippiResizeGetSrcRoi_L(const IppiResizeSpec* pSpec, IppiPointL dstRoiOffset, IppiSizeL dstRoiSize, IppiPointL* srcRoiOffset, IppiSizeL* srcRoiSize);
```

## Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstRoiOffset</i>	Offset of the tiled destination image ROI.

---

<i>dstRoiSize</i>	Size of the tiled destination image ROI.
<i>srcRoiOffset</i>	Offset of the source image ROI.
<i>srcRoiSize</i>	Pointer to the size of the source image ROI.

## Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcRoi` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, or `ippiResizeSuperInit`.

### NOTE

If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning if the destination ROI exceeds the destination image origin.

## ResizeSetMode

*Sets the rounding mode for resize functions*

---

### Syntax

```
IppStatusippiResizeSetMode(IppHintAlgorithm hint, IppiResizeSpec* pSpec
```

### Include Files

`ippi.h`

### Parameters

<i>hint</i>	Rounding mode for processing <code>Ipp8u</code> data. Possible values are:
<code>ippAlgHintFast</code>	Fast rounding mode (default).
<code>ippAlgHintAccurate</code>	Accurate rounding mode.
<i>pSpec</i>	Pointer to the specification structure for the resize filter.

## Description

This function sets the `roundMode` for the resize algorithm.

If you provide the *hint* parameter with the `ippAlgHintFast` value, a faster but less accurate mode will be used. In this case, output pixel values can differ from the exact result by 1. If you choose `ippAlgHintAccurate`, a more accurate but slower mode will be used and all output pixel values will be exact.

Before using this function, initialize the specification structure using the initialization function for a required interpolation method.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsAccurateModeNotSupported</code>	Indicates an error when the rounding mode is not supported for the selected data type. The rounding result can be inexact.

## ResizeNearestInit

*Initializes the specification structure for image resizing with the nearest neighbor interpolation method.*

---

## Syntax

```
IppStatusippiResizeNearestInit_<mod>(IppiSize srcSize, IppiSize dstSize,  
IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

## Platform-aware function

```
IppStatusippiResizeNearestInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, IppiResizeSpec* pSpec);
```

## Threading layer (TL) function

```
IppStatusippiResizeNearestInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>srcSize</code>	Size, in pixels, of the source image.
----------------------	---------------------------------------

---

<i>dstSize</i>	Size, in pixels, of the destination image.
<i>dataType</i>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>numChannels</i>	Number of channels. Possible values are 1, 3, and 4.

## Description

This function initializes the specification structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the specification structure object, call the `ippiResizeGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is negative.
<code>ippStsDataTypeErr</code>	Indicates an error if <i>dataType</i> has an illegal value.
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

## See Also

`ResizeGetSize` Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## ResizeNearest

*Changes an image size using the nearest neighbor interpolation method.*

---

## Syntax

```
IppStatus ippiResizeNearest_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const
IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Platform-aware functions

```
IppStatusippiResizeNearest_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppPointL dstOffset, IppSizeL dstSize, const
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Threading layer (TL) functions

```
IppStatusippiResizeNearest_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<code>dstOffset</code>	Offset of the tiled destination image with respect to the destination image origin.
<code>dstSize</code>	Size of the destination image in pixels.

---

<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

Function flavors operating on images of 64-bit sizes (with the `L` suffix) can process only whole images.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the `ippiResizeNearest` function, you need to initialize the resize specification structure using the `ippiResizeNearestInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error when pointer to the spec structure is invalid.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not data type multiple.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image size is more than the destination image origin size.

## See Also

### [ROI Processing in Geometric Transforms](#)

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

[ResizeNearestInit](#) Initializes the specification structure for image resizing with the nearest neighbor interpolation method.

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

## [ResizeLinearInit](#)

*Initializes the specification structure for image resizing with the linear interpolation method.*

## Syntax

```
IppStatusippiResizeLinearInit_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

```
IppStatusippiResizeLinearInit_64f(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_64f* pSpec);
```

### Platform-aware function

```
IppStatusippiResizeLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiResizeSpec* pSpec);
```

```
IppStatusippiResizeLinearInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppHintAlgorithm hint, IppiResizeSpec* pSpec);
```

### Threading layer (TL) function

```
IppStatusippiResizeLinearInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>srcSize</code>	Size, in pixels, of the source image.	
<code>dstSize</code>	Size, in pixels, of the destination image.	
<code>dataType</code>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .	
<code>hint</code>	Computation algorithm for processing <code>ipp8u</code> data. Possible values are:	
	<code>ippAlgHintNone</code>	The hint is absent. Equivalent to <code>ippAlgHintFast</code> .
	<code>ippAlgHintFast</code>	A faster but less accurate algorithm.
	<code>ippAlgHintAccurate</code>	A slower but more accurate algorithm.
<code>pSpec</code>	Pointer to the specification structure for the resize filter.	

*numChannels* Number of channels. Possible values are 1, 3, and 4.

## Description

This function initializes the specification structure for the resize algorithm with the linear interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

### NOTE

The function with the parameter *hint* allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter *hint* this function initializes the specification structure for the faster but less accurate algorithm for `Ipp8u` data type.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error if: <ul style="list-style-type: none"> <li>width or height of the source or destination image is negative</li> <li>the source image size is less than the size of a 2x2 linear filter</li> </ul>
<code>ippStsDataTypeErr</code>	Indicates an error if <i>dataType</i> has an illegal value.
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware or TL functions).
<code>ippStsNumChannelErr</code>	Indicates an error if <i>numChannel</i> has an illegal value.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## ResizeLinear

*Changes an image size using the linear interpolation method.*

## Syntax

### Case 1: Single precision

```
IppStatus ippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	

8u\_C4R      16u\_C4R      16s\_C4R      32f\_C4R

```
IppStatusippiResizeLinear_32f_C3R(const Ipp32f* pSrc, const Ipp32s srcStep, Ipp32f* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp32f* pBorderColor, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

### **Case 2: Double precision**

```
IppStatusippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp<datatype>* pBorderColor, const IppiResizeSpec_64f* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

64f\_C1R

64f\_C3R

64f\_C4R

### **Case 3: Platform-aware functions**

```
IppStatusippiResizeLinear_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, IppiBorderType border, const Ipp<datatype>* pBorderColor, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R    16u\_C1R    16s\_C1R    32f\_C1R    64f\_C1R

8u\_C3R    16u\_C3R    16s\_C3R    32f\_C3R    64f\_C3R

8u\_C4R    16u\_C4R    16s\_C4R    32f\_C4R    64f\_C4R

### **Case 3: Threading layer (TL) functions**

```
IppStatusippiResizeLinear_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp8u* pBorderColor, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R    16u\_C1R    16s\_C1R    32f\_C1R    64f\_C1R

8u\_C3R    16u\_C3R    16s\_C3R    32f\_C3R    64f\_C3R

8u\_C4R    16u\_C4R    16s\_C4R    32f\_C4R    64f\_C4R

## **Include Files**

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## **Domain Dependencies**

Flavors declared in ippi.h:

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_t1.h:

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_t1.lib, ippi\_t1.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.										
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.										
<i>pDst</i>	Pointer to the destination image.										
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.										
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.										
<i>dstSize</i>	Size of the destination image in pixels.										
<i>border</i>	Type of border. Possible values are: <table border="0"> <tr> <td><i>ippBorderRepl</i></td><td>Border is replicated from the edge pixels.</td></tr> <tr> <td><i>ippBorderInMem</i></td><td>Border is obtained from the source image pixels in memory.</td></tr> <tr> <td><i>ippBorderMirror</i></td><td>Border is obtained from the source image boundary pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i>.</td></tr> <tr> <td><i>ippBorderMirrorR</i></td><td>Border is obtained from the source image boundary pixels. The anchor cell value is replicated to the border pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i>.</td></tr> <tr> <td></td><td>Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <i>ippBorderRepl</i> type and the <i>ippBorderInMemTop</i>, <i>ippBorderInMemBottom</i>, <i>ippBorderInMemLeft</i>, <i>ippBorderInMemRight</i> types.</td></tr> </table>	<i>ippBorderRepl</i>	Border is replicated from the edge pixels.	<i>ippBorderInMem</i>	Border is obtained from the source image pixels in memory.	<i>ippBorderMirror</i>	Border is obtained from the source image boundary pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i> .	<i>ippBorderMirrorR</i>	Border is obtained from the source image boundary pixels. The anchor cell value is replicated to the border pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i> .		Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <i>ippBorderRepl</i> type and the <i>ippBorderInMemTop</i> , <i>ippBorderInMemBottom</i> , <i>ippBorderInMemLeft</i> , <i>ippBorderInMemRight</i> types.
<i>ippBorderRepl</i>	Border is replicated from the edge pixels.										
<i>ippBorderInMem</i>	Border is obtained from the source image pixels in memory.										
<i>ippBorderMirror</i>	Border is obtained from the source image boundary pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i> .										
<i>ippBorderMirrorR</i>	Border is obtained from the source image boundary pixels. The anchor cell value is replicated to the border pixels. Value can be specified via <i>ippiResizeLinear*</i> and <i>ippiResizeLinear*_LT</i> .										
	Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <i>ippBorderRepl</i> type and the <i>ippBorderInMemTop</i> , <i>ippBorderInMemBottom</i> , <i>ippBorderInMemLeft</i> , <i>ippBorderInMemRight</i> types.										
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <i>ippBorderConst</i> border type.										
<i>pSpec</i>	Pointer to the specification structure for the resize filter.										
<i>pBuffer</i>	Pointer to the work buffer.										

## Description

This function changes the size of an image using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function `ippiResizeLinear` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the `ippiResizeGetBorderSize` function for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the `ippiResizeLinear` function, you need to initialize the specification structure using the `ippiResizeLinearInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when the <code>border</code> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error when pointer to the specification structure is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not a multiple of data type.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image size is more than the destination image origin size.

## See Also

### ROI Processing in Geometric Transforms

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

### User-defined Border Types

[ResizeGetBorderSize](#) Computes the size of possible borders for the resize transform.

[ResizeLinearInit](#) Initializes the specification structure for image resizing with the linear interpolation method.

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

## ResizeCubicInit

*Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.*

### Syntax

```
IppStatusippiResizeCubicInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp32f valueB,  
Ipp32f valueC, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

### Platform-aware functions

```
IppStatusippiResizeCubicInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

```
IppStatusippiResizeCubicInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize, Ipp32f valueB,  
Ipp32f valueC, IppHintAlgorithm hint, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

### Threading layer (TL) functions

```
IppStatusippiResizeCubicInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec_LT* pSpec,  
Ipp8u* pInitBuf);
```

### Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

### Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

### Parameters

<code>srcSize</code>	Size, in pixels, of the source image.
<code>dstSize</code>	Size, in pixels, of the destination image.
<code>dataType</code>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<code>hint</code>	Computation algorithm for processing <code>Ipv8u</code> data. Possible values are:
<code>ippAlgHintNone</code>	The <code>hint</code> is absent. Equivalent to <code>ippAlgHintFast</code> .
<code>ippAlgHintFast</code>	A faster but less accurate algorithm.

	ippAlgHintAccurate A slower but more accurate algorithm.
<i>numChannels</i>	Number of channels, possible values: 1, 3, or 4.
<i>valueB</i>	The first parameter for cubic filters.
<i>valueC</i>	The second parameter for cubic filters.
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pInitBuf</i>	Pointer to the temporary buffer for the cubic filter initialization.

## Description

This function initializes the specification structure for the resize algorithm with interpolation with two-parameter cubic filters. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

Before using this function, you need to calculate the size of the specification structure and the external buffer *pInitBuf* using the `ippiResizeGetSize` function for the corresponding flavor.

### NOTE

The function with the parameter *hint* allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter *hint* this function initializes the specification structure for the faster but less accurate algorithm for `Ipp8u` data type.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error if: <ul style="list-style-type: none"> <li>• width or height of the source or destination image is negative</li> <li>• the source image size is less than the size of a 2x2 linear filter</li> </ul>
<code>ippStsDataTypeErr</code>	Indicates an error if <i>dataType</i> has an illegal value.
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
<code>ippStsNumChannelErr</code>	Indicates an error if <i>numChannel</i> has an illegal value.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## ResizeCubic

*Changes an image size using interpolation with two-parameter cubic filters.*

---

## Syntax

```
IppStatusippiResizeCubic_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Platform-aware functions

```
IppStatusippiResizeCubic_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Threading layer (TL) functions

```
IppStatusippiResizeCubic_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<datatype>*
pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance, in bytes, between the startings of consecutive lines in the destination image buffer.
<code>dstOffset</code>	Offset of the tiled destination image with respect to the destination image origin.
<code>dstSize</code>	Size of the destination image in pixels.
<code>border</code>	Type of border. Possible values are:  ippBorderRepl      Border is replicated from the edge pixels. ippBorderInMem     Border is obtained from the source image pixels in memory. ippBorderMirror    Border is obtained from the source image boundary pixels. Value can be specified via <code>ippiResizeCubic*</code> and <code>ippiResizeCubic*_LT</code> . ippBorderMirrorR   Border is obtained from the source image boundary pixels. The anchor cell value is replicated to the border pixels. Value can be specified via <code>ippiResizeCubic*</code> and <code>ippiResizeCubic*_LT</code> .  Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <code>ippBorderRepl</code> type and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> types.
<code>pBorderValue</code>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pSpec</code>	Pointer to the spec structure for the resize filter.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

This function changes an image size using interpolation with two-parameter cubic filters. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi`

function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function `ippiResizeCubic` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the `ippiResizeGetBorderSize` function for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the `ippiResizeLinear` function, you need to initialize the specification structure using the `ippiResizeCubicInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when the <code>border</code> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error when pointer to the spec structure is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not data type multiple.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image size is more than the destination image origin size.

## See Also

### ROI Processing in Geometric Transforms

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

### User-defined Border Types

[ResizeGetBorderSize](#) Computes the size of possible borders for the resize transform.

[ResizeCubicInit](#) Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

## ResizeLanczosInit

*Initializes the specification structure for image resizing with the Lanczos interpolation method.*

## Syntax

```
IppStatusippiResizeLanczosInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

Supported values for mod:

8u	16u	16s	32f
----	-----	-----	-----

## Platform-aware functions

```
IppStatusippiResizeLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

```
IppStatusippiResizeLanczosInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize, Ipp32u numLobes, IppHintAlgorithm hint, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

## Threading layer (TL) functions

```
IppStatusippiResizeLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

## Parameters

<i>srcSize</i>	Size, in pixels, of the source image.
<i>dstSize</i>	Size, in pixels, of the destination image.
<i>dataType</i>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<i>hint</i>	Computation algorithm for processing <code>Ipp8u</code> data. Possible values are:  ippAlgHintNone      The hint is absent. Equivalent to <code>ippAlgHintFast</code> . ippAlgHintFast      A faster but less accurate algorithm. ippAlgHintAccurate   A slower but more accurate algorithm.
<i>numChannels</i>	Number of image channels. Possible values are 1, 3, or 4.
<i>numLobes</i>	Parameter for Lanczos filters. Possible values are 2 or 3.

---

<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pInitBuf</i>	Pointer to the temporary buffer for the cubic filter initialization.

## Description

This function initializes the specification structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function, depending on the value of the *numLobes* parameter.

Before using this function, you need to calculate the size of the specification structure and the external buffer *pInitBuf* using the `ippiResizeGetSize` function for the corresponding flavor.

---

### NOTE

The function with the parameter *hint* allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter *hint* this function initializes the specification structure for the faster but less accurate algorithm for `Ipp8u` data type.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsResizeNoOperation</code>	Indicates an error when width or height of the image is equal to zero.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsDataTypeErr</code>	Indicates an error if <i>dataType</i> has an illegal value.
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• If width or height of the source or destination image is negative,</li> <li>• If the source image size is less than the Lanczos interpolation filter size: 4x4 for the 2-lobed Lanczos function, or 6x6 for the 3-lobed Lanczos function.</li> </ul>

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## ResizeLanczos

*Changes an image size using interpolation with the Lanczos filter.*

---

## Syntax

```
IppStatusippiResizeLanczos_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Platform-aware functions

```
IppStatusippiResizeLanczos_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Threading layer (TL) functions

```
IppStatusippiResizeLanczos_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<code>dstOffset</code>	Offset of the tiled destination image with respect to the destination image origin.
<code>dstSize</code>	Size of the destination image in pixels.
<code>border</code>	Type of border. Possible values are:  <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <code>ippBorderRepl</code> type and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> types.
<code>pBorderValue</code>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pSpec</code>	Pointer to the specification structure for the resize filter.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function `ippiResizeLanczos` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the `ippiResizeGetBorderSize` function for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, a combined approach is applied.

Before using the `ippiResizeLanczos` function, you need to initialize the specification structure using the `ippiResizeLanczosInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error when the <code>border</code> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error when pointer to the specification structure is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the destination image is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not data type multiple.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image size is more than the destination image origin size.

## See Also

[ROI Processing in Geometric Transforms](#)

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

[User-defined Border Types](#)

[ResizeGetBorderSize](#) Computes the size of possible borders for the resize transform.

[ResizeLanczosInit](#) Initializes the specification structure for image resizing with the Lanczos interpolation method.

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

## ResizeSuperShiftInit

*Initializes the specification structure for image resizing with the super sampling interpolation method and floating point shifts.*

---

## Syntax

```
IppStatus ippiResizeSuperInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp64f xShift,
Ipp64f yShift, IppiBorderType borderType, const Ipp8u * borderVal, int smoothEdge, int
numChannels, IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

8u	16u	16s	32f
----	-----	-----	-----

## Include Files

`ippi.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcSize</code>	Size, in pixels, of the source image.
<code>dstSize</code>	Size, in pixels, of the destination image.
<code>xShift</code>	Shift value in the <code>x</code> direction.
<code>yShift</code>	Shift value in the <code>y</code> direction.
<code>borderType</code>	Type of border. Supported values: <code>ippBorderTransp</code> . Outer pixels are not processed.
<code>borderVal</code>	Not used. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>smoothEdge</code>	Flag for edge smoothing. Supported values: 0 - transformation without edge smoothing. 1 - transformation with edge smoothing.
<code>pSpec</code>	Pointer to the specification structure for the resize filter.
<code>numChannels</code>	Number of channels. Possible values are 1, 3, and 4.

## Description

This function initializes the specification structure for the resize algorithm with the super sampling interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if: - width or height of the destination image is equal to zero. - smoothing edge is on and width or height of the destination image is equal to one.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"><li>• Indicates that one of the specified dimensions of the source image is less than the corresponding dimension of the destination image, or that the width or height of the source or destination image is negative.</li><li>• If the width or height of the source or destination image is negative.</li></ul>

ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
ippStsDataTypeErr	Indicates an error if <i>dataType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error if the number of channels is not supported.
ippStsBorderErr	Indicates an error if specified <i>borderType</i> is not supported.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## See Also

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

## ResizeSuperInit

*Initializes the specification structure for image resizing with the super sampling interpolation method.*

---

## Syntax

```
IppStatusippiResizeSuperInit_<mod>(IppiSize srcSize, IppiSize dstSize,  
IppiResizeSpec_32f* pSpec);
```

Supported values for mod:

8u            16u            16s            32f

## Platform-aware function

```
IppStatusippiResizeSuperInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, IppiResizeSpec* pSpec);
```

## Threading layer (TL) function

```
IppStatusippiResizeSuperInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType  
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

## Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

## Parameters

<i>srcSize</i>	Size, in pixels, of the source image.
<i>dstSize</i>	Size, in pixels, of the destination image.
<i>dataType</i>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>numChannels</i>	Number of channels. Possible values are 1, 3, and 4.

## Description

This function initializes the specification structure for the resize algorithm with the super sampling interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• Indicates that one of the specified dimensions of the source image is less than the corresponding dimension of the destination image, or that the width or height of the source or destination image is negative.</li> <li>• If the width or height of the source or destination image is negative.</li> </ul>
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
<code>ippStsDataTypeErr</code>	Indicates an error if <i>dataType</i> has an illegal value.

## See Also

`ResizeGetSize` Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

## ResizeSuper

*Changes an image size using the super sampling interpolation method.*

---

## Syntax

```
IppStatusippiResizeSuper_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppPoint dstOffset, IppiSize dstSize, const
IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`    `16u_C1R`    `16s_C1R`    `32f_C1R`

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

### Platform-aware functions

```
IppStatusippiResizeSuper_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, const
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

### Threading layer (TL) functions

```
IppStatusippiResizeSuper_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

---

<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>dstSize</i>	Size of the destination image in pixels.
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using the super sampling interpolation method. This method only reduces the image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the `ippiResizeLinear` function, you need to initialize the resize structure using the `ippiResizeSuperInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

---

### NOTE

You can get better performance if you use the following scaling factors along the `x` and `y` axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error when pointer to the spec structure is invalid.
<code>ippStsStepErr</code>	Indicates an error when the step value is not data type multiple.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image size is more than the destination image origin size.

## See Also

### ROI Processing in Geometric Transforms

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

[ResizeSuperInit](#) Initializes the specification structure for image resizing with the super sampling interpolation method.

**ResizeGetBufferSize** Computes the size of the external buffer for image resizing.

### ResizeAntialiasingLinearInit

*Initializes the specification structure for image resizing with antialiasing using linear interpolation.*

### Syntax

```
IppStatusippiResizeAntialiasingLinearInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

### Platform-aware function

```
IppStatusippiResizeAntialiasingLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

### Threading layer (TL) function

```
IppStatusippiResizeAntialiasingLinearInit_LT(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

### Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dataType</i>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<i>numChannels</i>	Number of image channels. Possible values: 1, 3, or 4.
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pInitBuf</i>	Pointer to the temporary buffer for initialization of the linear filter.

### Description

This function initializes the `IppiResizeSpec_32f` structure for the resize operation with antialiasing using the linear interpolation method.

Before using this function, calculate the size of the temporary buffer and specification structure using the `ippiResizeGetSize` function with the `antialiasing` parameter equal to 1.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if width or height of the source or destination image is negative.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF).
ippStsDataTypeErr	Indicates an error if <i>dataType</i> has an illegal value.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

[Linear Interpolation](#)

## ResizeAntialiasingCubicInit

*Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.*

### Syntax

```
IppStatusippiResizeAntialiasingCubicInit(IppiSize srcSize, IppiSize dstSize, Ipp32f
valueB, Ipp32f valueC, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

### Platform-aware function

```
IppStatusippiResizeAntialiasingCubicInit_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec* pSpec, Ipp8u*
pInitBuf);
```

### Threading layer (TL) function

```
IppStatusippiResizeAntialiasingCubicInit_LT(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC,
IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

### Include Files

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore\_tl.lib, ippi\_tl.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>valueB</i>	The first parameter for cubic filters.
<i>valueC</i>	The second parameter for cubic filters.
<i>numChannels</i>	Number of image channels. Possible values: 1, 3, or 4.
<i>dataType</i>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<i>pSpec</i>	Pointer to the specification structure for the resize filter.
<i>pInitBuf</i>	Pointer to the temporary buffer for initialization of the cubic filter.

## Description

This function initializes the `IppiResizeSpec_32f` structure for the resize operation with antialiasing using interpolation with the two-parameter cubic filters.

Before using this function, calculate the size of the temporary buffer and specification structure using the `ippiResizeGetSize` function with the `antialiasing` parameter equal to 1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error if width or height of the source or destination image is negative.
<code>ippStsExceededSizeErr</code>	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
<code>ippStsDataTypeErr</code>	Indicates an error if <code>dataType</code> has an illegal value.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

[Cubic Interpolation](#)

## ResizeAntialiasingLanczosInit

*Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.*

## Syntax

```
IppStatus ippiResizeAntialiasingLanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

## Platform-aware function

```
IppStatusippiResizeAntialiasingLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

## Threading layer (TL) function

```
IppStatusippiResizeAntialiasingLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec,
Ipp8u* pInitBuf);
```

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>dstSize</code>	Size of the destination image, in pixels.
<code>numLobes</code>	Number of lobes for the Lanczos window. Possible values: 2 or 3.
<code>numChannels</code>	Number of image channels. Possible values: 1, 3, or 4.
<code>dataType</code>	Data type of the image. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , <code>ipp64f</code> .
<code>pSpec</code>	Pointer to the specification structure for the resize filter.
<code>pInitBuf</code>	Pointer to the temporary buffer for initialization of the cubic filter.

## Description

This function initializes the `IppiResizeSpec_32f` structure for the resize operation with antialiasing using interpolation with the Lanczos filter. The Lanczos interpolation method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function depending on the value of the `numLobes` parameter.

Before using this function, calculate the size of the temporary buffer and specification structure using the `ippiResizeGetSize` function with the `antialiasing` parameter equal to 1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .

ippStsNoOperation	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if width or height of the source or destination image is negative.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
ippStsDataTypeErr	Indicates an error if <i>dataType</i> has an illegal value.

## See Also

[ResizeGetSize](#) Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

[Lanczos Interpolation](#)

## ResizeAntialiasing

*Changes an image size using the chosen interpolation method with antialiasing.*

---

### Syntax

```
IppStatusippiResizeAntialiasing_<data_type>_<chan>(const Ipp<data_type>* pSrc, Ipp32s  
srcStep, Ipp<data_type>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,  
IppiBorderType border, const Ipp<data_type>* pBorderValue, const IppiResizeSpec_32f*  
pSpec, Ipp8u* pBuffer);
```

Supported values for *data\_type*:

8u        16u        16s        32f

Supported values for *chan*:

C1R        C3R        C4R

### Platform-aware functions

```
IppStatusippiResizeAntialiasing_<data_type>_<chan>_L(const Ipp<data_type>* pSrc,  
IppSizeL srcStep, Ipp<data_type>* pDst, IppSizeL dstStep, IppiPointL dstOffset,  
IppiSizeL dstSize, IppiBorderType border, const Ipp<data_type>* pBorderValue, const  
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *data\_type*:

8u        16u        16s        32f

Supported values for *chan*:

C1R        C3R        C4R

## Threading layer (TL) functions

```
IppStatusippiResizeAntialiasing_<data_type>_<chan>_LT(const Ipp<data_type>* pSrc,
IppSizeL srcStep, Ipp<data_type>* pDst, IppSizeL dstStep, IppiBorderType border, const
Ipp<data_type>* pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `data_type`:

8u	16u	16s	32f
----	-----	-----	-----

Supported values for `chan`:

C1R	C3R	C4R
-----	-----	-----

## Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_t1.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_t1.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_t1.lib`, `ippi_t1.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.	
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.	
<code>pDst</code>	Pointer to the destination image.	
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.	
<code>dstOffset</code>	Offset of the tiled destination image with respect to the destination image origin.	
<code>dstSize</code>	Size of the destination image, in pixels.	
<code>border</code>	Type of border. Possible values are:	
	<code>ippBorderRepl</code>	Border is replicated from the edge pixels.
	<code>ippBorderConst</code>	Border pixels are set to constants.
	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.

	ippBorderMirror	Border is obtained from the source image boundary pixels. Value can be specified via <code>ippiResizeAntialiasing*</code> and <code>ippiResizeAntialiasing*_L</code> .
	ippBorderMirrorR	Border is obtained from the source image boundary pixels. The anchor cell value is replicated to the border pixels. Value can be specified via <code>ippiResizeAntialiasing*</code> and <code>ippiResizeAntialiasing*_L</code> .
		Mixed borders are also supported. They can be obtained by the bitwise operation OR between the <code>ippBorderRepl</code> type and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> types.
<code>pBorderValue</code>		Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pSpec</code>		Pointer to the specification structure for the resize filter.
<code>pBuffer</code>		Pointer to the external buffer.

## Description

The `ippiResizeAntialiasing` function changes the size of an image using the chosen interpolation method with antialiasing. The interpolation method to be applied is defined by the function that you use for the resize filter initialization. Use this function to reduce the image size with minimization of moire artifacts. For more information about the implemented algorithm, refer to [SCHU92].

If you use `ippiResizeAntialiasing` to increase the image size, the function applies the same algorithm as one of the following resize functions, depending on the interpolation type chosen at the initialization stage: `ippiResizeLinear`, `ippiResizeCubic`, or `ippiResizeLanczos`.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. Define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the corresponding flavor of the `ippiResizeGetSrcRoi` function. To obtain the source image ROI origin offset, call the corresponding flavor of the `ippiResizeGetSrcOffset` function. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The `ippiResizeAntialiasing` function uses the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the corresponding flavor of the `ippiResizeGetBorderSize` function.

Before using the `ippiResizeAntialiasing` function, you need to initialize the `IppiResizeSpec_32f` structure using one of the following functions, depending on the interpolation method to be applied: `ippiResizeAntialiasingLinearInit`, `ippiResizeAntialiasingCubicInit`, or `ippiResizeAntialiasingLanczosInit`, and compute the size of the external buffer `pBuffer` using the corresponding flavor of the `ippiResizeGetBufferSize` function.

For more information about the supported border types, see [User-defined Border Types](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error when the <i>border</i> value is illegal.
ippStsContextMatchErr	Indicates an error when a pointer to the specification structure is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the destination image is negative.
ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

## Example

To better understand usage of the `ippiResizeAntialiasing` function, refer to the `ResizeAntialiasing.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

The figure below demonstrates the results of reducing a 1751x1044 image by five times with antialiasing (a) and without (b).



## See Also

[ROI Processing in Geometric Transforms](#)

[User-defined Border Types](#)

[ResizeGetSrcRoi](#) Computes the ROI of the source image for resizing by tile processing.

[ResizeGetSrcOffset](#) Computes the offset of the source image for resizing by tile processing.

[ResizeGetBorderSize](#) Computes the size of possible borders for the resize transform.

[ResizeGetBufferSize](#) Computes the size of the external buffer for image resizing.

[ResizeAntialiasingLinearInit](#) Initializes the specification structure for image resizing with antialiasing using linear interpolation.

[ResizeAntialiasingCubicInit](#) Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.

[ResizeAntialiasingLanczosInit](#) Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.

## ResizeFilterGetSize

*Calculates the size of the state structure for resizing filter.*

---

## Syntax

```
IppStatus ippiResizeFilterGetSize_8u_C1R(IppiSize srcRoiSize, IppiSize dstRoiSize,
IppiResizeFilterType filter, Ipp32u* pSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>filter</i>	Type of filter used in resizing; possible values: ippResizeFilterHann, ippResizeFilterLanczos.
<i>pSize</i>	Pointer to the size (in bytes) of the state structure.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function calculates the size *pSize* of the state structure required for the function [ippiResizeFilter](#). The type of filter is specified by the parameter *filter*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if the <i>pSize</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
ippStsNotSupportedModeErr	Indicates an error condition if <i>filter</i> has an invalid value.

## ResizeFilterInit

*Initializes the state structure for the resize filter.*

---

## Syntax

```
IppStatusippiResizeFilterInit_8u_C1R(IppiResizeFilterState* pState, IppiSize  
srcRoiSize, IppiSize dstRoiSize, IppiResizeFilterType filter);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pState</i>	Pointer to the state structure for resize filter.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.

---

<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>filter</i>	Type of filter used in resizing; possible values: <code>ippResizeFilterHann</code> , <code>ippResizeFilterLanczos</code>

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function initializes the state structure `pState` for the resizing filter used by the function `ippiResizeFilter`. The size of the structure must be computed by the function `ippiResizeFilterGetSize` beforehand. The type of filter is specified by the parameter `filter` and it must be the same as in the function `ippiResizeFilterGetSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pState</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if <code>filter</code> has an invalid value.

## ResizeFilter

Changes the size of an image using a generic filter.

## Syntax

```
IppStatus ippiResizeFilter_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiResizeFilterState* pState);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>pState</i>	Pointer to the state structure for the resize filter.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image *pSrc* using the special generic filters. The state structure *pState* contains the parameters of filtering and must be initialized by the function [ippiResizeFilterInit](#) beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.

## ResizeYUV420GetSize

*Computes sizes of the spec structure and the external buffer for the NV12 resize transform initialization.*

## Syntax

```
IppStatus ippiResizeYUV420GetSize(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s* pInitBufSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image in pixels.
<i>dstSize</i>	Size of the destination image in pixels.
<i>interpolation</i>	Interpolation method. Supported values are <code>ippLanczos</code> and <code>ippSuper</code> .
<i>antialiasing</i>	Antialiasing method.
<i>pSpecSize</i>	Pointer to the size in bytes of the spec structure.
<i>pInitBufSize</i>	Pointer to the size in bytes of the temporal buffer.

## Description

This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: [ResizeYUV420LanczosInit](#) and [ResizeYUV420SuperInit](#).

The size of the 2-Lobed Lanczos filter is 8x8.

**NOTE**

Antialiasing is currently not supported. The value for the *antialiasing* parameter must be equal to zero.

**Return Values**

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the image is equal to zero.
ippStsSizeErr	<p>Indicates an error in the following cases:</p> <ul style="list-style-type: none"> <li>• if width or height of the image is equal to 1,</li> <li>• if the source image size is less than a filter size of the chosen interpolation method (except <i>ippSuper</i>),</li> <li>• if one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for <i>ippSuper</i> method only),</li> <li>• if width or height of the source or destination image is negative,</li> <li>• if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of the one of the processed images is too large).</li> </ul> <p>if width or height of the source or destination image is negative.</p>
ippStsSizeWrn	Indicates a warning if width or height of the image is odd.
ippStsInterpolationErr	Indicates an error if <i>interpolation</i> has an illegal value.
ippStsNoAntialiasing	Indicates a warning if the specified interpolation method does not support antialiasing.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is currently not supported.

**ResizeYUV420GetSrcRoi**

Computes the ROI of the source image for NV12 resizing by tile processing.

**Syntax****Single precision**

```
IppStatusippiResizeYUV420GetSrcRoi(const IppiResizeYUV420Spec* pSpec, IppiPoint
dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstRoiOffset</i>	Offset of the tiled destination image ROI.
<i>dstRoiSize</i>	Size of the tiled destination image ROI.
<i>srcRoiOffset</i>	Offset of the source image ROI.
<i>srcRoiSize</i>	Pointer to the size of the source image ROI.

## Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetSrcRoi` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` or `ippiResizeYUV420SuperInit` functions.

---

### NOTE

If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if one of the fields of the <i>dstRoiOffset</i> parameter is odd.
<code>ippStsSizeErr</code>	Indicates an error if one of the fields of the <i>dstRoiSize</i> is less than 2.
<code>ippStsSizeWrn</code>	Indicates a warning if the destination ROI exceeds the destination image origin or contains odd values.

## ResizeYUV420LanczosInit

*Initializes the spec structure for the NV12 resize transform by interpolation with the Lanczos filter.*

---

## Syntax

```
IppStatus ippiResizeYUV420LanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeYUV420Spec* pSpec, Ipp8u* pInitBuf);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size in pixels of the source image.
<i>dstSize</i>	Size in pixels of the destination image.
<i>numLobes</i>	Parameter for Lanczos filters. Possible values are 2 or 3.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>pInitBuf</i>	Pointer to the temporal buffer for the cubic filter initialization.

## Description

This function initializes the `IppiResizeYUV420Spec` structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function depending on the value of the *numLobes* parameter.

To calculate the size of the spec structure object, call the `ippiResizeYUV420GetSize` function.

The function `ippiResizeYUV420LanczosInit` requires the external buffer *pInitBuf*. Prior to using this function, you need to call `ippiResizeYUV420GetSize` for the corresponding flavors to compute the size of the buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsResizeNoOperation</code>	Indicates an error if width or height of the image is equal to zero.
<code>ippStsSizeWrn</code>	Indicates a warning if width or height of the image is odd.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>if width or height of the source or destination image is equal to 1,</li> <li>if width or height of the source or destination image is negative,</li> <li>if the source image size is less than the Lanczos interpolation filter size: 8x8 for 2-lobed Lanczos function, or 12x12 for 3-lobed Lanczos function.</li> </ul>
<code>ippStsNotSupportedModeErr</code>	Indicates an error if the requested mode is not supported.

## ResizeYUV420SuperInit

*Initializes the spec structure for the NV12 resize transform by interpolation with the super sampling algorithm.*

## Syntax

```
IppStatus ippiResizeYUV420SuperInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV420Spec* pSpec);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size in pixels of the source image.
<i>dstSize</i>	Size in pixels of the destination image.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.

## Description

This function initializes the `IppiResizeYUV420Spec` structure for the resize algorithm using interpolation with the super sampling algorithm.

To calculate the size of the spec structure object, call the `ippiResizeYUV420GetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsResizeNoOperation</code>	Indicates an error if width or height of the image is equal to zero.
<code>ippStsSizeWrn</code>	Indicates a warning if width or height of the image is odd.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• if width or height of the image is equal to 1,</li> <li>• if one of the specified dimensions of the source image is less than the corresponding dimension of the destination image,</li> <li>• if width or height of the source or destination image is negative.</li> </ul>

## ResizeYUV420GetBorderSize

*Computes the size of possible borders for the NV12 resize transform.*

---

## Syntax

```
IppStatusippiResizeYUV420GetBorderSize(const IppiResizeYUV420Spec* pSpec,  
IppiBorderSize* borderSize);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
--------------	--

*borderSize* Size in pixels of necessary borders.

## Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries for Luma and Chroma planes. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetBorderSize` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` or `ippiResizeYUV420SuperInit`.

---

### NOTE

The returned border size is in Luma/Chroma plane pixels. This means that the chosen resize algorithm uses the returned outer size of the source image ROI for each plane.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.

## ResizeYUV420GetSrcOffset

*Computes the offset of the source image for the NV12 resize transform by tile processing.*

---

## Syntax

```
IppStatus ippiResizeYUV420GetSrcOffset(const IppiResizeYUV420Spec* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>srcOffset</i>	Offset of the source image.

## Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcOffset` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` and `ippiResizeYUV420SuperInit`.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsOutOfRangeErr	Indicates an error if the destination image offset point is outside the destination image origin.
ippStsMisalignedOffsetErr	Indicates an error if one of the fields of the <code>dstOffset</code> parameter is odd.

## ResizeYUV420GetBufferSize

Computes the size of the external buffer for the NV12 resize transform.

### Syntax

```
IppStatusippiResizeYUV420GetBufferSize(const IppiResizeYUV420Spec* pSpec, Ippisize dstSize, Ipp32s* pBufSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstSize</i>	Size in pixels of the destination image.
<i>pBufSize</i>	Pointer to the size in bytes of the external buffer.

### Description

This function computes the size of the external buffer for the NV12 resize transform. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetBufferSize` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` and `ippiResizeYUV420SuperInit`.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if <i>pBufferSize</i> pointer is <code>NULL</code> .
ippStsNoOperation	Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn	Indicates a warning in the following cases:

- if width or height of the image is odd,
- if the destination image size is more than the destination image origin size.

ippStsSizeErr

Indicates an error in the following cases:

- if width or height of the image is equal to 1,
- if width or height of the destination image is negative,
- if the calculated buffer size exceeds maximum 32 bit signed integer positive value (the processed image size is too large).

## ResizeYUV420Lanczos

*Changes the size of the NV12 image by interpolation with the Lanczos filter.*

### Syntax

```
IppStatusippiResizeYUV420Lanczos_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const
Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s
dstUVStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp8u* pBorderValue,
const IppiResizeYUV420Spec* pSpec, Ipp8u* pBuffer);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrcY</i>	Pointer to the source image Y plane.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the source image Y plane.
<i>pSrcUV</i>	Pointer to the source image UV plane.
<i>srcUVStep</i>	Distance in bytes between starts of consecutive lines in the source image UV plane.
<i>pDstY</i>	Pointer to the destination image Y plane.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the destination image Y plane.
<i>pDstUV</i>	Pointer to the destination image UV plane.
<i>dstUVStep</i>	Distance in bytes between starts of consecutive lines in the destination image UV plane.
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>dstSize</i>	Size of the destination image in pixels.
<i>border</i>	Type of the border.

<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the `ippiResizeYUV420GetSrcOffset` function. Parameters *pSrcY*, *pSrcUV* and *pDstY*, *pDstUV* must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (*pSrcY*) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (*pSrcY*) of unsigned chars containing interleaved U and V samples. Supported values for *border* are `ippBorderRepl` and `ippBorderInMem`.

Applied interpolation algorithm uses edge pixels of the source image that are out of the image origin. The function `ippiResizeYUV420Lanczos` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the function `ippiResizeYUV420GetBorderSize` for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Prior to using the `ippiResizeLinear` function, initialize the `IppiResizeYUV420Spec` structure by calling the `ippiResizeYUV420LanczosInit` and compute the size of the external buffer *pBuffer* by calling the `ippiResizeYUV420GetBufferSize` for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error if the <i>border</i> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if one of the fields of the <i>dstOffset</i> parameter is odd.
<code>ippStsSizeWrn</code>	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is odd,</li> <li>• if the destination image size is more than the destination image origin.</li> </ul>

---

ippStsSizeErr	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
ippStsOutOfRangeErr	Indicates an error if the destination image offset point is outside the destination image origin.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.

## ResizeYUV420Super

*Changes the size of the NV12 image by the super sampling interpolation method.*

---

### Syntax

```
IppStatusippiResizeYUV420Super_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const
Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s
dstUVStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeYUV420Spec* pSpec,
Ipp8u* pBuffer);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrcY</i>	Pointer to the source image Y plane.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the source image Y plane.
<i>pSrcUV</i>	Pointer to the source image UV plane.
<i>srcUVStep</i>	Distance in bytes between starts of consecutive lines in the source image UV plane.
<i>pDstY</i>	Pointer to the destination image Y plane.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the destination image Y plane.
<i>pDstUV</i>	Pointer to the destination image UV plane.
<i>dstUVStep</i>	Distance in bytes between starts of consecutive lines in the destination image UV plane.
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>dstSize</i>	Size of the destination image in pixels.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using interpolation with the super sampling algorithm. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the [ippiResizeYUV420GetSrcOffset](#) function. Parameters *pSrcY*, *pSrcUV* and *pDstY*, *pDstUV* must point to the processed source and destination image ROI origins respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (*pSrcY*) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (*pSrcY*) of unsigned chars containing interleaved U and V samples.

Prior to using the [ippiResizeLinear](#) function, initialize the [IppiResizeYUV420Spec](#) structure by calling the [ippiResizeYUV420LanczosInit](#) and compute the size of the external buffer *pBuffer* by calling the [ippiResizeYUV420GetBufferSize](#) for the corresponding flavor.

---

### NOTE

This function provides optimized code paths for the following scaling factors along the *x* and *y* axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error if the <i>border</i> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsSizeWrn</code>	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is odd,</li> <li>• if the destination image size is more than the destination image origin.</li> </ul>
<code>ippStsSizeErr</code>	Indicates an error if width or height of the destination image is equal to 1; or if width or height of the source or destination image is negative.
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if one of the fields of the <i>dstOffset</i> parameter is odd.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.

## ResizeYUV422GetSize

*Computes sizes of the spec structure and the external buffer for YUY2 resize transform initialization.*

### Syntax

```
IppStatusippiResizeYUV422GetSize(IppiSize srcSize, IppiSize dstSize,
IppInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s* pInitBufSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>srcSize</i>	Size of the source image in pixels.
<i>dstSize</i>	Size of the destination image in pixels.
<i>interpolation</i>	Interpolation method. Supported values are <code>ippNearest</code> and <code>ippLinear</code> .
<i>antialiasing</i>	Antialiasing method.
<i>pSpecSize</i>	Pointer to the size in bytes of the spec structure.
<i>pInitBufSize</i>	Pointer to the size in bytes of the temporal buffer.

### Description

This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: [ResizeYUV422NearestInit](#) and [ResizeYUV422LinearInit](#).

The filter sizes of the Nearest Neighbor and Linear interpolation algorithms are 2x1 and 4x2 respectively.

---

#### NOTE

Antialiasing is currently not supported. The value for the *antialiasing* parameter must be equal to zero.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the image is equal to zero.
<code>ippStsSizeErr</code>	Indicates an error in the following cases:

- if the source image size is less than the filter size for the chosen interpolation method,
- if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of one of the processed images is too large).

`ippStsSizeWrn`

Indicates a warning if width of the image is odd.

`ippStsInterpolationErr`

Indicates an error if `interpolation` has an illegal value.

`ippStsNoAntialiasing`

Indicates a warning if the specified interpolation method does not support antialiasing.

`ippStsNotSupportedModeErr`

Indicates an error if the requested mode is currently not supported.

## ResizeYUV422GetBorderSize

*Computes the size of possible borders for the YUY2 resize transform.*

---

### Syntax

```
IppStatusippiResizeYUV422GetBorderSize(const IppiResizeYUV422Spec* pSpec,  
IppiBorderSize* borderSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

`pSpec` Pointer to the spec structure for the resize filter.

`borderSize` Size in pixels of necessary borders.

### Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetBorderSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV422NearestInit`, and `ippiResizeYUV422LinearInit`.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error if one of the specified pointers is `NULL`.

`ippStsContextMatchErr` Indicates an error if pointer to the spec structure is invalid.

## ResizeYUV422GetSrcOffset

*Computes the offset of the source image for the YUY2 resize transform by tile processing.*

### Syntax

```
IppStatusippiResizeYUV422GetSrcOffset(const IppiResizeYUV422Spec* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>srcOffset</i>	Offset of the source image.

### Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetSrcOffset` function, you need to initialize the *pSpec* parameter by calling one of the following functions:

`ippiResizeYUV422NearestInit` and `ippiResizeYUV422LinearInit`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if the <code>x</code> field of the <i>dstOffset</i> parameter is odd.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.

## ResizeYUV422GetBufSize

*Computes the size of the external buffer for the NV12 resize transform.*

### Syntax

```
IppStatusippiResizeYUV422GetBufSize(const IppiResizeYUV422Spec* pSpec, IppiSize dstSize, Ipp32s* pBufSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>dstSize</i>	Size in pixels of the destination image.
<i>pBufSize</i>	Pointer to the size in bytes of the external buffer.

## Description

This function computes the size of the external buffer for the YUY2 resize transform. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetBufferSize` function, you need to initialize the *pSpec* parameter by calling one of the following functions:  
`ippiResizeYUV422NearestInit` and `ippiResizeYUV422LinearInit`.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsNoOperation	Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is odd,</li> <li>• if the destination image size is more than the destination image origin size.</li> </ul>
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is equal to 1,</li> <li>• if width or height of the destination image is negative,</li> <li>• if the calculated buffer size exceeds maximum 32 bit signed integer positive value (the processed image size is too large).</li> </ul>

## ResizeYUV422GetSrcRoi

Computes the ROI of the source image for YUV422 resizing by tile processing.

## Syntax

```
IppStatusippiResizeYUV422GetSrcRoi(const IppiResizeYUV422Spec* pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSpec</code>	Pointer to the spec structure for the resize filter.
<code>dstRoiOffset</code>	Offset of the tiled destination image ROI.
<code>dstRoiSize</code>	Size of the tiled destination image ROI.
<code>srcRoiOffset</code>	Offset of the source image ROI.
<code>srcRoiSize</code>	Pointer to the size of the source image ROI.

## Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetSrcRoi` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV422NearestInit` or `ippiResizeYUV422LinearInit`.

### NOTE

If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the specification structure is invalid.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if x-value of the parameter <code>dstRoiOffset</code> is odd.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• If the height of the destination ROI is zero or negative.</li> <li>• If the width of the destination ROI is less than 2.</li> </ul>
<code>ippStsSizeWrn</code>	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• If the width of the destination ROI is odd.</li> <li>• If the destination ROI exceeds the destination image origin.</li> </ul>

## ResizeYUV422NearestInit

Initializes the spec structure for the YUY2 resize transform by the nearest neighbor interpolation method.

## Syntax

```
IppStatusippiResizeYUV422NearestInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size in pixels of the source image.
<i>dstSize</i>	Size in pixels of the destination image.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.

## Description

This function initializes the `IppiResizeYUV422Spec` structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the spec structure object, call the `ippiResizeYUV422GetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the image is equal to zero.
<code>ippStsSizeWrn</code>	Indicates a warning if width of the image is odd.
<code>ippStsSizeErr</code>	Indicates an error if width of the image is equal to 1, or if width or height of the source or destination image is negative.

## ResizeYUV422LinearInit

*Initializes the spec structure for the YUY2 resize transform by the linear interpolation method.*

---

## Syntax

```
IppStatusippiResizeYUV422LinearInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size in pixels of the source image.
<i>dstSize</i>	Size in pixels of the destination image.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.

## Description

This function initializes the `IppiResizeYUV422Spec` structure for the resize algorithm with the linear interpolation method. To calculate the size of the spec structure object, call the `ippiResizeYUV422GetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the image is equal to zero.
<code>ippStsSizeWrn</code>	Indicates a warning if width of the image is odd.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is equal to 1,</li> <li>• if width or height of the source or destination image is negative,</li> <li>• if the source image size is less than the linear filter size 4x2.</li> </ul>

## ResizeYUV422Nearest

*Changes an YUY2 image size by the nearest neighbor interpolation method.*

## Syntax

```
IppStatusippiResizeYUV422Nearest_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, IppPoint dstOffset, IppiSize dstSize, const
IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>dstSize</i>	Size of the destination image in pixels.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the [ippiResizeYUV422GetSrcOffset](#) function. Parameters *pSrc* and *pDst* must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the YUY2 pixel format ( Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,...).

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Prior to using the [ippiResizeYUV422Nearest](#) function, initialize the `IppiResizeYUV422Spec` structure by calling the [ippiResizeYUV422NearestInit](#) and compute the size of the external buffer *pBuffer* by calling the [ippiResizeYUV422GetBufSize](#) for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsSizeWrn</code>	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• if width of the image is odd,</li> <li>• if the destination image size is more than the destination image origin.</li> </ul>
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if the <code>x</code> field of the <i>dstOffset</i> parameter is odd.
<code>ippStsSizeErr</code>	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.

## ResizeYUV422Linear

*Changes an YUY2 image size by the linear interpolation method.*

---

## Syntax

```
IppStatusippiResizeYUV422Linear_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst,
Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const
Ipp8u* pBorderValue, const IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstOffset</i>	Offset of the tiled destination image with respect to the destination image origin.
<i>dstSize</i>	Size of the destination image in pixels.
<i>border</i>	Type of the border.
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	Pointer to the spec structure for the resize filter.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function changes an image size using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the [ippiResizeYUV422GetSrcOffset](#) function. The source and destination images are in the YUY2 pixel format (Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,...).

Supported values for *border* are `ippBorderRepl` and `ippBorderInMem`.

The interpolation algorithm applied uses edge pixels of the source image that are out of the ROI. The function `ippiResizeYUV422Linear` uses the weighted values of these outer pixels in calculation. To obtain the size of the out of the ROI source image, call the function [ippiResizeYUV422GetBorderSize](#).

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the ROI. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the ROI image space.

Prior to using the `ippiResizeYUV422Linear` function, initialize the `IppiResizeYUV422Spec` structure by calling the `ippiResizeYUV422LinearInit` and compute the size of the external buffer `pBuffer` by calling the `ippiResizeYUV422GetBufSize` for the corresponding flavor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning if width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error if the <code>border</code> value is illegal.
<code>ippStsContextMatchErr</code>	Indicates an error if pointer to the spec structure is invalid.
<code>ippStsSizeWrn</code>	Indicates a warning in the following cases: <ul style="list-style-type: none"> <li>• if width of the destination image is odd,</li> <li>• if the destination image size is more than the destination image origin.</li> </ul>
<code>ippStsMisalignedOffsetErr</code>	Indicates an error if the <code>x</code> field of the <code>dstOffset</code> parameter is odd.
<code>ippStsSizeErr</code>	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset point is outside the destination image origin.

## Warp Functions with Prior Initialization

This section describes the Intel® IPP warping functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

### Using Intel® IPP Warp Affine Functions with Prior Initialization

You can use one of the following approaches to image warping:

- [Warping the whole image](#)
- [Warping a tiled image with one prior initialization](#)

Interpolation algorithms of the Nearest Neighbor, Linear, and Cubic types can use edge pixels of the source image that are out of the image origin. When calling the `ippiWarpAffine<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels
- Constant border: values of all border pixels are set to a constant
- Transparent borders: destination pixels that have inverse transformed location out of the source image are not processed
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory
- Mixed borders: combination of transparent borders and borders in memory is applied

### Warping the Whole Image

You can follow the approach described below to apply affine transformation when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

To transform the whole image:

1. Call the `WarpAffineGetSize` function with the appropriate interpolation type. This function uses source and destination image sizes to calculate how much memory must be allocated for the `IppWarpSpec` structure and work buffer.
2. Initialize the `IppWarpSpec` structure by calling the `ippiWarp<Filter>Init`, where `<Filter>` can take one of the following values: Nearest, Linear, and Cubic. These prerequisite steps enable calling the warp functions multiple times without recalculations.
3. Call the `WarpGetBufferSize` function for the initialized `IppWarpSpec` structure. This function uses the destination image size to calculate how much memory must be allocated for the warp work buffer.
4. Call `ippiWarpAffine<Filter>` with the appropriate image type.
5. Specify the algorithm for borders processing by setting the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:
  - If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
  - If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.
  - If the border type is equal to `ippBorderTransp`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are replicated from the edge pixels, if they are required by interpolation algorithm.
  - If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are obtained from the out of the source image origin space, if they are required by interpolation algorithm.
  - The mixed border types can be obtained by the bitwise operation OR between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values.

Figure *Whole Image Warping* shows a simple image affine transformation example. Transformation coefficients are  $\{\{1.0, 0.5, 0.0\}, \{0.5, 1.0, 0.0\}\}$ , border type is `ippBorderConst`, `pBorderValue` is a white color pixel. The size of the destination image is 1.2x of the source image size.

### Whole Image Warping



## Example

The code example below demonstrates affine transformation of the whole image with the linear interpolation method:

```
IppStatus warpAffineExample_8u_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize,
    Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    Ipp64f pBorderValue[numChannels];

    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;

    /* Spec and init buffer sizes */
    status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction,
borderType, &specSize, &initSize);

    if (status != ippStsNoErr) return status;

    /* Memory allocation */
    pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);

    if (pSpec == NULL)
    {
        return ippStsNoMemErr;
    }

    /* Filter initialization */
    status = ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction, numChannels,
borderType, pBorderValue, 0, pSpec);

    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    /* work buffer size */
    status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    pBuffer = ippsMalloc_8u(bufSize);
    if (pBuffer == NULL)
    {
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }
```

```
/* Resize processing */
status =ippiWarpAffineLinear_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset, dstSize,
pSpec, pBuffer);

ippsFree(pSpec);
ippsFree(pBuffer);

return status;
}
```

## Warping a Tiled Image with One Prior Initialization

You can follow the approach described below to apply affine transformation when the source image is fully accessible in memory and destination image is not fully accessible in memory, or to improve performance of warping by external threading.

The main difference between this approach and whole image warping is that the image is split into sections called tiles. Each call of the `WarpAffine<Filter>` function works with the destination image origin region of interest (ROI) that is defined by `dstRoiOffset` and `dstRoiSize` parameters. The destination ROI must be fully accessible in memory.

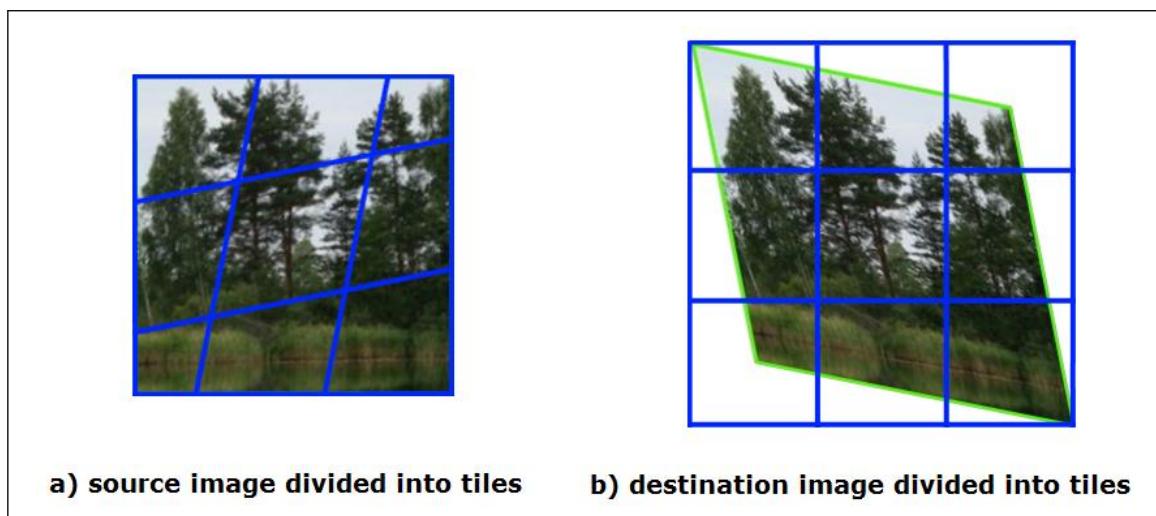
To resize an image with the tiled approach:

1. Call the `WarpAffineGetSize` function with the appropriate interpolation type. This function uses the size of the source and destination images and transformation parameters to calculate how much memory must be allocated for the `IppWarpSpec` structure and initialization work buffer.
2. Initialize the `IppWarpSpec` structure by calling `ippiWarpAffine<Filter>Init`, where `<Filter>` can take one of the following values: Nearest, Linear, and Cubic.
3. Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is sufficient in most cases.
4. Call the `WarpGetBufferSize` function to obtain the size of the work buffer required for each tile processing. The `dstRoiSize` parameter must be equal to the tile size.
5. Call `ippiWarpAffine<Filter>` for each tile (ROI). The `dstRoiOffset` parameter must specify the image ROI offset with respect to the destination image origin. The `dstRoiSize` parameter must be equal to the ROI size. The `pDst` parameter must point to the beginning of the destination ROI in memory. The source and destination ROIs must be fully accessible in memory.

You can process tiles in any order. When using multiple threads you can process all tiles simultaneously.

Figure *Tiled Image Warping* shows the affine transformation of the image divided into tiles. Transformation coefficients are  $\{\{1.0, 0.5, 0.0\}, \{0.5, 1.0, 0.0\}\}$ , applied border type is `ippBorderConst`, `pBorderValue` is a white color pixel. The size of the destination image is 1.2x of the source image size.

### Tiled Image Warping



### Example

The code example below demonstrates a multithreading affine transformation using OpenMP\* with parallelization only in y direction:

```
IppStatus tileWarpAffineExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize, Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    int numThreads, slice, tail;
    int bufSize1, bufSize2;
    IppiSize dstTileSize, dstLastTileSize; IppStatus pStatus[MAX_NUM_THREADS];
    Ipp64f pBorderValue[numChannels];

    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;

    /* Spec and init buffer sizes */
    status =ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction,
borderType, &specSize, &initSize);

    if (status != ippStsNoErr) return status;

    /* Memory allocation */
    pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);

    if (pSpec == NULL)
    {
        return ippStsNoMemErr;
```

```

/* Filter initialization */
status =ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction, numChannels,
borderType, pBorderValue, 0, pSpec);

if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* General transform function */
/* Parallelized only by Y-direction here */
#pragma omp parallel num_threads(MAX_NUM_THREADS)
{
#pragma omp master
{
    numThreads = omp_get_num_threads();

    slice = dstSize.height / numThreads; tail = dstSize.height % numThreads;

    dstTileSize.width = dstLastTileSize.width = dstSize.width;
    dstTileSize.height = slice;
    dstLastTileSize.height = slice + tail;

    ippiWarpGetBufferSize(pSpec, dstTileSize, &bufSize1);
    ippiWarpGetBufferSize(pSpec, dstLastTileSize, &bufSize2);

    pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
}

#pragma omp barrier
{
    if (pBuffer)
    {
        Ipp32u i;
        Ipp8u *pDstT; Ipp8u *pOneBuf;
        IppiPoint srcOffset = {0, 0};
        IppiPoint dstOffset = {0, 0};
        IppiSize srcSizeT = srcSize; IppiSize dstSizeT = dstTileSize;

        i = omp_get_thread_num();

        dstSizeT.height = slice; dstOffset.y += i * slice;

        if (i == numThreads - 1) dstSizeT = dstLastTileSize;

        pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);

        pOneBuf = pBuffer + i * bufSize1;

        pStatus[i] = ippiWarpAffineLinear_8u_C3R (pSrc, srcStep, pDstT, dstStep,
dstOffset, dstSizeT, pSpec, pOneBuf);
    }
}
}

```

```

    ippsFree(pSpec);

    if (pBuffer == NULL) returnippStsNoMemErr;

    ippsFree(pBuffer);

    for (Ipp32u i = 0; i < numThreads; ++i)
    {
        /* Return bad status */
        if(pStatus[i] !=ippStsNoErr) return pStatus[i];
    }

    return status;
}

```

## See Also

[User-defined Border Types](#)

[WarpAffineGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

## Edge Smoothing

The *Smooth Edge* feature is an artificial method to reduce aliasing artifacts at the transformed source image edges. Aliasing artifacts may appear because the transformation algorithms skip a destination pixel if its source origin is out of the source image ROI. Thus, borders of the transformed source image can look stepped:



a) SMOOTH\_EDGE option turned off



b) SMOOTH\_EDGE option turned on

If the *smoothEdge* flag is set, destination pixels that are closest to the transformed source image edges are mixed with sampled source pixels by the following formula:

$$dstRes = srcSampled * (1-a) + dstExist * a$$

where

- *srcSampled* is the intensity of the source pixel after transformation.
- *dstExist* is the intensity of the destination pixel before transformation.
- *a* is the weight of the outer pixel; set by the function.
- *dstRes* is the intensity of the resulting destination pixel.

The edge smoothing method is not universal: in some cases it can improve the image, but in other cases it can be inefficient. For example, edge smoothing does not increase the quality of images with high contrast borders, and it is not recommended to apply edge smoothing to such images.

#### **NOTE**

Edge smoothing is a post-processing operation: it is performed after transformation. When warping a tiled image, artifacts may appear on tile borders. In this case, edges are not smoothed.

## **GetAffineQuad**

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.*

### **Syntax**

```
IppStatusippiGetAffineQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[2][3]);
IppStatusippiGetAffineQuad_L(IppiRectL srcRoi, double quad[4][2], const double coeffs[2][3]);
```

### **Include Files**

ippi.h

Flavors with the \_L suffix: ippi\_l.h

### **Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### **Parameters**

<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
<i>coeffs</i>	The given affine transform coefficients.

### **Description**

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [WarpAffineNearest](#), [WarpAffineLinear](#), and [WarpAffineCubic](#) functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the affine transform function using the given coefficients *coeffs*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad*[0] corresponds to the transformed top-left corner of the source ROI,

*quad*[1] corresponds to the transformed top-right corner of the source ROI,

*quad*[2] corresponds to the transformed bottom-right corner of the source ROI,

*quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsCoeffErr	Indicates an error condition if $c_{00} \cdot c_{11} - c_{01} \cdot c_{10} = 0$ .
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

## GetAffineBound

*Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.*

---

### Syntax

```
IppStatus ippiGetAffineBound (IppiRect srcRoi, double bound[2][2], const double coeffs[2][3]);
```

### Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given affine transform coefficients.

### Description

This function is used as a support function for [WarpAffineNearest](#), [WarpAffineLinear](#), and [WarpAffineCubic](#) functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the affine transform function using coefficients *coeffs*.

*bound[0]* specifies x, y coordinates of the top-left corner, *bound[1]* specifies x, y coordinates of the bottom-right corner.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsCoeffErr	Indicates an error condition if $c_{00} \cdot c_{11} - c_{01} \cdot c_{10} = 0$ .
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

## GetAffineSrcRoi

*Computes ROI of an image for affine transform.*

---

## Syntax

```
IppStatusippiGetAffineSrcRoi (IppiSize srcSize, const double coeffs[2][3],  
IppiWarpDirection direction, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiRect  
*srcRoi);
```

## Include Files

ippi.h

Flavors with the `_L` suffix: ippi\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>coeffs</code>	Coefficients for affine transform.
<code>direction</code>	Transformation direction. Supported values:  ippWarpForward      Forward transformation. ippWarpBackward     Backward transformation.
<code>dstRoiSize</code>	Size of the ROI of destination image.
<code>dstRoiOffset</code>	Offset of the destination image ROI.
<code>srcRoi</code>	Pointer to the computed region of interest in the source image.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for the [ippiWarpAffineLinear](#), [WarpAffineNearest](#), and [WarpAffineCubic](#) functions. It computes ROI of the source image to perform affine transformation for a given destination ROI. To process the given destination ROI, the computed source ROI with borders must be accessible in memory. If the source ROI outside pixels are out of the source image origin, the border pixels are processed according to the `border` flag that is passed to the [ippiWarpAffineLinear](#), [WarpAffineNearest](#), and [WarpAffineCubic](#) functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or warning.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <code>srcRoi</code> is less than or equal to 1.
<code>ippStsOutOfRangeErr</code>	Indicates an error if the destination image offset has a field with a negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of the source or destination image is less than, or equal to zero.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed because the transformed source image has no intersection with the destination ROI.

ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsWarpDirectionErr	Indicates an error when the direction value is illegal.
ippStsCoefErr	Indicates an error condition, if affine transformation is singular.

## GetAffineTransform

Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

---

### Syntax

```
IppStatusippiGetAffineTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[2][3]);
```

### Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_1.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
<code>coeffs</code>	Output array. Contains the target affine transform coefficients.

### Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for the `ippiWarpAffineLinear`, `WarpAffineNearest`, and `WarpAffineCubic` functions. It computes the coefficients `coeffs` of the affine transform that must be used by the warping function to map the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`. The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad[0]` corresponds to the transformed top-left corner of the source ROI, `quad[1]` corresponds to the transformed top-right corner of the source ROI, `quad[2]` corresponds to the transformed bottom-right corner of the source ROI, `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in `quad`, the function returns the warning message and continues operation with the computed values.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or warning.
--------------------------	--

ippStsRectErr	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.
ippStsCoeffErr	Indicates an error condition if $c_{00} \cdot c_{11} - c_{01} \cdot c_{10} = 0$ .
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
ippStsAffineQuadChanged	Indicates a warning that coordinates of the 4th vertex of the specified quadrangle <i>quad</i> are not correct.

## GetRotateTransform

Computes the affine coefficients for the rotation transform.

### Syntax

```
IppStatusippiGetRotateTransform(double angle, double xShift, double yShift, double coeffs[2][3]);
```

### Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>angle</i>	Angle of rotation, in degrees. The source image is rotated counterclockwise around the origin (0, 0).
<i>xShift, yShift</i>	Shift along horizontal ( <i>x</i> ) or vertical ( <i>y</i> ) axis that is performed after rotation.
<i>coeffs</i>	Computed affine transform coefficients for the given rotation parameters.

### Description

This function computes the coefficients for the affine transform that rotates an image by the specified angle around the origin (0, 0) and shifts the image after rotation. The result is stored in the *coeffs* parameter.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error when one of the <i>coeffs</i> values is NULL.
ippStsOutOfRangeErr	Indicates an error when <i>angle</i> is not-a-number (NaN) or infinity.

### Example

To better understand usage of the ippiGetRotateTransform function, refer to the GetRotateTransform.c example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## GetRotateShift

*Computes shift values for rotation of an image around the specified center.*

### Syntax

```
IppStatusippiGetRotateShift (double xCenter, double yCenter, double angle, double*  
xShift, double* yShift);
```

### Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>xCenter, yCenter</i>	Coordinates of the required center of rotation.
<i>angle</i>	The angle in degrees to rotate the image clockwise around the point with coordinates ( <i>xCenter, yCenter</i> ).
<i>xShift, yShift</i>	Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to <code>ippiRotate</code> function to bring about the desired rotation around ( <i>xCenter, yCenter</i> ).

### Description

Use this function if you need to rotate an image about an arbitrary center (*xCenter, yCenter*) rather than the origin (0,0). The function helps compute shift values *xShift, yShift* that should be passed to the warping function for the rotation around (*xCenter, yCenter*) to take place.

**Example** shows how to use the function `ippiGetRotateShift`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>xShift</i> or <i>yShift</i> pointer is NULL.

## WarpAffineGetSize

*Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.*

### Syntax

```
IppStatusippiWarpAffineGetSize(IppiSize srcSize, IppiSize dstSize, IppDataType  
dataType, const double coeffs[2][3], IppiInterpolationType interpolation,  
IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize, int*  
pInitBufSize);
```

## Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>dstSize</code>	Size of the destination image, in pixels.
<code>dataType</code>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<code>coeffs</code>	Coefficients for the affine transform.
<code>interpolation</code>	Interpolation method. Supported values: <code>ippNearest</code> , <code>ippLinear</code> , and <code>ippCubic</code> .
<code>direction</code>	Transformation direction. Supported values: <code>ippWarpForward</code> Forward transformation <code>ippWarpBackward</code> Backward transformation
<code>borderType</code>	Type of border. Possible values are:  <code>ippBorderConst</code> Values of all border pixels are set to constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderTransp</code> Outer pixels are not processed. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<code>pSpecSize</code>	Pointer to the size, in bytes, of the specification structure.
<code>pInitBufSize</code>	Pointer to the size, in bytes, of the temporary buffer.

## Description

This function computes the size of the specification structure and the external work buffer for the following functions, depending on the *interpolation* parameter: [WarpAffineNearestInit](#), [WarpAffineLinearInit](#), or [WarpAffineCubicInit](#).

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error If the width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsWarpDirectionErr	Indicates an error when <i>direction</i> has an illegal value.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsExceededSizeErr	<p>Indicates an error in the following cases:</p> <ul style="list-style-type: none"> <li>• If one of the calculated sizes exceeds maximum of the <i>pSpecSize</i> variable data type positive value (the size of one of the processed images is too large).</li> <li>• If width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFFF) .</li> </ul>

## See Also

[WarpAffineNearestInit](#) Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

[WarpAffineLinearInit](#) Initializes the specification structure for image affine warping with the linear interpolation method.

[WarpAffineCubicInit](#) Initializes the specification structure for image affine warping with the cubic interpolation method.

## WarpQuadGetSize

*Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.*

## Syntax

```
IppStatusippiWarpQuadGetSize(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, IppiInterpolationType interpolation, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcQuad</i>	Quadrangle in the source image.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dstQuad</i>	Quadrangle in the destination image.
<i>transform</i>	Type of the warp transform. Supported values: ippWarpAffine      Affine warping ippWarpPerspective      Perspective warping
<i>dataType</i>	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
<i>interpolation</i>	Interpolation method. Supported values: ippNearest, ippLinear, and ippCubic.
<i>borderType</i>	Type of border. Supported values: ippBorderTransp      Outer pixels are not processed. ippBorderInMem      Border is obtained from the source image pixels in memory.
<i>pSpecSize</i>	Pointer to the size, in bytes, of the specification structure.
<i>pInitBufSize</i>	Pointer to the size, in bytes, of the temporary buffer.

## Description

This function computes the size of the specification structure and the temporary buffer for the following functions, depending on the *interpolation* parameter: [ippiWarpQuadNearestInit](#), [ippiWarpQuadLinearInit](#), or [ippiWarpQuadCubicInit](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• If the width or height of the source or destination image is less than, or equal to one.</li> <li>• If one of the calculated sizes exceeds the maximum positive 32-bit signed integer value. The size of the one of the processed images is too large.</li> </ul>
<i>ippStsDataTypeErr</i>	Indicates an error when <i>dataType</i> has an illegal value.

ippStsWarpTransformErr	Indicates an error when <i>transform</i> has an illegal value.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.
ippStsQuadErr	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of <i>dstQuad</i> are changed by the function, if <i>transform</i> is set to <i>ippWarpAffine</i> .
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## See Also

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

[WarpQuadLinearInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

[WarpQuadCubicInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

## WarpGetBufferSize

*Computes the size of the work buffer for the warp transform.*

---

## Syntax

```
IppStatusippiWarpGetBufferSize(const IppiWarpSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);
```

## Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSpec</i>	Pointer to the specification structure for the warp operation.
<i>dstRoiSize</i>	Size of the processed destination image ROI, in pixels.
<i>pBufSize</i>	Pointer to the size of the external buffer, in bytes.

## Description

This function computes the size of the external buffer for the warp transform. The specification structure pointed by *pSpec* defines the warp algorithm parameters.

Before using this function, you need to initialize the specification structure using one of the following functions: [WarpAffineNearestInit](#), [WarpAffineLinearInit](#), or [WarpAffineCubicInit](#).

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is <code>NULL</code> .
ippStsContextMatchErr	Indicates an error when the specification structure is invalid.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>If width or height of the destination image is negative, or equal to zero.</li> <li>If the calculated buffer size exceeds the maximum positive <code>pBufSize</code> data type. The size of the processed image ROI is too large.</li> </ul>
ippStsSizeWrn	Indicates a warning when the size of the destination image is more than the size of the destination image origin.

## See Also

[WarpAffineNearestInit](#) Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

[WarpAffineLinearInit](#) Initializes the specification structure for image affine warping with the linear interpolation method.

[WarpAffineCubicInit](#) Initializes the specification structure for image affine warping with the cubic interpolation method.

## WarpAffineNearestInit

*Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.*

## Syntax

```
IppStatusippiWarpAffineNearestInit(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels,
IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec*
pSpec);
```

## Include Files

ippi.h

Flavors with the `_L` suffix: ippi\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>dstSize</code>	Size of the destination image, in pixels.
<code>dataType</code>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .

<i>coeffs</i>	Coefficients for the affine transform.
<i>direction</i>	Transformation direction. Supported values: ippWarpForward Forward transformation ippWarpBackward Backward transformation
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>borderType</i>	Type of border. Supported values: ippBorderConst Values of all border pixels are set to a constant. ippBorderRepl Border is replicated from the edge pixels. ippBorderTransp Outer pixels are not processed. ippBorderInMem Border is obtained from the source image pixels in memory. Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>smoothEdge</i>	Flag for edge smoothing. Supported values: <ul style="list-style-type: none"><li>• 0 - transformation without edge smoothing.</li><li>• 1 - transformation with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.</li></ul>
<i>pSpec</i>	Pointer to the specification structure.

## Description

This function initializes the `IppiWarpSpec` structure for the `ippiWarpAffineNearest` function that performs warp affine transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the `WarpAffineGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when: <ul style="list-style-type: none"><li>• one of the specified pointers is NULL, excepting <code>pBorderValue</code></li><li>• <code>pBorderValue</code> is NULL when border type is set to <code>ippBorderConst</code></li></ul>

ippStsNoOperation	Indicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsWarpDirectionErr	Indicates an error when <i>direction</i> has an illegal value.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image. The edge smoothing feature is not supported for the <i>ippBorderRepl</i> and <i>ippBorderConst</i> border types.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.
ippStsExceededSizeErr	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

## See Also

[WarpAffineGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

## WarpQuadNearestInit

*Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.*

## Syntax

```
IppStatusippiWarpQuadNearestInit(IppiSize srcSize, const double srcQuad[4][2],
IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform,
IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f*
pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcQuad</i>	Quadrangle in the source image.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dstQuad</i>	Quadrangle in the destination image.

<i>transform</i>	Type of the warp transform. Supported values:	
	ippWarpAffine	Affine warping
	ippWarpPerspective	Perspective warping
<i>dataType</i>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .	
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.	
<i>borderType</i>	Type of border. Supported values:	
	ippBorderTransp	Outer pixels are not processed.
	ippBorderInMem	Border is obtained from the source image pixels in memory.
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.	
<i>smoothEdge</i>	Flag for edge smoothing. Supported values:	
	<ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing.</li> </ul>	
<i>pSpec</i>	Pointer to the specification structure.	

## Description

This function initializes the `IppiWarpSpec` structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the nearest neighbor interpolation method. To compute the size of the specification structure, use the [WarpQuadGetSize](#) function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle `dstQuad` and transform type `transform`. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in `dstQuad`, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle `srcQuad[4][2]` or `dstQuad[4][2]` is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li>• One of the specified pointers is <code>NULL</code>, excepting <code>pBorderValue</code></li> <li>• The value of <code>pBorderValue</code> is <code>NULL</code> when the border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpTransformErr</code>	Indicates an error when <code>transform</code> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.

---

ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of <i>dstQuad</i> are changed by the function, if <i>transform</i> is set to <i>ippWarpAffine</i> .
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

## Example

To better understand usage of the *ippiWarpQuadNearest*, *ippiWarpQuadLinear*, and *ippiWarpQuadCubic* functions, refer to the *WarpQuadNearestInit.c* example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

**WarpQuadGetSize** Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

## WarpAffineNearest

*Performs warp affine transformation of an image using the nearest neighbor interpolation method.*

---

## Syntax

```
IppStatusippiWarpAffineNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R 64f\_C1R

8u\_C3R 16u\_C3R 16s\_C3R 32f\_C3R 64f\_C3R

8u\_C4R 16u\_C4R 16s\_C4R 32f\_C4R 64f\_C4R

## Include Files

ippi.h

Flavors with the \_L suffix: ippi\_l.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiOffset</i>	Offset of the destination image ROI with respect to the destination image origin.
<i>dstRoiSize</i>	Size of the destination image ROI, in pixels.
<i>pSpec</i>	Pointer to the specification structure for the warp operation.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = c_{00} * x + c_{01} * y + c_{02}$$

$$y' = c_{10} * x + c_{11} * y + c_{12}$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the *coeffs* array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The `WarpAffineNearest` function operates with ROI. The transformed part of the image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter *pSrc* must point to the source image origin. The parameter *pDst* must point to the processed destination image ROI.

If you initialize the specification structure using the `WarpQuadNearestInit` function, the operations take place only inside the specified source quadrangle *srcQuad* that is set in `WarpQuadNearestInit`.

To specify the algorithm for borders processing, set the *borderType* and *pBorderValue* parameters when initializing the `IppiWarpSpec` structure. The data type of *pBorderValue* is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in *pBorderValue*.
- If the border type is equal to `ippBorderTransp` or `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed.

Before using the `WarpAffineNearest` function, you need to initialize the `IppiWarpSpec` structure using the `WarpAffineNearestInit` function and compute the size of the external buffer *pBuffer* using the `WarpGetBufferSize` function.

To compute the affine transform parameters, use the `GetAffineQuad`, `GetAffineBound`, and `GetAffineTransform` functions.

For an example on how to use this function, refer to the `WarpQuadNearestInit` function description.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is <code>NULL</code> .
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error if border type has an illegal value.
ippStsContextMatchErr	Indicates an error when context data is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the destination ROI has no intersection with the transformed source image origin.

## See Also

[ROI Processing in Geometric Transforms](#)

[WarpAffineNearestInit](#) Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetAffineBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpAffine` function.

[GetAffineQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

[GetAffineTransform](#) Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

## [WarpAffineLinearInit](#)

*Initializes the specification structure for image affine warping with the linear interpolation method.*

## Syntax

```
IppStatus ippiWarpAffineLinearInit(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels,
IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec*
pSpec);
```

## Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>dstSize</code>	Size of the destination image, in pixels.
<code>dataType</code>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<code>coeffs</code>	Coefficients for the affine transform.
<code>direction</code>	Transformation direction. Supported values:  <code>ippWarpForward</code> Forward transformation <code>ippWarpBackward</code> Backward transformation
<code>numChannels</code>	Number of channels in the image. Supported values: 1, 3, or 4.
<code>borderType</code>	Type of border. Supported values:  <code>ippBorderConst</code> Values of all border pixels are set to a constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderTransp</code> Outer pixels are not processed. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<code>pBorderValue</code>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>smoothEdge</code>	Flag for edge smoothing. Supported values: <ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.</li> </ul>
<code>pSpec</code>	Pointer to the specification structure.

## Description

This function initializes the `IppiWarpSpec` structure for the `ippiWarpAffineLinear` function that performs warp affine transformation with the linear interpolation method. To compute the size of the specification structure, use the `WarpAffineGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li>• one of the specified pointers is <code>NULL</code>, excepting <code>pBorderValue</code></li> <li>• <code>pBorderValue</code> is <code>NULL</code> when border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsNoOperation</code>	Indicates a warning if width or height of any image is zero.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <code>direction</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error when affine transformation is singular.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the <code>ippBorderRepl</code> and <code>ippBorderConst</code> border types.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.
<code>ippStsExceededSizeErr</code>	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

## See Also

`WarpAffineGetSize` Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

## WarpQuadLinearInit

*Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.*

## Syntax

```
IppStatus ippiWarpQuadLinearInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
<code>srcQuad</code>	Quadrangle in the source image.
<code>dstSize</code>	Size of the destination image, in pixels.
<code>dstQuad</code>	Quadrangle in the destination image.
<code>transform</code>	Type of the warp transform. Supported values:  ippWarpAffine      Affine warping ippWarpPerspective      Perspective warping
<code>dataType</code>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<code>numChannels</code>	Number of channels in the image. Supported values: 1, 3, or 4.
<code>borderType</code>	Type of border. Supported values:  ippBorderTransp      Outer pixels are not processed. ippBorderInMem      Border is obtained from the source image pixels in memory.
<code>pBorderValue</code>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>smoothEdge</code>	Flag for edge smoothing. Supported values:  <ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing.</li> </ul>
<code>pSpec</code>	Pointer to the specification structure.

## Description

This function initializes the `IppiWarpSpec` structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the linear interpolation method. To compute the size of the specification structure, use the [WarpQuadGetSize](#) function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle `dstQuad` and transform type `transform`. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in `dstQuad`, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle `srcQuad[4][2]` or `dstQuad[4][2]` is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

For an example on how to use this function, refer to the example provided with the [WarpQuadNearestInit](#) function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when:
	<ul style="list-style-type: none"> <li>• One of the specified pointers is <code>NULL</code>, excepting <code>pBorderValue</code></li> <li>• The value of <code>pBorderValue</code> is <code>NULL</code> when the border type is set to <code>ippBorderConst</code></li> </ul>
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when <code>dataType</code> has an illegal value.
ippStsWarpTransformErr	Indicates an error when <code>transform</code> has an illegal value.
ippStsQuadErr	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of <code>dstQuad</code> are changed by the function, if <code>transform</code> is set to <code>ippWarpAffine</code> .
ippStsBorderErr	Indicates an error when <code>borderType</code> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[WarpQuadGetSize](#) Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

## WarpAffineLinear

Performs warp affine transformation of an image using the linear interpolation method.

## Syntax

```
IppStatusippiWarpAffineLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R 64f\_C1R

8u\_C3R 16u\_C3R 16s\_C3R 32f\_C3R 64f\_C3R

8u\_C4R 16u\_C4R 16s\_C4R 32f\_C4R 64f\_C4R

## Include Files

ippi.h

Flavors with the `_L` suffix: `ippi_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<code>dstRoiOffset</code>	Offset of the destination image ROI with respect to the destination image origin.
<code>dstRoiSize</code>	Size of the destination image ROI, in pixels.
<code>pSpec</code>	Pointer to the specification structure for the warp operation.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

This function transforms the source image pixel coordinates  $(x, y)$  according to the following formulas:

$$x' = c_{00} * x + c_{01} * y + c_{02}$$

$$y' = c_{10} * x + c_{11} * y + c_{12}$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the `coeffs` array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The `WarpAffineLinear` function operates with ROI. The transformed part of the image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter `pSrc` must point to the source image origin. The parameter `pDst` must point to the processed destination image ROI.

If you initialize the specification structure using the `ippiWarpQuadLinearInit` function, the operations take place only inside the specified source quadrangle `srcQuad` that is set in `ippiWarpQuadLinearInit`.

To specify the algorithm for borders processing, set the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.

- If the border type is equal to `ippBorderTransp` or `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed.

Before using the `ippiWarpAffineLinear` function, you need to initialize the `IppiWarpSpec` structure using the `ippiWarpAffineLinearInit` function and compute the size of the external buffer `pBuffer` using the `WarpGetBufferSize` function.

To compute the affine transform parameters, use the `GetAffineQuad`, `GetAffineBound`, and `GetAffineTransform` functions.

For an example on how to use this function, refer to the `WarpQuadNearestInit` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsBorderErr</code>	Indicates an error if border type has an illegal value.
<code>ippStsContextMatchErr</code>	Indicates an error when context data is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image ROI is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not a multiple of data type.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image ROI size is more than the destination image origin size.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the destination ROI has no intersection with the transformed source image origin.

## See Also

[ROI Processing in Geometric Transforms](#)

[WarpAffineLinearInit](#) Initializes the specification structure for image affine warping with the linear interpolation method.

[WarpQuadLinearInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetAffineBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpAffine` function.

[GetAffineQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

[GetAffineTransform](#) Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

## **WarpAffineCubicInit**

*Initializes the specification structure for image affine warping with the cubic interpolation method.*

### **Syntax**

```
IppStatusippiWarpAffineCubicInit(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels,
Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue,
int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

### **Include Files**

ippi.h

Flavors with the \_L suffix: ippi\_L.h

### **Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

### **Parameters**

<i>srcSize</i>	Size of the source image, in pixels.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dataType</i>	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
<i>coeffs</i>	Coefficients for the affine transform.
<i>direction</i>	Transformation direction. Supported values: ippWarpForward Forward transformation ippWarpBackward Backward transformation
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>valueB, valueC</i>	The first (B) and second (C) parameter for the cubic filter.
<i>borderType</i>	Type of border. Supported values: ippBorderConst Values of all border pixels are set to a constant. ippBorderRepl Border is replicated from the edge pixels. ippBorderTransp Outer pixels are not processed. ippBorderInMem Border is obtained from the source image pixels in memory.

<i>pBorderValue</i>	Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>smoothEdge</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>	The smooth edge flag. The following values are supported: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.
<i>pInitBuf</i>	Pointer to the specification structure.
	Pointer to the temporary buffer for the cubic filter initialization.

## Description

This function initializes the `IppiWarpSpec` structure for the `ippiWarpAffineCubic` function that performs warp affine transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer *pInitBuf* using the `WarpAffineGetSize` function.

## Application Notes

Intel IPP warping functions do not support the `IPPI_INTER_CUBIC` mode. You can use interpolation with two-parameter cubic filters instead. This approach provides the interpolation quality that is comparable with `IPPI_INTER_CUBIC`. For interpolation formulas refer to [Interpolation with Two-Parameter Cubic Filters](#). You can vary *B* and *C* values to get a result that fits the required task.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li>• one of the specified pointers is <code>NULL</code>, excepting <i>pBorderValue</i></li> <li>• <i>pBorderValue</i> is <code>NULL</code> when border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsNoOperation</code>	Indicates a warning if width or height of any image is zero.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <i>direction</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error when affine transformation is singular.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the <code>ippBorderRepl</code> and <code>ippBorderConst</code> border types.
ippStsBorderErr	Indicates an error when <code>borderType</code> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <code>numChannels</code> has an illegal value.
ippStsExceededSizeErr	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0xFFFFFFFF).

## See Also

[WarpAffineGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

[Interpolation with Two-Parameter Cubic Filters](#)

## WarpQuadCubicInit

*Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.*

## Syntax

```
IppStatusippiWarpQuadCubicInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ipcore.h, ippvm.h, ipps.h

Libraries: ipcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcQuad</i>	Quadrangle in the source image.
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dstQuad</i>	Quadrangle in the destination image.
<i>transform</i>	Type of the warp transform. Supported values:  ippWarpAffine      Affine warping ippWarpPerspective      Perspective warping
<i>dataType</i>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>valueB</i> , <i>valueC</i>	The first (B) and second (C) parameter for the cubic filter.

<i>borderType</i>	Type of border. Supported values:
	ippBorderTransp Outer pixels are not processed.
	ippBorderInMem Border is obtained from the source image pixels in memory.
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>smoothEdge</i>	Flag for edge smoothing. Supported values:
	<ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing.</li> </ul>
<i>pSpec</i>	Pointer to the specification structure.
<i>pInitBuf</i>	Pointer to the temporary buffer for the cubic filter initialization.

## Description

This function initializes the `IppiWarpSpec` structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer *pInitBuf* using the [WarpQuadGetSize](#) function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle *dstQuad* and transform type *transform*. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in *dstQuad*, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

For an example on how to use this function, refer to the example provided with the [WarpQuadNearestInit](#) function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• One of the specified pointers is NULL, excepting <i>pBorderValue</i></li> <li>• The value of <i>pBorderValue</i> is NULL when the border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsWarpTransformErr</code>	Indicates an error when <i>transform</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of <i>dstQuad</i> are changed by the function, if <i>transform</i> is set to <code>ippWarpAffine</code> .
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[WarpQuadGetSize](#) Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

## WarpAffineCubic

*Performs warp affine transformation of an image using the cubic interpolation method.*

---

## Syntax

```
IppStatusippiWarpAffineCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C1R 16u\_C1R 16s\_C1R 32f\_C1R 64f\_C1R

8u\_C3R 16u\_C3R 16s\_C3R 32f\_C3R 64f\_C3R

8u\_C4R 16u\_C4R 16s\_C4R 32f\_C4R 64f\_C4R

## Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiOffset</i>	Offset of the destination image ROI with respect to the destination image origin.

---

<i>dstRoiSize</i>	Size of the destination image ROI, in pixels.
<i>pSpec</i>	Pointer to the specification structure for the warp operation.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = c_{00} \cdot x + c_{01} \cdot y + c_{02}$$

$$y' = c_{10} \cdot x + c_{11} \cdot y + c_{12}$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the *coeffs* array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The `WarpAffineCubic` function operates with ROI. The transformed part of the image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter *pSrc* must point to the source image origin. The parameter *pDst* must point to the processed destination image ROI.

If you initialize the specification structure using the `ippiWarpQuadCubicInit` function, the operations take place only inside the specified source quadrangle *srcQuad* that is set in `ippiWarpQuadCubicInit`.

To specify the algorithm for borders processing, set the *borderType* and *pBorderValue* parameters when initializing the `IppiWarpSpec` structure. The data type of *pBorderValue* is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in *pBorderValue*.
- If the border type is equal to `ippBorderTransp`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.
- If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation OR between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values

Before using the `ippiWarpAffineCubic` function, you need to initialize the `IppiWarpSpec` structure using the `ippiWarpAffineCubicInit` function and compute the size of the external buffer *pBuffer* using the `WarpGetBufferSize` function.

To compute the affine transform parameters, use the `GetAffineQuad`, `GetAffineBound`, and `GetAffineTransform` functions.

For an example on how to use this function, refer to the `WarpQuadNearestInit` function description.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error if border type has an illegal value.
ippStsContextMatchErr	Indicates an error when context data is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed, if the transformed source image has no intersection with the destination image.

## See Also

### [ROI Processing in Geometric Transforms](#)

[WarpAffineCubicInit](#) Initializes the specification structure for image affine warping with the cubic interpolation method.

[WarpQuadCubicInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetAffineBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpAffine` function.

[GetAffineQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

[GetAffineTransform](#) Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

## [GetPerspectiveQuad](#)

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.*

## Syntax

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[3][3]);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib`

## Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
<code>coeffs</code>	The given perspective transform coefficients.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for the `WarpPerspectiveNearest`, `WarpPerspectiveLinear`, and `WarpPerspectiveCubic` functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the perspective transform function using the given coefficients `coeffs`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad[0]` corresponds to the transformed top-left corner of the source ROI,

`quad[1]` corresponds to the transformed top-right corner of the source ROI,

`quad[2]` corresponds to the transformed bottom-right corner of the source ROI,

`quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetPerspectiveBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.*

## Syntax

```
IppStatusippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2], const double coeffs[3][3]);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib`

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given perspective transform coefficients.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for the `WarpPerspectiveNearest`, `WarpPerspectiveLinear`, and `WarpPerspectiveCubic` functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the perspective transform function using the given coefficients *coeffs*.

*bound[0]* specifies x, y coordinates of the top-left corner, *bound[1]* specifies x, y coordinates of the bottom-right corner.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

## GetPerspectiveTransform

*Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

## Syntax

```
IppStatusippiGetPerspectiveTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[3][3]);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
<i>coeffs</i>	Output array. Contains the target perspective transform coefficients.

## Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the [WarpPerspectiveNearest](#), [WarpPerspectiveLinear](#), and [WarpPerspectiveCubic](#) functions. It computes the coefficients  $\text{coeffs}$  that should be used by the function to map the source rectangular ROI to the quadrangle with the given vertex coordinates  $\text{quad}$ .

The first dimension [4] of the array  $\text{quad}[4][2]$  is equal to the number of vertices, and the second dimension [2] means  $x$  and  $y$  coordinates of the vertex. Quadrangle vertices have the following meaning:

$\text{quad}[0]$  corresponds to the transformed top-left corner of the source ROI,

$\text{quad}[1]$  corresponds to the transformed top-right corner of the source ROI,

$\text{quad}[2]$  corresponds to the transformed bottom-right corner of the source ROI,

$\text{quad}[3]$  corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if $\text{srcRoi}$ has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsRectErr	Indicates an error condition if width or height of the $\text{srcRoi}$ is less than or equal to 1.

## Example

To better understand usage of the `ippiGetPerspectiveTransform` function, refer to the `GetPerspectiveTransform.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## WarpGetRectInfinite

Returns an infinite rectangle.

### Syntax

```
IppiRect ippiWarpGetRectInfinite(void);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Description

The function returns a constant rectangle that is considered as an infinite rectangle by Intel IPP `WarpPerspective` functions. Use this rectangle in the following functions: `WarpPerspectiveGetSize`, `WarpPerspectiveInitNearest`, `WarpPerspectiveInitLinear`, and `WarpPerspectiveCubic`.

**NOTE**

The macro definition is: #define `ippRectInfinite` `ippiWarpGetRectInfinite()`.

---

**WarpPerspectiveGetSize**

*Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.*

---

**Syntax**

```
IppStatus ippiWarpPerspectiveGetSize(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiInterpolationType interpolation, IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

<code>srcSize</code>	Size of the source image, in pixels.
<code>srcRoi</code>	Source image ROI (of the <code>IppiRect</code> type).
<code>dstSize</code>	Size of the destination image, in pixels.
<code>dataType</code>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<code>coeffs</code>	Coefficients for the perspective transform.
<code>interpolation</code>	Interpolation method. Supported values: <code>ippNearest</code> , <code>ippLinear</code> , and <code>ippCubic</code> .
<code>direction</code>	Transformation direction. Supported values:  <code>ippWarpForward</code> Forward transformation <code>ippWarpBackward</code> Backward transformation
<code>borderType</code>	Type of border. Supported values:  <code>ippBorderConst</code> Values of all border pixels are set to a constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderTransposed</code> Outer pixels are not processed.

<code>pSpecSize</code>	Pointer to the size, in bytes, of the specification structure.
<code>pInitBufSize</code>	Pointer to the size, in bytes, of the temporary buffer.

## Description

This function computes the size of the specification structure and the external work buffer for the following functions, depending on the `interpolation` parameter: `ippiWarpPerspectiveNearestInit`, `ippiWarpPerspectiveLinearInit`, or `ippiWarpPerspectiveCubicInit`.

You can set the value of the `srcRoi` parameter to `ippRectInfinite`, which means that the ROI is not specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• If the width or height of the source or destination image is less than, or equal to one.</li> <li>• If one of the calculated sizes exceeds the maximum positive 32-bit signed integer value. The size of the one of the processed images is too large.</li> </ul>
<code>ippStsRectErr</code>	Indicates an error in the following cases, if the source image ROI is not <code>ippRectInfinite</code> : <ul style="list-style-type: none"> <li>• If the source image ROI has no intersection with the image.</li> <li>• Either <code>x</code> or <code>y</code> component of the source image ROI is negative.</li> <li>• Width or height of the source image ROI is less than, or equal to zero.</li> </ul>
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <code>direction</code> has an illegal value.
<code>ippStsInterpolationErr</code>	Indicates an error when <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error when perspective transformation is singular.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an illegal value.
<code>ippStsSizeWrn</code>	Indicates a warning when <code>srcRoi</code> exceeds the source image.

## See Also

[WarpPerspectiveNearestInit](#) Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

[WarpPerspectiveLinearInit](#) Initializes the specification structure for image perspective warping with the linear interpolation method.

[WarpPerspectiveCubicInit](#) Initializes the specification structure for image perspective warping with the cubic interpolation method.

### [WarpPerspectiveNearestInit](#)

*Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.*

---

## Syntax

```
IppStatusippiWarpPerspectiveNearestInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcRoi</i>	Source image ROI (of the <a href="#">IppiRect type</a> ).
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dataType</i>	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
<i>coeffs</i>	Coefficients for the perspective transform.
<i>direction</i>	Transformation direction. Supported values:  ippWarpForward Forward transformation  ippWarpBackward Backward transformation
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>borderType</i>	Type of border. Supported values:  ippBorderConst Values of all border pixels are set to a constant.  ippBorderRepl Border is replicated from the edge pixels.

<i>pBorderValue</i>	<code>ippBorderTrans p</code>	Outer pixels are not processed.
	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
		Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>smoothEdge</i>		Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pSpec</i>		Flag for edge smoothing. Supported values: <ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.</li> </ul>
		Pointer to the specification structure.

## Description

This function initializes the `IppiWarpSpec` structure for the `WarpPerspectiveNearest` function that performs warp perspective transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the `WarpPerspectiveGetSize` function.

You can set the value of the `srcRoi` parameter to `ippRectInfinite`, which means that the ROI is not specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>pSpec</code> is NULL</li> <li>• <code>pBorderValue</code> is NULL when the border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsRectErr</code>	Indicates an error in the following cases, if the source image ROI is not <code>ippRectInfinite</code> : <ul style="list-style-type: none"> <li>• If the source image ROI has no intersection with the image.</li> <li>• Either <code>x</code> or <code>y</code> component of the source image ROI is negative.</li> <li>• Width or height of the source image ROI is less than, or equal to zero.</li> </ul>
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <code>direction</code> has an illegal value.

ippStsCoeffErr	Indicates an error when perspective transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the <code>ippBorderRepl</code> and <code>ippBorderConst</code> border types.
ippStsBorderErr	Indicates an error when <code>borderType</code> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <code>numChannels</code> has an illegal value.
ippStsSizeWrn	Indicates a warning when <code>srcRoi</code> exceeds the source image.

## See Also

[WarpPerspectiveGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

[WarpPerspectiveNearest](#) Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

## WarpPerspectiveNearest

Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

### Syntax

```
IppStatusippiWarpPerspectiveNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
--------	---------	---------	---------

8u_C3R	16u_C3R	16s_C3R	32f_C3R
--------	---------	---------	---------

8u_C4R	16u_C4R	16s_C4R	32f_C4R
--------	---------	---------	---------

### Include Files

ippi.h

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image ROI.

---

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiOffset</i>	Offset of the destination image ROI with respect to the destination image origin.
<i>dstRoiSize</i>	Size of the destination image ROI, in pixels.
<i>pSpec</i>	Pointer to the specification structure for the warp operation.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = (c_{00} \cdot x + c_{01} \cdot y + c_{02}) / (c_{20} \cdot x + c_{21} \cdot y + c_{22})$$

$$y' = (c_{10} \cdot x + c_{11} \cdot y + c_{12}) / (c_{20} \cdot x + c_{21} \cdot y + c_{22})$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the *coeffs* array during initialization

The `ippiWarpPerspectiveNearest` function operates with ROI (see [Regions of Interest in Intel IPP](#)). The transformed part of the source image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter *pDst* must point to the processed destination image ROI.

If you initialize the warp specification structure using the [WarpPerspectiveNearestInit](#) function, you can specify the source image ROI in the following ways:

- Set the *srcRoi* value to `ippRectInfinite`, which means that the ROI is not specified. In this case, *pSrc* must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the *srcRoi* value to the part of the processed source image. In this case, *pSrc* must point to the processed source image ROI. The operations take place only inside the specified region of interest *srcRoi*. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the [WarpQuadNearestInit](#) function, set the *pSrc* value to the processed source image. The operations take place only inside the source quadrangle *srcQuad* that is specified in the [WarpQuadNearestInit](#) function.

To specify the algorithm for borders processing, set the *borderType* and *pBorderValue* parameters when initializing the `IppiWarpSpec` structure. The data type of *borderValue* is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in *pBorderValue*.
- If the border type is equal to `ippBorderTransp` or `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed.

Before using the `ippiWarpPerspectiveNearest` function, you need to initialize the `IppiWarpSpec` structure using the [WarpPerspectiveNearestInit](#) function and compute the size of the external buffer *pBuffer* using the [WarpGetBufferSize](#) function.

To compute the perspective transform parameters, use the [GetPerspectiveQuad](#), [GetPerspectiveBound](#), and [GetPerspectiveTransform](#) functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error when context data is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image ROI is negative.
<code>ippStsStepErr</code>	Indicates an error when the step value is not a multiple of data type.
<code>ippStsOutOfRangeErr</code>	Indicates an error when the destination image offset point is outside the destination image origin.
<code>ippStsSizeWrn</code>	Indicates a warning when the destination image ROI size is more than the destination image origin size.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

## Example

To better understand usage of Intel IPP functionality for perspective warping with Nearest Neighbor, Linear, and Cubic interpolation modes, refer to the `WarpPerspectiveNearest.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

### [ROI Processing in Geometric Transforms](#)

[WarpPerspectiveNearestInit](#) Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

[WarpQuadNearestInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetPerspectiveBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.

[GetPerspectiveQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

[GetPerspectiveTransform](#) Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

## [WarpPerspectiveLinearInit](#)

*Initializes the specification structure for image perspective warping with the linear interpolation method.*

## Syntax

```
IppStatusippiWarpPerspectiveLinearInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcRoi</i>	Source image ROI (of the <a href="#">IppiRect type</a> ).
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dataType</i>	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
<i>coeffs</i>	Coefficients for the perspective transform.
<i>direction</i>	Transformation direction. Supported values:  ippWarpForward Forward transformation  ippWarpBackward Backward transformation
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>borderType</i>	Type of border. Supported values:  ippBorderConst Values of all border pixels are set to a constant.  ippBorderRepl Border is replicated from the edge pixels.  ippBorderTransp Outer pixels are not processed.  ippBorderInMem Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>smoothEdge</i>	Flag for edge smoothing. Supported values: <ul style="list-style-type: none"> <li>• 0 - transformation without edge smoothing.</li> <li>• 1 - transformation with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.</li> </ul>
<i>pSpec</i>	Pointer to the specification structure.

## Description

This function initializes the `IppiWarpSpec` structure for the `ippiWarpPerspectiveLinear` function that performs warp perspective transformation with the linear interpolation method. To compute the size of the specification structure, use the `WarpPerspectiveGetSize` function.

You can set the value of the `srcRoi` parameter to `ippRectInfinite`, which means that the ROI is not specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <i>pSpec</i> is NULL</li> <li>• <i>pBorderValue</i> is NULL when the border type is set to <code>ippBorderConst</code></li> </ul>
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsRectErr</code>	Indicates an error in the following cases, if the source image ROI is not <code>ippRectInfinite</code> : <ul style="list-style-type: none"> <li>• If the source image ROI has no intersection with the image.</li> <li>• Either <code>x</code> or <code>y</code> component of the source image ROI is negative.</li> <li>• Width or height of the source image ROI is less than, or equal to zero.</li> </ul>
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <code>direction</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error when perspective transformation is singular.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the <code>ippBorderRepl</code> and <code>ippBorderConst</code> border types.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

ippStsSizeWrn Indicates a warning when *srcRoi* exceeds the source image.

## See Also

[WarpPerspectiveGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

[WarpPerspectiveLinear](#) Performs warp perspective transformation of an image using the linear interpolation method.

## WarpPerspectiveLinear

*Performs warp perspective transformation of an image using the linear interpolation method.*

### Syntax

```
IppStatusippiWarpPerspectiveLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiOffset</i>	Offset of the destination image ROI with respect to the destination image origin.
<i>dstRoiSize</i>	Size of the destination image ROI, in pixels.
<i>pSpec</i>	Pointer to the specification structure for the warp operation.
<i>pBuffer</i>	Pointer to the work buffer.

### Description

This function transforms the source image pixel coordinates (*x*, *y*) according to the following formulas:

$$x' = (c_{00} \cdot x + c_{01} \cdot y + c_{02}) / (c_{20} \cdot x + c_{21} \cdot y + c_{22})$$

$$y' = (c_{10}x + c_{11}y + c_{12}) / (c_{20}x + c_{21}y + c_{22})$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the `coeffs` array during initialization

The `ippiWarpPerspectiveLinear` function operates with ROI (see [Regions of Interest in Intel IPP](#)). The transformed part of the source image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter `pDst` must point to the processed destination image ROI.

If you initialize the warp specification structure using the `ippiWarpPerspectiveLinearInit` function, you can specify the source image ROI in the following ways:

- Set the `srcRoi` value to `ippRectInfinite`, which means that the ROI is not specified. In this case, `pSrc` must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the `srcRoi` value to the part of the processed source image. In this case, `pSrc` must point to the processed source image ROI. The operations take place only inside the specified region of interest `srcRoi`. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the `ippiWarpQuadLinearInit` function, set the `pSrc` value to the processed source image. The operations take place only inside the source quadrangle `srcQuad` that is specified in the `ippiWarpQuadLinearInit` function.

To specify the algorithm for borders processing, set the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.
- If the border type is equal to `ippBorderTransp` or `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed.

Before using the `ippiWarpPerspectiveLinear` function, you need to initialize the `IppiWarpSpec` structure using the `ippiWarpPerspectiveLinearInit` function and compute the size of the external buffer `pBuffer` using the `ippiWarpGetBufferSize` function.

To compute the perspective transform parameters, use the `ippiGetPerspectiveQuad`, `ippiGetPerspectiveBound`, and `ippiGetPerspectiveTransform` functions.

For an example on how to use this functionality, refer to the example provided with the `ippiWarpPerspectiveNearest` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error when context data is invalid.

---

ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

## See Also

### [ROI Processing in Geometric Transforms](#)

[WarpPerspectiveLinearInit](#) Initializes the specification structure for image perspective warping with the linear interpolation method.

[WarpQuadLinearInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetPerspectiveBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.

[GetPerspectiveQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

[GetPerspectiveTransform](#) Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

[WarpPerspectiveNearest](#) Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

## [WarpPerspectiveCubicInit](#)

*Initializes the specification structure for image perspective warping with the cubic interpolation method.*

## Syntax

```
IppStatus ippiWarpPerspectiveCubicInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

*srcSize*

Size of the source image, in pixels.

<i>srcRoi</i>	Source image ROI (of the <code>IppiRect</code> type).
<i>dstSize</i>	Size of the destination image, in pixels.
<i>dataType</i>	Data type of the source and destination images. Supported values: <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , and <code>ipp32f</code> .
<i>coeffs</i>	Coefficients for the perspective transform.
<i>direction</i>	Transformation direction. Supported values: <code>ippWarpForward</code> Forward transformation <code>ippWarpBackward</code> Backward transformation <code>d</code>
<i>numChannels</i>	Number of channels in the image. Supported values: 1, 3, or 4.
<i>valueB</i> , <i>valueC</i>	The first (B) and second (C) parameter for the cubic filter.
<i>borderType</i>	Type of border. Supported values:  <code>ippBorderConst</code> Values of all border pixels are set to a constant. <code>ippBorderRepl</code> Border is replicated from the edge pixels. <code>ippBorderTransp</code> Outer pixels are not processed. <code>ippBorderInMem</code> Border is obtained from the source image pixels in memory.  Mixed borders are also supported. They can be obtained by the bitwise operation <code>OR</code> between <code>ippBorderTransp</code> and the <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> values.
<i>pBorderValue</i>	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>smoothEdge</i>	Flag to enable/disable edge smoothing. Possible values: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the <code>ippBorderTransp</code> and <code>ippBorderInMem</code> border types.
<i>pSpec</i>	Pointer to the specification structure.
<i>pInitBuf</i>	Pointer to the temporary buffer for the cubic filter initialization.

## Description

This function initializes the `IppiWarpSpec` structure for the `ippiWarpPerspectiveCubic` function that performs warp perspective transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer `pInitBuf` using the `WarpPerspectiveGetSize` function.

You can set the value of the `srcRoi` parameter to `ippRectInfinite`, which means that the ROI is not specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when: <ul style="list-style-type: none"><li><code>pSpec</code> is NULL</li><li><code>pBorderValue</code> is NULL when the border type is set to <code>ippBorderConst</code></li></ul>
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image is less than, or equal to one.
<code>ippStsRectErr</code>	Indicates an error in the following cases, if the source image ROI is not <code>ippRectInfinite</code> : <ul style="list-style-type: none"><li>If the source image ROI has no intersection with the image.</li><li>Either <code>x</code> or <code>y</code> component of the source image ROI is negative.</li><li>Width or height of the source image ROI is less than, or equal to zero.</li></ul>
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsWarpDirectionErr</code>	Indicates an error when <code>direction</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error when perspective transformation is singular.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the <code>ippBorderRepl</code> and <code>ippBorderConst</code> border types.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.
<code>ippStsSizeWrn</code>	Indicates a warning when <code>srcRoi</code> exceeds the source image.

## See Also

[WarpPerspectiveGetSize](#) Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

[WarpPerspectiveCubic](#) Performs warp perspective transformation of an image using the cubic interpolation method.

## WarpPerspectiveCubic

*Performs warp perspective transformation of an image using the cubic interpolation method.*

## Syntax

```
IppStatusippiWarpPerspectiveCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<code>dstRoiOffset</code>	Offset of the destination image ROI with respect to the destination image origin.
<code>dstRoiSize</code>	Size of the destination image ROI, in pixels.
<code>pSpec</code>	Pointer to the specification structure for the warp operation.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

This function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = (c_{00} \cdot x + c_{01} \cdot y + c_{02}) / (c_{20} \cdot x + c_{21} \cdot y + c_{22})$$

$$y' = (c_{10} \cdot x + c_{11} \cdot y + c_{12}) / (c_{20} \cdot x + c_{21} \cdot y + c_{22})$$

where

- $x'$  and  $y'$  are the pixel coordinates in the transformed image
- $c_{ij}$  are the affine transform coefficients passed to the `coeffs` array during initialization

The `ippiWarpPerspectiveCubic` function operates with ROI (see [Regions of Interest in Intel IPP](#)). The transformed part of the source image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter `pDst` must point to the processed destination image ROI.

If you initialize the warp specification structure using the [ippiWarpPerspectiveCubicInit](#) function, you can specify the source image ROI in the following ways:

- Set the *srcRoi* value to `ippRectInfinite`, which means that the ROI is not specified. In this case, *pSrc* must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the *srcRoi* value to the part of the processed source image. In this case, *pSrc* must point to the processed source image ROI. The operations take place only inside the specified region of interest *srcRoi*. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the [ippiWarpQuadCubicInit](#) function, set the *pSrc* value to the processed source image. The operations take place only inside the source quadrangle *srcQuad* that is specified in the [ippiWarpQuadCubicInit](#) function.

To specify the algorithm for borders processing, set the *borderType* and *pBorderValue* parameters when initializing the `IppiWarpSpec` structure. The data type of *pBorderValue* is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in *pBorderValue*.
- If the border type is equal to `ippBorderTransp`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.
- If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation OR between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values

Before using the [ippiWarpPerspectiveCubic](#) function, you need to initialize the `IppiWarpSpec` structure using the [ippiWarpPerspectiveCubicInit](#) function and compute the size of the external buffer *pBuffer* using the [ippiWarpGetBufferSize](#) function.

To compute the perspective transform parameters, use the [ippiGetPerspectiveQuad](#), [ippiGetPerspectiveBound](#), and [ippiGetPerspectiveTransform](#) functions.

For an example on how to use this functionality, refer to the example provided with the [ippiWarpPerspectiveNearest](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsNoOperation</code>	Indicates a warning when width or height of the destination image is equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error when context data is invalid.
<code>ippStsNotSupportedModeErr</code>	Indicates an error when the requested mode is not supported.
<code>ippStsSizeErr</code>	Indicates an error when width or height of the source or destination image ROI is negative.

ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

## See Also

### [ROI Processing in Geometric Transforms](#)

[WarpPerspectiveCubicInit](#) Initializes the specification structure for image perspective warping with the cubic interpolation method.

[WarpQuadCubicInit](#) Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

[WarpGetBufferSize](#) Computes the size of the work buffer for the warp transform.

[GetPerspectiveBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.

[GetPerspectiveQuad](#) Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

[GetPerspectiveTransform](#) Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

[WarpPerspectiveNearest](#) Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

## GetBilinearQuad

*Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform.*

## Syntax

```
IppStatusippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2], const double  
coeffs[2][4]);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
<code>coeffs</code>	The given bilinear transform coefficients.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform function [ippiWarpBilinear](#) using coefficients *coeffs*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad*[0] corresponds to the transformed top-left corner of the source ROI,

*quad*[1] corresponds to the transformed top-right corner of the source ROI,

*quad*[2] corresponds to the transformed bottom-right corner of the source ROI,

*quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

## GetBilinearBound

*Computes the bounding rectangle for the source ROI transformed by the [ippiWarpBilinear](#) function.*

## Syntax

```
IppStatusippiGetBilinearBound(IppiRect srcRoi, double bound[2][2], const double  
                          coeffs[2][4]);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given bilinear transform coefficients.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the bilinear transform function [ippiWarpBilinear](#) using coefficients *coeffs*.

*bound[0]* specifies x, y coordinates of the top-left corner, *bound[1]* specifies x, y coordinates of the bottom-right corner.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

## GetBilinearTransform

*Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

---

## Syntax

```
IppStatusippiGetBilinearTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[2][4]);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>quad</i>	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
<i>coeffs</i>	Output array. Contains the target bilinear transform coefficients.

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes the coefficients *coeffs* of the bilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates *quad*.

The first dimension [4] of the array *quad[4][2]* is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

*quad[0]* corresponds to the transformed top-left corner of the source ROI,

*quad[1]* corresponds to the transformed top-right corner of the source ROI,

*quad[2]* corresponds to the transformed bottom-right corner of the source ROI,

*quad[3]* corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsRectErr	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.

## Example

To better understand usage of the `ippiGetBilinearTransform` function, refer to the `GetBilinearTransform.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## WarpBilinearGetBufferSize

Computes the size of the work buffer for bilinear warping.

### Syntax

```
IppStatusippiWarpBilinearGetBufferSize(IppiSize srcSize, IppiRect srcRoi, IppiRect dstRoi, IppiWarpDirection direction, const double coeffs[2][4], int interpolation, int* pBufSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<i>srcSize</i>	Size of the source image, in pixels.
<i>srcRoi</i>	Source image ROI (of the <code>IppiRect</code> type).
<i>dstRoi</i>	Destination image ROI (of the <code>IppiRect</code> type).
<i>direction</i>	Transformation direction. Supported values:  ippWarpForward      Forward transformation ippWarpBackward      Backward transformation
<i>coeffs</i>	Coefficients for the perspective transform.
<i>interpolation</i>	Interpolation mode. Supported values:  IPPI_INTER_NN      Nearest neighbor interpolation IPPI_INTER_LINEAR    Linear interpolation IPPI_INTER_CUBIC     Cubic interpolation

	IPPI_INTER_EDGE	Use edge smoothing in addition to one of the above modes
<i>pBufSize</i>		Pointer to the size, in bytes, of the external buffer.

## Description

This function computes the size of the external work buffer required for bilinear warping of the source image ROI. The result is stored in the *pBufSize* parameter.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pBufSize</i> is NULL.
ippStsSizeErr	Indicates an error when one of the image dimensions is less than, or equal to zero.
ippStsWarpDirectionErr	Indicates an error when <i>direction</i> has an illegal value.
ippStsCoeffErr	Indicates an error when bilinear transformation is singular.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

## See Also

[WarpBilinear](#) MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

## WarpBilinear

MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

---

## Syntax

```
IppStatusippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>coeffs</i>	The bilinear transform coefficients.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values:  IPPI_INTER_NN      Nearest neighbor interpolation IPPI_INTER_LIN      Linear interpolation EAR IPPI_INTER_CUB      Cubic interpolation IC IPPI_SMOOTH_ED      Use edge smoothing in addition to one of the above modes. GE
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

---

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This bilinear warp function transforms the source image pixel coordinates ( $x, y$ ) according to the following formulas:

$$x' = c_{00} * x * y + c_{01} * x + c_{02} * y + c_{03}$$

$$y' = c_{10} * x * y + c_{11} * x + c_{12} * y + c_{13}$$

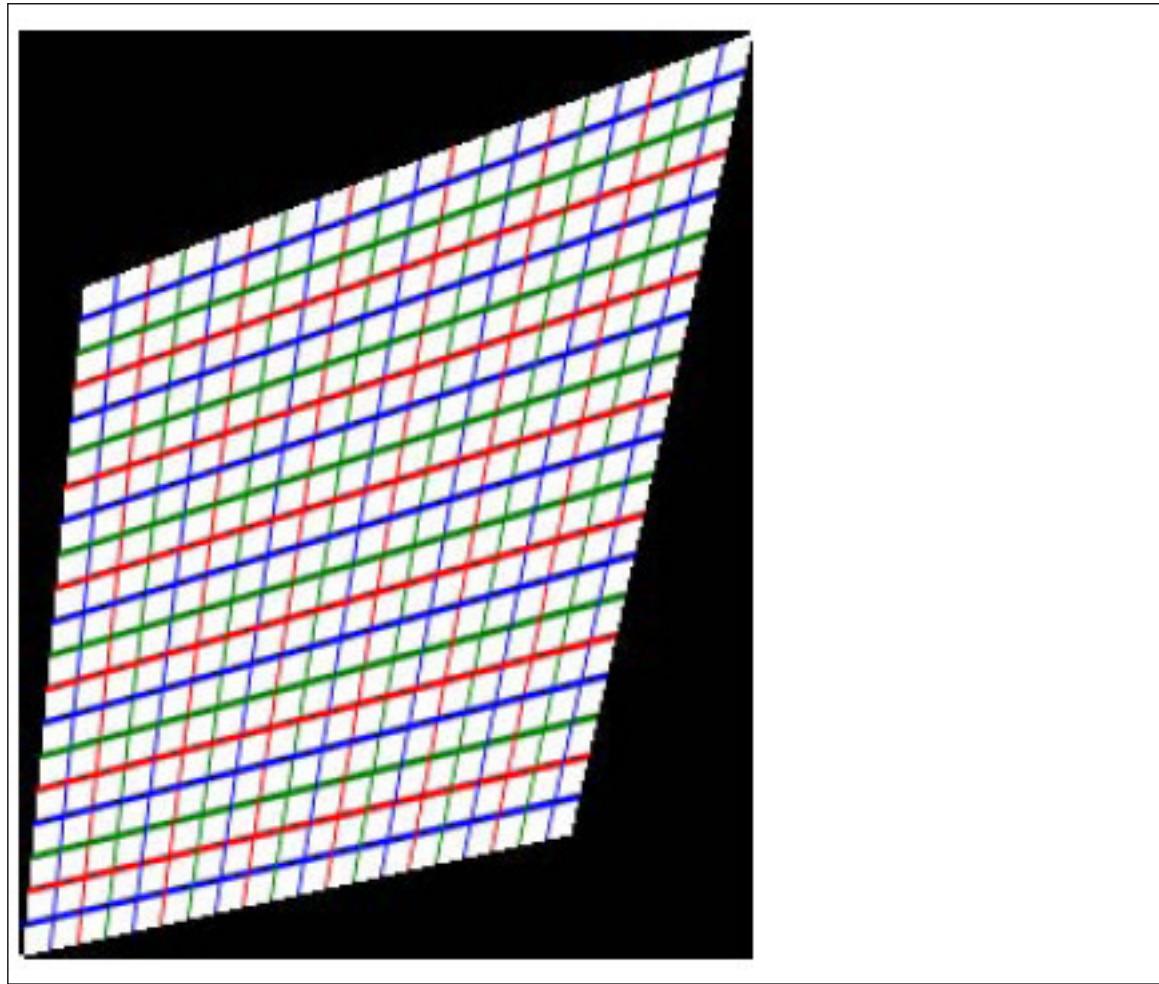
where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the bilinear transform coefficients passed in the array *coeffs*.

The bilinear transform preserves equal distances between points on a line.

The transformed part of the source image is resampled using the [interpolation mode](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure “Bilinear Transform of an Image”](#) gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

### Bilinear Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions `ippiWarpBilinearQuad` and `ippiGetBilinearBound`. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use `ippiGetBilinearTransform` function.

Before using this function, compute the size of the external work buffer *pBuffer* using the `WarpBilinearGetBufferSize` function.

[Example](#) shows how to use the `ippiWarpBilinear_32f_C1R` function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.

---

ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if <i>interpolation</i> has an illegal value.
ippStsRectErr	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsWrongIntersectROIErr	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## See Also

[Regions of Interest in Intel IPP](#)

[WarpBilinearQuad](#) MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

[GetBilinearBound](#) Computes the bounding rectangle for the source ROI transformed by the `ippiWarpBilinear` function.

[GetBilinearTransform](#) Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

[WarpBilinearGetBufferSize](#) Computes the size of the work buffer for bilinear warping.

## WarpBilinearBack

*MODIFIED API. Performs an inverse bilinear warping of the source image.*

---

### Syntax

```
IppStatusippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>coeffs</i>	The bilinear transform coefficients.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values:
	IPPI_INTER_NN    Nearest neighbor interpolation
	IPPI_INTER_LIN    Linear interpolation
	EAR
	IPPI_INTER_CUB    Cubic interpolation.
	IC
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

---

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by `ippiWarpBilinear` function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations

$$c_{00} * x' * y' + c_{01} * x' + c_{02} * y' + c_{03} = x$$

$$c_{10} * x' * y' + c_{11} * x' + c_{12} * y' + c_{13} = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array *coeffs*. Thus, you do not need to invert transform coefficients in your application program before calling `ippiWarpBilinearBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. The implementation of the inverse transform functions has the following logic:

- Backward transform is applied to coordinates of each pixel in the destination ROI. The result is coordinates of some pixel in the source image.
- If the obtained source pixel is inside the source ROI, the corresponding pixel in the destination ROI is modified accordingly; otherwise, no changes are made.

Before using this function, compute the size of the external work buffer *pBuffer* using the [WarpBilinearGetBufferSize](#) function.

[Example](#) shows how to use the function `ippiWarpBilinearBack_32f_C1R`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

## See Also

[Regions of Interest in Intel IPP](#)

[WarpBilinear](#) MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

[WarpBilinearGetBufferSize](#) Computes the size of the work buffer for bilinear warping.

## WarpBilinearQuadGetBufferSize

*Computes the size of the work buffer for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image.*

## Syntax

```
IppStatus ippiWarpBilinearQuadGetBufferSize(IppiSize srcSize, IppiRect srcRoi, const double srcQuad[4][2], IppiRect dstRoi, const double dstQuad[4][2], int interpolation, int* pBufSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcSize</code>	Size of the source image, in pixels.
----------------------	--------------------------------------

<i>srcRoi</i>	Source image ROI (of the <code>IppiRect</code> type).
<i>srcQuad</i>	Quadrangle in the source image.
<i>dstRoi</i>	Destination image ROI (of the <code>IppiRect</code> type).
<i>dstQuad</i>	Quadrangle in the destination image.
<i>interpolation</i>	Interpolation mode. Supported values:  IPPI_INTER_NN      Nearest neighbor interpolation IPPI_INTER_LINEAR    Linear interpolation IPPI_INTER_CUBIC     Cubic interpolation IPPI_INTER_EDGE     Use edge smoothing in addition to one of the above modes
<i>pBufSize</i>	Pointer to the size, in bytes, of the external buffer.

## Description

This function computes the size of the external work buffer required for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image. The result is stored in the *pBufSize* parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error: <ul style="list-style-type: none"><li>• If one of the ROI coordinates has a negative value.</li><li>• If one of the ROI dimensions is less than, or equal to zero.</li></ul>
<code>ippStsCoeffErr</code>	Indicates an error when bilinear transformation is singular.
<code>ippStsInterpolationErr</code>	Indicates an error when <i>interpolation</i> has an illegal value.

## See Also

[WarpBilinearQuad](#) MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

## WarpBilinearQuad

MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

## Syntax

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int
srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep,
IppiRect dstRoi, const double dstQuad[4][2], int interpolation, Ipp8u* pBuffer);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

---

8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>srcQuad</i>	A given quadrangle in the source image.
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>dstQuad</i>	A given quadrangle in the destination image.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Use one of the following values:
	IPPI_INTER_NN      Nearest neighbor interpolation
	IPPI_INTER_LIN      Linear interpolation
	EAR
	IPPI_INTER_CUB      Cubic interpolation
	IC
	IPPI_SMOOTH_ED      Use edge smoothing in addition to one of the above modes.
	GE
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

---

**Important** The API of this function has been modified in Intel IPP 9.0 release.

---

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies a bilinear transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle *srcQuad*. The function `ippiWarpBilinearQuad` uses the same formulas for pixel mapping as in the case of the `ippiWarpBilinear` function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad[4][2]* or *dstQuad[4][2]* is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

Before using this function, compute the size of the external work buffer *pBuffer* using the [WarpBilinearGetBufferSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .

## See Also

[Regions of Interest in Intel IPP](#)

[WarpBilinear](#) MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

[WarpBilinearGetBufferSize](#) Computes the size of the work buffer for bilinear warping.

## Mirror

[Mirrors an image about the specified axis \(axes\).](#)

## Syntax

### Case 1: Not-in-place operation

```
IppStatusippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiAxis flip);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

### Case 2: In-place operation

```
IppStatusippiMirror_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiAxis flip);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source buffer.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for the in-place operation.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.

<i>roiSize</i>	Size of the destination ROI in pixels.
<i>flip</i>	Specifies the axis to mirror the image about. Use the following values to specify the axes:
ippAxsHorizontal	for the horizontal axis.
al	
ippAxsVertical	for the vertical axis.
ippAxsBoth	for both horizontal and vertical axes.
ippAxs45	for the 45-degree rotated axis.
ippAxs135	for the 135-degree rotated axis.

## Description

The `ippiMirror` function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function mirrors the source image *pSrc* about the axis (axes) specified by the value of the *flip* parameter and writes the result to the destination image *pDst*. Each function flavor can mirror an image about the horizontal or vertical axis or both.

The `ippiMirror_8u_C1R`, `ippiMirror_16u_C1R`, `ippiMirror_16s_C1R`, and `ippiMirror_32f_C1R` function flavors can also use the `ippAxs45` or `ippAxs135` value of the *flip* parameter to mirror the source image about an axis rotated counterclockwise by 45 degrees or 135 degrees, respectively. For mirroring with each of these values of the *flip* parameter, the sizes of the source and destination ROI are different, and

```
roiSize.height = srcRoiSize.width  
roiSize.width = srcRoiSize.height
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or when one of the dimensions is equal to 1.
<code>ippStsMirrorFlipErr</code>	Indicates an error condition if <i>flip</i> has an illegal value.
<code>ippStsNotSupportedModeErr</code>	Indicates an error condition if intersection of the source and destination ROI is detected.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images)

## Examples

To better understand usage of the `ippiMirror` function, refer to the `Mirror1.c` and `Mirror2.c` examples in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Remap

Performs the look-up coordinate mapping of pixels of the source image.

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatusippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatusippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp64f* pxMap, int xMapStep, const Ipp64f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for mod:

64f_C1R
64f_C3R
64f_C4R
64f_AC4R

```
IppStatusippiRemap_8u_AC4R(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size, in pixels, of the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the IppiRect type).
<i>pxMap, pyMap</i>	Pointers to the starts of 2D buffers, containing tables of the <i>x</i> - and <i>y</i> -coordinates.

<i>xMapStep</i> , <i>yMapStep</i>	Steps, in bytes, through the buffers containing tables of the <i>x</i> - and <i>y</i> -coordinates.
<i>pDst</i>	Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>interpolation</i>	Specifies the <a href="#">interpolation</a> mode. Possible values are:  IPPI_INTER_NN - nearest neighbor interpolation IPPI_INTER_LINEAR - linear interpolation IPPI_INTER_CUBIC - cubic interpolation IPPI_INTER_LANCZOS - interpolation with Lanczos window IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline the following flag is used additionally to the above modes: IPPI_SMOOTH_EDGE - edge smoothing

## Description

This function operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function transforms the source image by remapping its pixels. Pixel remapping is performed using *pxMap* and *pyMap* buffers to look-up the coordinates of the source image pixel that is written to the target destination image pixel. The application has to supply these look-up tables. The remapping of the source pixels to the destination pixels is made according to the following formula:

*dst\_pixel[i, j]=src\_pixel[pxMap[i, j], pyMap[i, j]]*

where *i, j* are the *x*- and *y*-coordinates of the target destination image pixel *dst\_pixel*;

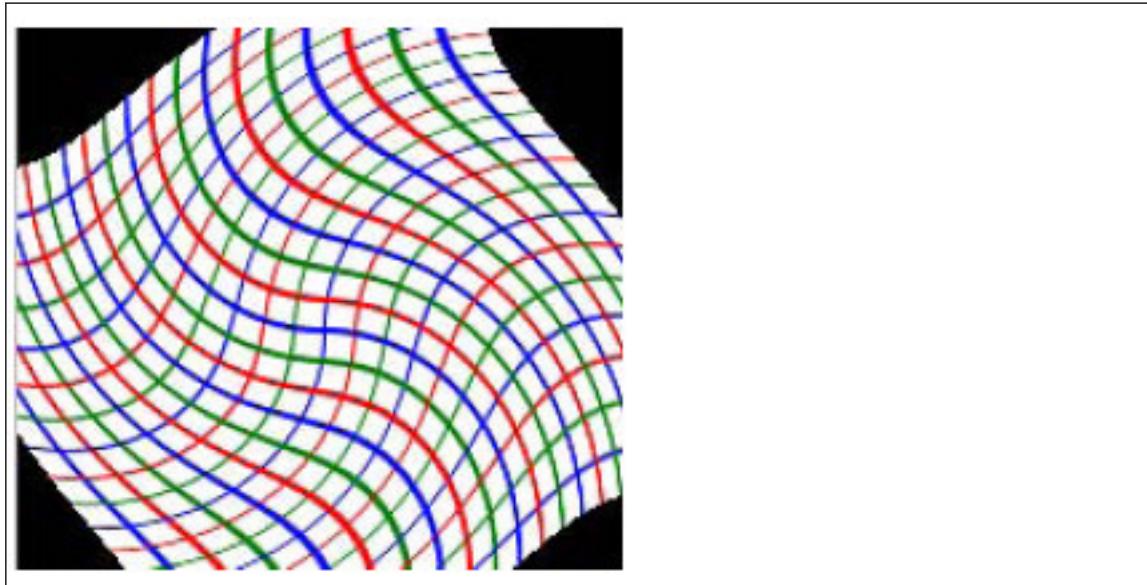
*pxMap[i, j]* contains the *x*- coordinates of the source image pixels *src\_pixel* that are written to *dst\_pixel*;

*pyMap[i, j]* contains the *y*- coordinates of the source image pixels *src\_pixel* that are written to *dst\_pixel*.

If the referenced coordinates correspond to a pixel outside of the source ROI, and the flag IPPI\_SMOOTH\_EDGE is not set, then no mapping of the source pixel is performed.

Figure “Remapping the Sample Image” gives an example of applying the function `ippiRemap` to a sample image that is a square grid of alternating blue, red, and green lines.

### Remapping the Sample Image



The transformed part of the image is resampled using the [interpolation method](#) specified by the `interpolation` parameter, and is written to the destination image ROI. The function can be used with or without edge smoothing. The pseudo code below shows how it works.

The function works without edge smoothing - the flag `IPPI_SMOOTH_EDGE` is not set:

```
if ( xMap < srcRoi . x || xMap > srcRoi . x + srcRoi . width -1 || yMap < srcRoi . y ||  
yMap > srcRoi . y + srcRoi . height -1)  
    not fill dst /* do not remap */  
else  
    fill dst with Interpolate(Src, xMap, yMap) /* remap */
```

The function works with edge smoothing - the flag `IPPI_SMOOTH_EDGE` is set:

```
if (xMap < srcRoi.x - 1 || xMap > srcRoi.x+srcRoi.width || yMap < srcRoi.y - 1 || yMap  
> srcRoi.y+srcRoi.height)  
    not fill dst /* do not remap */  
else if (xMap < srcRoi.x || xMap > srcRoi.x+srcRoi.width-1 || yMap < srcRoi.y || yMap >  
srcRoi.y+srcRoi.height-1)  
    fill dst with Interpolate(Src, fillvalue, xMap, yMap) /* smoothing */  
else  
    fill dst with Interpolate(Src, xMap, yMap) /* remap */
```

### Return Values

`ippStsNoErr`

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the <i>srcStep</i> , <i>dstStep</i> , <i>xMapStep</i> , or <i>yMapStep</i> has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if <i>interpolation</i> has an illegal value.
ippStsWrongIntersectROIErr	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

## Example

To better understand usage of the `ippiRemap` function, refer to the `Remap.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

# Miscellaneous Image Transforms

13

This section describes the Intel® IPP image processing functions that perform adaptive thresholding of the image ROI.

## ThresholdAdaptiveBoxGetBufferSize

*Computes the size of the work buffer for adaptive thresholding with the Box method.*

### Syntax

```
IppStatusippiThresholdAdaptiveBoxGetBufferSize(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

### Parameters

<i>roiSize</i>	Size of the destination image ROI, in pixels.
<i>maskSize</i>	Size of the kernel that is used to calculate a threshold level. Width and height of <i>maskSize</i> must be equal and odd.
<i>dataType</i>	Data type of the source and destination images. Possible value is <code>ipp8u</code> .
<i>numChannels</i>	Number of channels in the images. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external work buffer.

### Description

This function computes the size, in bytes, of the external work buffer needed for the `ThresholdAdaptiveBox` function. The result is stored in *pBufferSize*.

For an example on how to use this function, refer to the example provided with the `ThresholdAdaptiveBox` function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>maskSize</i> has a field with a zero or negative value, or fields of <i>maskSize</i> are not equal.

- `ippStsDataTypeErr` Indicates an error when `dataType` has an illegal value.  
`ippStsNumChannelsErr` Indicates an error when `numChannels` has an illegal value.

## See Also

[ThresholdAdaptiveBox](#) Performs adaptive thresholding with the Box method.

## ThresholdAdaptiveBox

---

*Performs adaptive thresholding with the Box method.*

### Syntax

```
IppStatusippiThresholdAdaptiveBox_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u
valLE, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatusippiThresholdAdaptiveBox_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType
borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination image ROI (for the in-place function).
<code>srcDstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).
<code>roiSize</code>	Size of the destination image ROI, in pixels.
<code>maskSize</code>	Size of the kernel that is used to calculate a threshold level. Width and height of <code>maskSize</code> must be equal and odd.
<code>delta</code>	Value for threshold calculation.
<code>valGT</code>	Output pixel if the source pixel value is more than threshold.
<code>valLE</code>	Output pixel if the source pixel value is less than, or equal to threshold.
<code>borderType</code>	Type of border. Possible values are:

---

ippBorderConst	Values of all border pixels are set to a constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderRepl` or `ippBorderConst` and the following flags:

- `ippBorderInMemTop`
- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

*borderValue*

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

*pBuffer*

Pointer to the work buffer. To calculate the size of the temporary buffer, use the [ThresholdAdaptiveBoxGetBufferSize](#) function.

## Description

This function performs adaptive thresholding of the source image ROI using the Box method. Output pixels are calculated according to the following formulas:

$$pDst(x, y) = valGT, \text{ if } pSrc(x, y) > T(x, y)$$

$$pDst(x, y) = valLE, \text{ if } pSrc(x, y) \leq T(x, y)$$

where

$T(x, y)$  is a mean of the `maskSize.width*maskSize.height` neighborhood of a  $(x, y)$  pixel minus  $\delta$ .

Before using this function, compute the size of the external work buffer using the [ThresholdAdaptiveBoxGetBufferSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> , <code>pDst</code> , <code>pSrcDst</code> , or <code>pBuffer</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>maskSize</code> has a field with a zero or negative value, or fields of <code>maskSize</code> are not equal.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiThresholdAdaptiveBox_8u_C1R` and `ThresholdAdaptiveBoxGetBufferSize` functions.

```
IppStatus threshold_adaptive_box_8u_c1( void ) {
    Ipp8u pSrc[8*8] =
    {
        0, 255, 1, 254, 2, 253, 3, 252,
        251, 4, 250, 5, 249, 6, 248, 7,
        8, 247, 9, 246, 10, 245, 11, 244,
        243, 12, 242, 13, 241, 14, 240, 15,
        16, 239, 17, 238, 18, 237, 19, 236,
        235, 20, 234, 21, 233, 22, 232, 23,
        24, 231, 25, 230, 26, 229, 26, 228,
        227, 27, 226, 28, 225, 29, 224, 30
    };
    Ipp8u pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int srcStep = 8 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp8u);
    int bufferSize;
    IppStatus status;
    Ipp32f delta = 0.5f;
    Ipp8u valGT = 254;
    Ipp8u valLE = 1;
    Ipp8u *pBuffer;

    ippithresholdadaptiveboxgetbuffersize(roiSize, maskSize, ipp8u, 1, &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippithresholdadaptivebox_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, maskSize, delta,
valGT, valLE, borderType, 33, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

`pDst` after function execution:

1	254	1	254	1	254	1	254
254	1	254	1	254	1	254	1
1	254	1	254	1	254	1	254
254	1	254	1	254	1	254	1
1	254	1	254	1	254	1	254
254	1	254	1	254	1	254	1
1	254	1	254	1	254	1	254
254	1	254	1	254	1	254	1

## See Also

[ThresholdAdaptiveBoxGetBufferSize](#) Computes the size of the work buffer for adaptive thresholding with the Box method.

## ThresholdAdaptiveGaussGetBufferSize

*Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.*

## Syntax

```
IppStatusippiThresholdAdaptiveGaussGetBufferSize(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>roiSize</i>	Size of the destination image ROI, in pixels.
<i>maskSize</i>	Size of the kernel that is used to calculate a threshold level. Width and height of <i>maskSize</i> must be equal and odd.
<i>dataType</i>	Data type of the source and destination images. Possible value is <code>ipp8u</code> .
<i>numChannels</i>	Number of channels in the images. Possible value is 1.
<i>pSpecSize</i>	Pointer to the size of the adaptive threshold specification structure.
<i>pBufferSize</i>	Pointer to the size (in bytes) of the external work buffer.

## Description

This function computes the size, in bytes, of the adaptive threshold specification structure and the size of the external work buffer needed for the `ThresholdAdaptiveGauss` function. The results are stored in *pSpecSize* and *pBufferSize*.

For an example on how to use this function, refer to the example provided with the `ThresholdAdaptiveGauss` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSpecSize</i> or <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>maskSize</i> has a field with a zero or negative value, or fields of <i>maskSize</i> are not equal.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

`ThresholdAdaptiveGauss` Performs adaptive thresholding with the Gaussian method.

## ThresholdAdaptiveGaussInit

*Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.*

### Syntax

```
IppStatusippiThresholdAdaptiveGaussInit(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, Ipp32f sigma, IppiThresholdAdaptiveSpec* pSpec);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>roiSize</i>	Size of the destination image ROI, in pixels.
<i>maskSize</i>	Size of the kernel that is used to calculate a threshold value. Width and height of <i>maskSize</i> must be equal and odd.
<i>dataType</i>	Data type of the source and destination images. Possible value is ipp8u.
<i>numChannels</i>	Number of channels in the images. Possible value is 1.
<i>sigma</i>	Value of sigma that is used to calculate a threshold value for the Gaussian method. If sigma value is less than, or equal to zero, <i>sigma</i> is set automatically in compliance with the kernel size.
<i>pSpec</i>	Pointer to the adaptive threshold specification structure.

### Description

This function initializes the adaptive threshold specification structure *pSpec* for adaptive thresholding with the Gaussian method. Before using this function, compute the size of the specification structure using the [ThresholdAdaptiveGaussGetSize](#) function.

If sigma is less than, or equal to zero, it is set according to the following formula:

$$\text{sigma} = 0.3 * ((\text{maskSize.width} - 1) * 0.5 - 1) + 0.8$$

For an example on how to use this function, refer to the example provided with the [ThresholdAdaptiveGauss](#) function description.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error when <i>pSpecSize</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error when <i>dstRoiSize</i> has a field with a zero or negative value.
<i>ippStsMaskSizeErr</i>	Indicates an error when <i>maskSize</i> has a field with a zero or negative value, or fields of <i>maskSize</i> are not equal.

---

ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[ThresholdAdaptiveGaussGetSize](#) Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.  
[ThresholdAdaptiveGauss](#) Performs adaptive thresholding with the Gaussian method.

# ThresholdAdaptiveGauss

---

*Performs adaptive thresholding with the Gaussian method.*

## Syntax

```
IppStatusippiThresholdAdaptiveGauss_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
pDst, int dstStep, IppiSize roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE,  
IppiBorderType borderType, Ipp8u borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u*  
pBuffer);  
  
IppStatusippiThresholdAdaptiveGauss_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u  
borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place function).
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).
<i>roiSize</i>	Size of the destination image ROI, in pixels.
<i>delta</i>	Value for threshold calculation.
<i>valGT</i>	Output pixel if the source pixel value is more than threshold.
<i>valLE</i>	Output pixel if the source pixel value is less than, or equal to threshold.

*borderType*

Type of border. Possible values are:

<code>ippBorderConst</code>	Values of all border pixels are set to a constant.
<code>ippBorderRepl</code>	Border is replicated from the edge pixels.
<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderRepl` or `ippBorderConst` and the following flags:

- `ippBorderInMemTop`
- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

*borderValue*

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

*pSpec*

Pointer to the adaptive threshold specification structure.

*pBuffer*

Pointer to the work buffer. To calculate the size of the temporary buffer, use the [ThresholdAdaptiveGaussGetBufferSize](#) function.

## Description

This function performs adaptive thresholding of the source image ROI using the Gaussian method. Output pixels are calculated according to the following formulas:

$$pDst(x, y) = valGT \text{ if } pSrc(x, y) > T(x, y)$$

$$pDst(x, y) = valLE \text{ if } pSrc(x, y) \leq T(x, y)$$

where

$T(x, y)$  is a weighted sum (cross-correlation with a Gaussian window) of the `maskSize.width*maskSize.height` neighborhood of a  $(x, y)$  pixel minus `delta`.

The function uses a separable Gaussian filter. Filter coefficients are computed according to the following formula:

$$G_i = A * \exp(-((i - (maskSize.width-1)/2)^2)/(0.5 * sigma^2))$$

where

$A$  is a scale factor for  $\sum G_i = 1$  ( $i = 0, \dots, maskSize.width-1$ )

Before using this function, compute the size of the external work buffer and specification structure using the [ThresholdAdaptiveGaussGetBufferSize](#) function, and initialize the structure using the [ThresholdAdaptiveGaussInit](#) function.

## Return Values

`ippStsNoErr`

Indicates no error.

`ippStsNullPtrErr`

Indicates an error when `pSrc`, `pDst`, `pSrcDst`, `pSpec`, or `pBuffer` is `NULL`.

---

ippStsSizeErr	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
ippStsContextMatchErr	Indicates an error when <i>pSpec</i> does not match.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.

## Example

The code example below demonstrates how to use the `ippiThresholdAdaptiveGauss_8u_C1R`, `ThresholdAdaptiveGaussGetBufferSize`, and `ThresholdAdaptiveGaussInit` functions.

```
IppStatus threshold_adaptive_gauss_8u_c1( void ) {
    Ipp8u pSrc[8*8] =
    {
        0, 255,   1, 254,   2, 253,   3, 252,
        251,   4, 250,   5, 249,   6, 248,   7,
        8, 247,   9, 246,  10, 245,  11, 244,
       243,  12, 242,  13, 241,  14, 240,  15,
       16, 239,  17, 238,  18, 237,  19, 236,
       235,  20, 234,  21, 233,  22, 232,  23,
       24, 231,  25, 230,  26, 229,  26, 228,
       227,  27, 226,  28, 225,  29, 224,  30
    };
    Ipp8u     pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int      srcStep = 8 * sizeof(Ipp8u);
    int      dstStep = 8 * sizeof(Ipp8u);
    int      bufferSize;
    int      specSize;
    IppStatus status;
    Ipp32f   sigma = 10.0f;
    Ipp32f   delta = 0.5f;
    Ipp8u   valGT = 254;
    Ipp8u   valLE = 1;
    IppiThresholdAdaptiveSpec *pSpec;
    Ipp8u   *pBuffer;

    ippiThresholdAdaptiveGaussGetBufferSize(roiSize, maskSize, ipp8u, 1, &specSize, &bufferSize);
    pSpec = (IppiThresholdAdaptiveSpec *)ippsMalloc_8u(specSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippiThresholdAdaptiveGaussInit(roiSize, maskSize, ipp8u, 1, sigma, pSpec);
    ippiThresholdAdaptiveGauss_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, delta, valGT,
valLE, borderType, 33, pSpec, pBuffer);
    ippsFree(pBuffer);
    ippsFree(pSpec);

    return status;
}
```

*pDst* after function execution:

1 254	1 254	1 254	1 254	1 254
254	1 254	1 254	1 254	1
1 254	1 254	1 254	1 254	1 254
254	1 254	1 254	1 254	1
1 254	1 254	1 254	1 254	1 254

```
254 1 254 1 254 1  
1 254 1 254 1 254 1  
254 1 254 1 254 1
```

## See Also

[ThresholdAdaptiveGaussGetSize](#) Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

[ThresholdAdaptiveGaussInit](#) Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.

# Wavelet Transforms

# 14

This chapter describes the Intel® IPP image processing functions that perform two-dimensional discrete wavelet transform (DWT).

In many applications the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

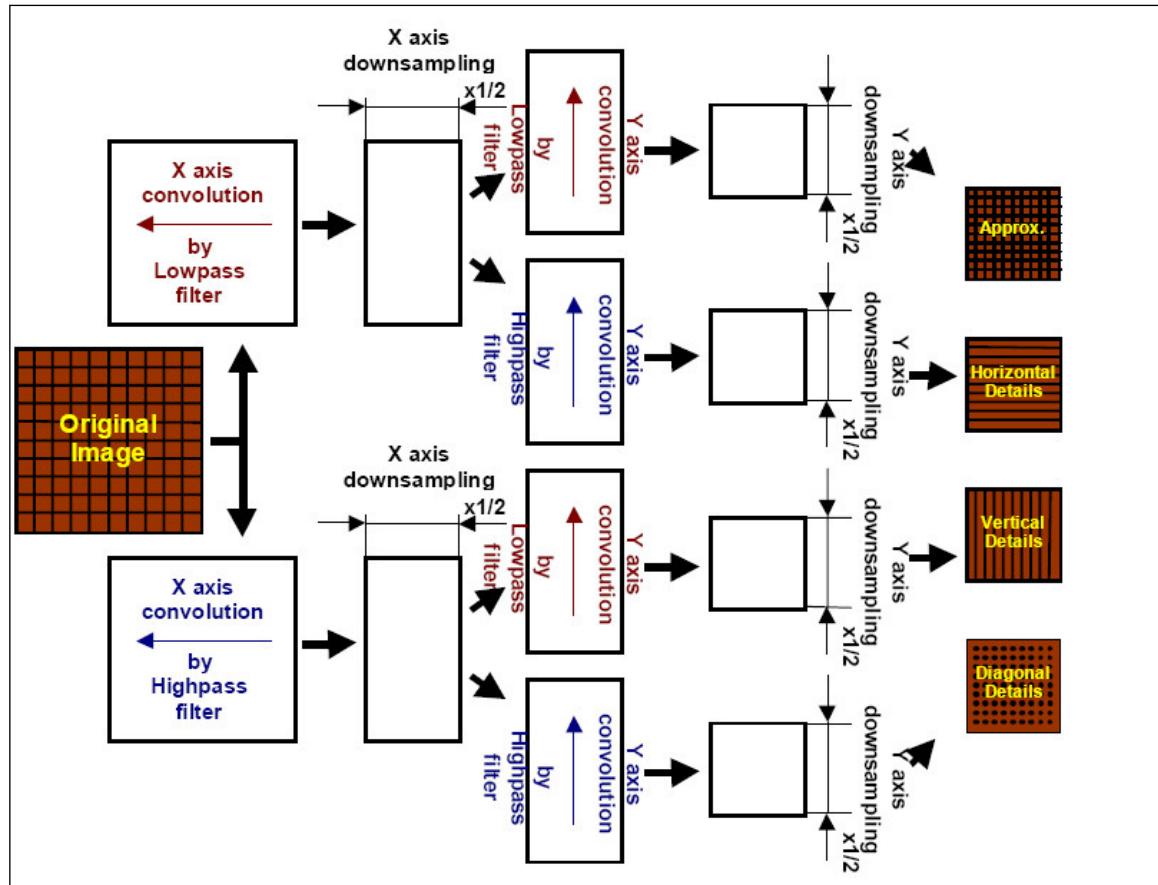
Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure.

The wavelet transform type can be set by specifying the appropriate filter taps in the initialization function.

Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP functions for wavelet decomposition and reconstruction use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. [Figure "Equivalent Scheme of Wavelet Decomposition Algorithm"](#) shows the equivalent algorithm of wavelet-based image decomposition:

## Equivalent Scheme of Wavelet Decomposition Algorithm



Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.

These decomposition components have the following meaning:

- The 'approximation' image is obtained by vertical and horizontal lowpass filtering.
- The 'horizontal detail' image is obtained by vertical highpass and horizontal lowpass filtering.
- The 'vertical detail' image is obtained by vertical lowpass and horizontal highpass filtering.
- The 'diagonal detail' image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this document for identification convenience only.

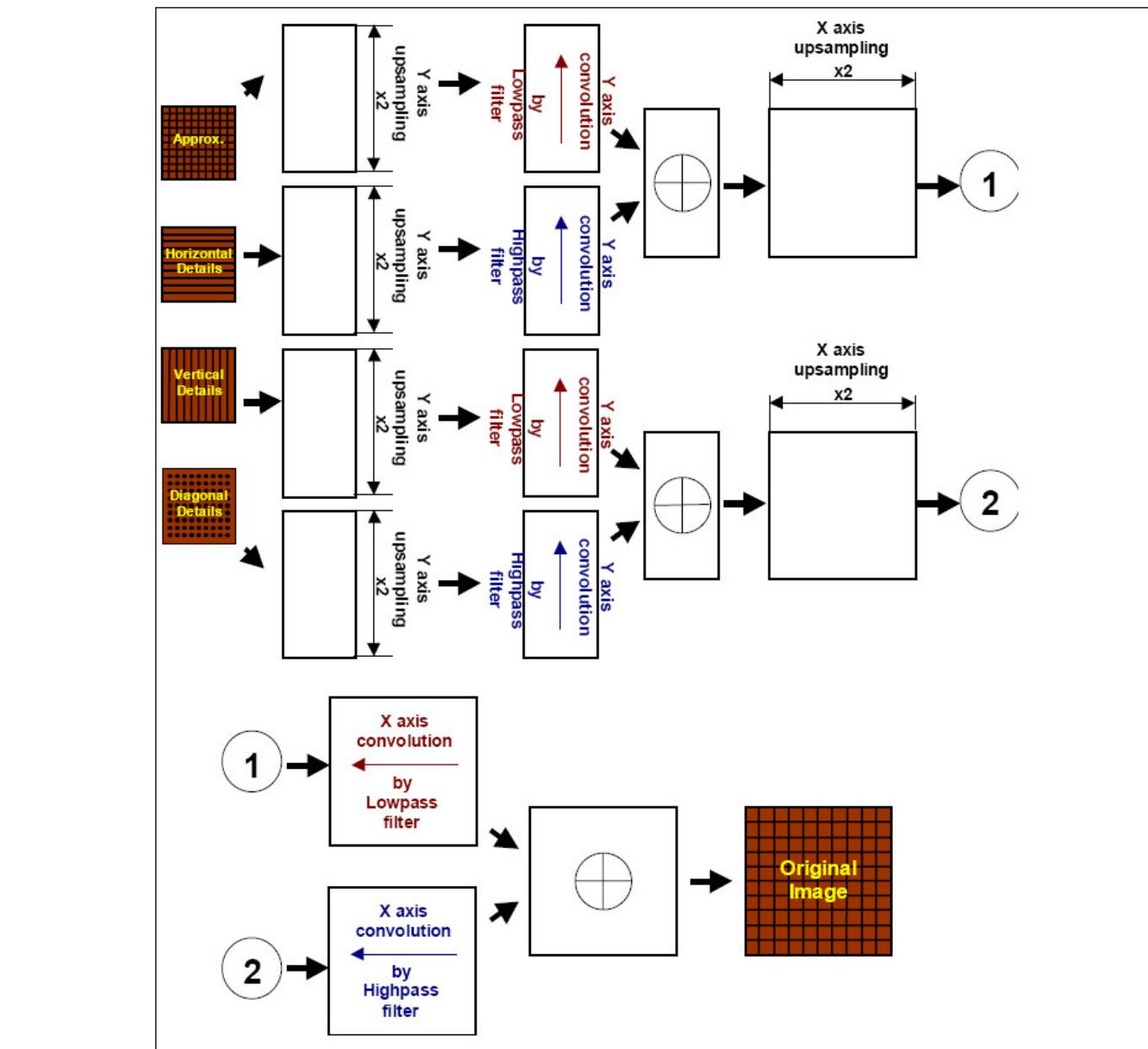
The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

[Figure "Equivalent Scheme of Wavelet Reconstruction Algorithm"](#) shows the equivalent algorithm of wavelet reconstruction of an image.

Wavelet transform functions support regions of interest (ROI, see [Regions of Interest in Intel IPP in chapter 2](#)) in the images. However, these functions do not perform internally any border extensions of image ROI data. It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of the functions `ippiWTFwd` and `ippiWTInv` for detailed information on how to calculate extended image border sizes.

### Equivalent Scheme of Wavelet Reconstruction Algorithm



## WTFwdGetSize

Calculates the size of the specification structure and work buffer for a forward wavelet transform.

## Syntax

```
IppStatusippiWTFwdGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh,
int anchorHigh, int* pSpecSize, int* pBufSize);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>numChannels</i>	Number of channels in the image. Possible values are 1 or 3.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.
<i>pSpecSize</i>	Pointer to the computed size of the <code>ippiWTFwd</code> specification structure, in bytes.
<i>pBufSize</i>	Pointer to the computed size of the work buffer, in bytes.

## Description

This function computes the size, in bytes, of the specification structure and work buffer required for the forward wavelet transform function `ippiWTFwd`.

For an example on how to use this function, refer to the example provided with the `ippiWTInv` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsNumChannlesErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.
<code>ippStsNumChannlesErr</code>	Indicates an error when <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error when <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.

## See Also

[WTFwd](#) Performs one-level wavelet decomposition of an image.

[WTInv](#) Performs one-level wavelet reconstruction of an image.

## WTFwdInit

---

*Initializes the forward wavelet transform context structure.*

## Syntax

```
IppStatusippiWTFwdInit_32f_C1R(IppiWTFwdSpec_32f_C1R* pSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

```
IppStatusippiWTFwdInit_32f_C3R(IppiWTFwdSpec_32f_C3R* pSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSpec</i>	Double pointer to the forward wavelet transform specification structure.
<i>pTapsLow</i>	Pointer to lowpass filter taps.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>pTapsHigh</i>	Pointer to highpass filter taps.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.

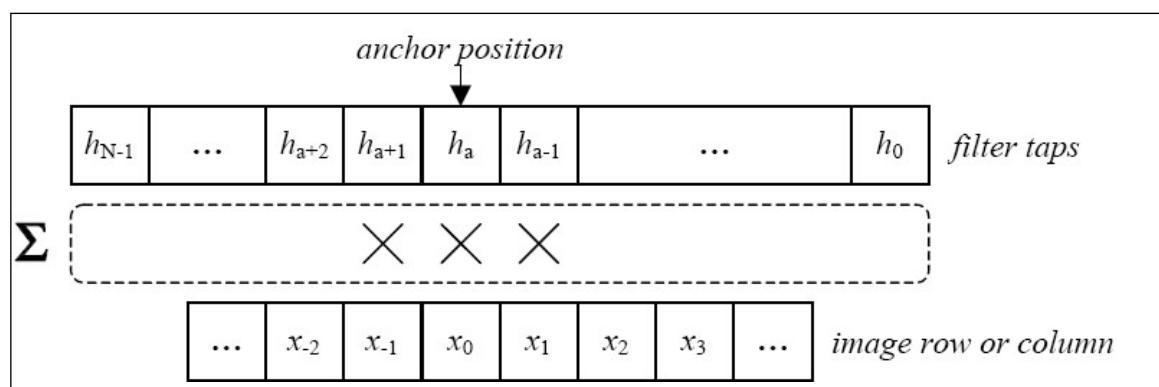
## Description

This function initializes the specification structure *pSpec* for a one-level wavelet decomposition.

The forward wavelet transform specification structure contains parameters of a wavelet filter bank used for image decomposition. The filter bank consists of two analysis filters and includes the lowpass decomposition filter (or *coarse* filter) and the highpass decomposition filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

### Anchor Value and Initial Filter Position for Wavelet Decomposition



Here  $a$  stands for anchor value,  $N$  is filter length,  $x_0$  is the starting pixel of the processed row or column, and  $x_{-1}, x_{-2}, \dots$  are the additional border pixels that are needed for calculations. The anchor value and filter length completely determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of `ippiWTFwd` function.

For an example on how to use this function, refer to the example provided with the `ippiWTInv` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when filter length <code>lenLow</code> or <code>lenHigh</code> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error when anchor position <code>anchorLow</code> or <code>anchorHigh</code> is less than zero.

## See Also

`WTFwdGetSize` Calculates the size of the specification structure and work buffer for a forward wavelet transform.

`WTFwd` Performs one-level wavelet decomposition of an image.

`WTInv` Performs one-level wavelet reconstruction of an image.

## WTFwd

---

Performs one-level wavelet decomposition of an image.

---

### Syntax

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetail1XDst, int detail1XStep, Ipp32f* pDetail1YDst, int detail1YStep, Ipp32f* pDetail1XYDst, int detail1XYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R* pSpec, Ipp8u* pBuffer);

IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetail1XDst, int detail1XStep, Ipp32f* pDetail1YDst, int detail1YStep, Ipp32f* pDetail1XYDst, int detail1XYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

### Include Files

`ippi.h`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
<code>pApproxDst</code>	Pointer to ROI of the destination approximation image.

<i>approxStep</i>	Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.
<i>pDetail1xDst</i>	Pointer to ROI of the destination horizontal detail image.
<i>detail1xStep</i>	Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.
<i>pDetail1yDst</i>	Pointer to ROI of the destination vertical detail image.
<i>detail1yStep</i>	Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.
<i>pDetail1xyDst</i>	Pointer to ROI of the destination diagonal detail image.
<i>detail1xyStep</i>	Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.
<i>dstRoiSize</i>	Size of the ROI in pixels for all destination images.
<i>pSpec</i>	Pointer to the allocated and initialized forward DWT specification structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs one-level wavelet decomposition of a source image pointed to by *pSrc* into four destination subimages pointed to by *pApproxDst*, *pDetail1xDst*, *pDetail1yDst*, and *pDetail1xyDst*. See [Figure "Equivalent Scheme of Wavelet Decomposition Algorithm"](#) for the equivalent algorithm of `ippiWTFwd` function operation.

Wavelet parameters are contained in the forward transform specification structure *pSpec*. Before using this function, compute the size of the structure and work buffer using the [WTFwdGetSize](#) function and initialize the structure using [WTFwdInit](#).

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a> .
Notice revision #20201201

The pointer *pSrc* points to memory location of the source image rectangular ROI of size *srcWidth* by *srcHeight* which is uniquely determined by the size of destination ROI as:

$$\text{srcWidth} = 2 * \text{dstRoiSize.width}$$

$$\text{srcHeight} = 2 * \text{dstRoiSize.height}$$

The source image ROI size always has even dimensions, as it is computed from the *dstRoiSize* parameter as follows:

$$\text{srcRoiSize.width} = 2 * \text{dstRoiSize.width}$$

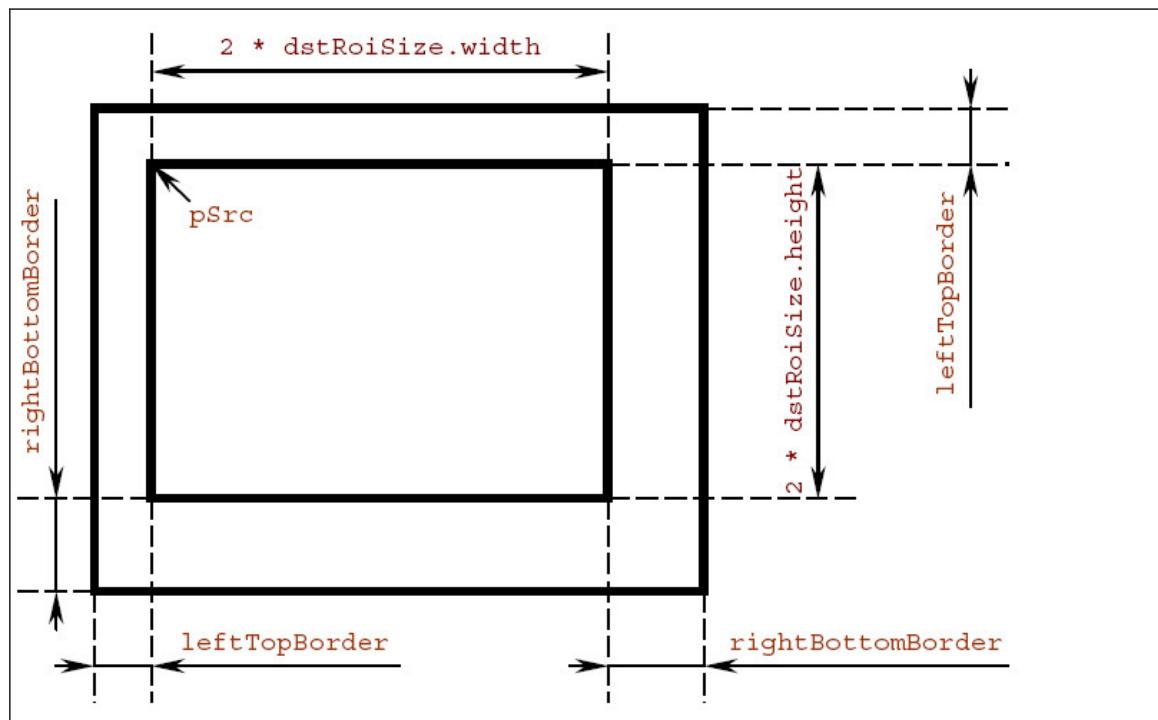
$$\text{srcRoiSize.height} = 2 * \text{dstRoiSize.height}$$

To use this function for images with uneven width or height, you should truncate the last column/row or extend an image to even dimensions.

**NOTE**

The source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using `ippiWTFwd` function the application program must apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations. As a result, the size of memory block allocated for the source image must be extended to accommodate for added border pixels outside ROI formal boundaries.

Figure “Extended Source Image for Wavelet Decomposition” schematically shows the source image ROI and extended image area.

**Extended Source Image for Wavelet Decomposition**

Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow    = lenLow    - 1 - anchorLow;
int leftBorderHigh   = lenHigh   - 1 - anchorHigh;
int rightBorderLow   = anchorLow;
int rightBorderHigh  = anchorHigh;
int leftTopBorder    = IPP_MAX(leftBorderLow, leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of the function `WTFwdInit` for the explanation of the parameters.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```
srcWidthWithBorders = srcWidth + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

All destination images have equal size specified by the parameter `dstRoiSize`.

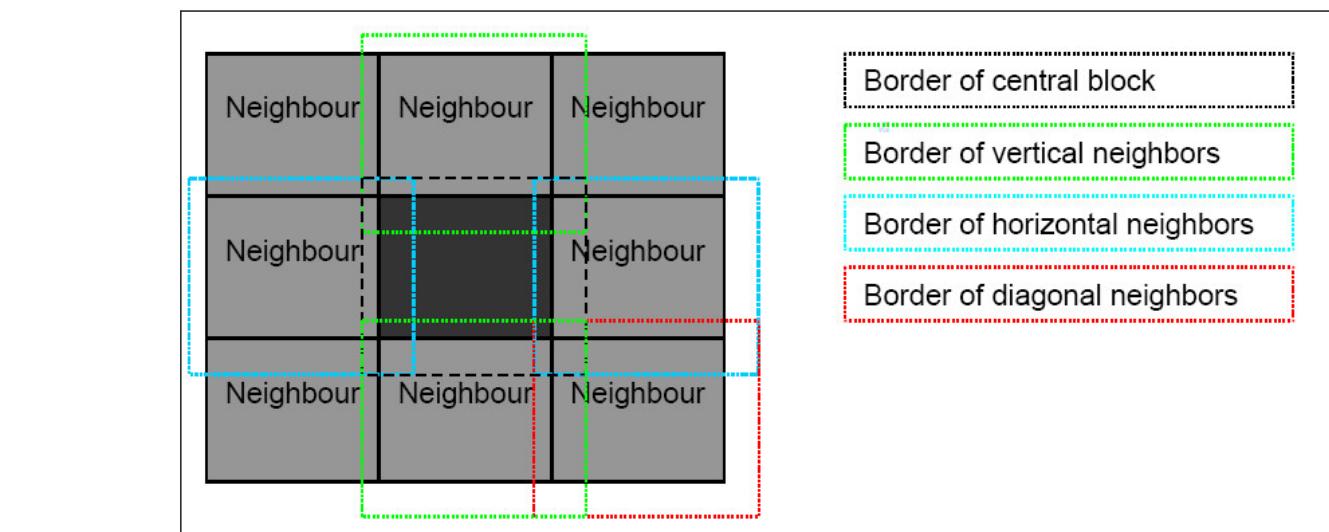
Conversely, to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, use extended component images for the reconstruction pass. See the description of the function [ippiWTInv](#) for more details.

The ROI concept used in wavelet transform functions can be applied to processing large images by blocks, or 'tiles'. To accomplish this, the source image should be subdivided into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor's ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in [Figure "Image Division into Blocks with Overlapping Borders"](#).

### Image Division into Blocks with Overlapping Borders



For an example on how to use this function, refer to the example provided with the [WTInv](#) function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if step through any buffer is less than or equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid specification structure is passed.

### See Also

[WTFwdGetSize](#) Calculates the size of the specification structure and work buffer for a forward wavelet transform.

[WTFwdInit](#) Initializes the forward wavelet transform context structure.

[WTInv](#) Performs one-level wavelet reconstruction of an image.

## WTFwdGetSize

*Calculates the size of the specification structure and work buffer for an inverse wavelet transform.*

### Syntax

```
IppStatusippiWTInvGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh,
int anchorHigh, int* pSpecSize, int* pBufSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>numChannels</i>	Number of channels in the image. Possible values are 1 or 3.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.
<i>pSpecSize</i>	Pointer to the computed size of the <code>ippiWTInv</code> specification structure, in bytes.
<i>pBufSize</i>	Pointer to the computed size of the work buffer, in bytes.

### Description

This function computes the size, in bytes, of the specification structure and work buffer required for the inverse wavelet transform function `ippiWTInv`.

For an example on how to use this function, refer to the example provided with the `ippiWTInv` function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsNumChannlesErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.
<code>ippStsNumChannlesErr</code>	Indicates an error when <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error when <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.

### See Also

`WTInv` Performs one-level wavelet reconstruction of an image.

## WTInvInit

*Initializes the inverse wavelet transform specification structure.*

---

### Syntax

```
IppStatusippiWTInvInit_32f_C1R(IppiWTInvSpec_32f_C1R* pSpec, const Ipp32f* pTapsLow,  
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

```
IppStatusippiWTInvInit_32f_C3R(IppiWTInvSpec_32f_C3R* pSpec, const Ipp32f* pTapsLow,  
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSpec</i>	Double pointer to a new allocated and initialized inverse DWT context structure.
<i>pTapsLow</i>	Pointer to lowpass filter taps.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>pTapsHigh</i>	Pointer to highpass filter taps.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.

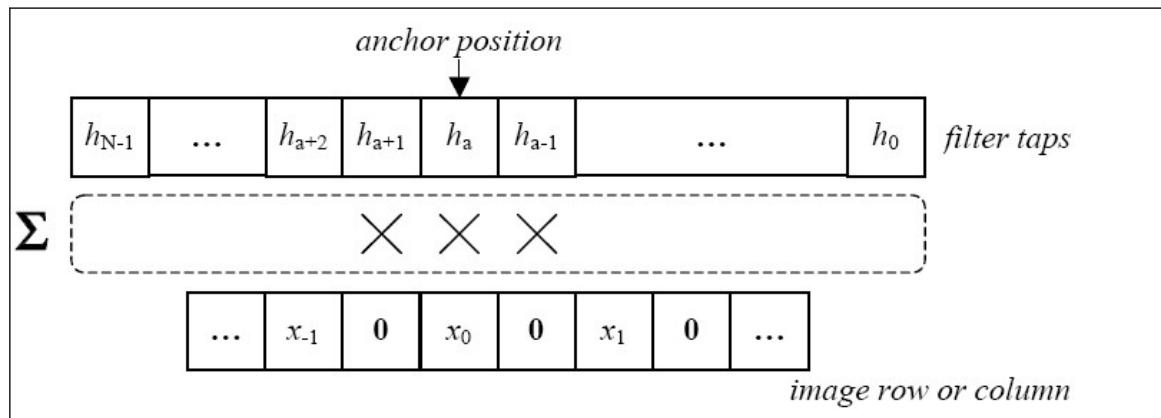
### Description

This function initializes the specification structure *pSpec* for a one-level wavelet reconstruction.

The inverse wavelet transform specification structure contains parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the lowpass reconstruction filter (or *coarse* filter) and the highpass reconstruction filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

### Anchor Value and Initial Filter Position for Wavelet Reconstruction



Here  $a$  stands for anchor value,  $N$  is filter length,  $x_0$  is the starting pixel of the processed row or column, and  $x_{-1}$ ,  $x_{-2}$ , ... are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of the [ippWTInv](#) function.

For an example on how to use this function, refer to the example provided with the [ippiWTInv](#) function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when filter length <code>lenLow</code> or <code>lenHigh</code> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error when anchor position <code>anchorLow</code> or <code>anchorHigh</code> is less than zero.

### See Also

[WTInvGetSize](#) Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

[WTInv](#) Performs one-level wavelet reconstruction of an image.

## WTInv

Performs one-level wavelet reconstruction of an image.

## Syntax

```
IppStatusippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C1R* pSpec,Ipp8u* pBuffer);

IppStatusippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C3R* pSpec,Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pApproxSrc</i>	Pointer to ROI of the source approximation image.
<i>approxStep</i>	Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.
<i>pDetailXSrc</i>	Pointer to ROI of the source horizontal detail image.
<i>detailXStep</i>	Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.
<i>pDetailYSrc</i>	Pointer to ROI of the source vertical detail image.
<i>detailYStep</i>	Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.
<i>pDetailXYSrc</i>	Pointer to ROI of the source diagonal detail image.
<i>detailXYStep</i>	Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.
<i>srcRoiSize</i>	Size of ROI in pixels for all source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
<i>pSpec</i>	Pointer to the allocated and initialized inverse DWT specification structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations.

## Description

This function operates with ROI (see Regions of Interest in Intel IPP ). This function performs wavelet reconstruction of the output image *pDst* from the four component images. See [Figure "Image Division into Blocks with Overlapping Borders"](#) for the equivalent algorithm of `ippiWTInv` function operation. Wavelet parameters are contained in the inverse transform specification structure *pSpec*. Before using this function, compute the size of the structure and work buffer using the `ippiWTInvGetSize` function and initialize the structure using `ippiWTInvInit`.

## Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

The pointers `pApproxSrc`, `pDetailXSrc`, `pDetailYSrc`, and `pDetailXYSrc` point to ROIs of source images excluding borders. All source ROIs have the same size `srcRoiSize`, while the destination image size is uniquely determined from the following relations:

```
dstWidth = 2 * srcRoiSize.width; dstHeight = 2 * srcRoiSize.height;
```

As source ROIs do not include border pixels required to computations, the application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images filling the neighboring memory locations. Note the border sizes may be different for different source images. The following C-language expressions can be used to calculate extended image border sizes:

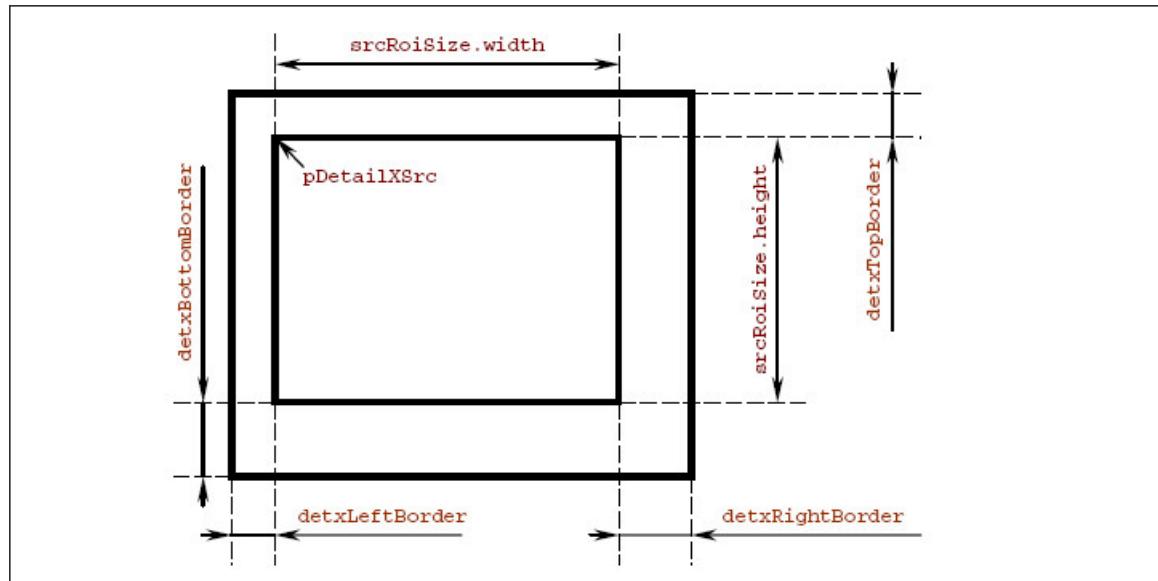
```
int leftBorderLow      = (lenLow - 1 - anchorLow) / 2;
int leftBorderHigh     = (lenHigh - 1 - anchorHigh) / 2;
int rightBorderLow     = (anchorLow + 1) / 2;
int rightBorderHigh    = (anchorHigh + 1) / 2;
int apprLeftBorder     = leftBorderLow;
int apprRightBorder    = rightBorderLow;
int apprTopBorder       = leftBorderLow;
int apprBottomBorder   = rightBorderLow;
int detxLeftBorder     = leftBorderLow;
int detxRightBorder    = rightBorderLow;
int detxTopBorder       = leftBorderHigh;
int detxBottomBorder   = rightBorderHigh;
int detyLeftBorder     = leftBorderHigh;
int detyRightBorder    = rightBorderHigh;
int detyTopBorder       = leftBorderLow;
int detyBottomBorder   = rightBorderLow;
int detxyLeftBorder    = leftBorderHigh;
int detxyRightBorder   = rightBorderHigh;
int detxyTopBorder     = leftBorderHigh;
int detxyBottomBorder  = rightBorderHigh;
```

See the description of the function `ippiWTInvInit` for the explanation of the used parameters.

The above relations show that left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image must be extended to accommodate for added border pixels outside ROI.

Figure "Extended Horizontal Detail Source Image for Wavelet Reconstruction" shows ROI and extended image area for the horizontal detail source image.

### Extended Horizontal Detail Source Image for Wavelet Reconstruction



Sizes of source images extended by border pixels can be calculated as follows:

```

apprWidthWithBorders = srcWidth + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder + apprBottomBorder;
detxWidthWithBorders = srcWidth + detxLeftBorder + detxRightBorder;
detxHeightWithBorders = srcHeight + detxTopBorder + detxBottomBorder;
detyWidthWithBorders = srcWidth + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder + detyBottomBorder;
detxyWidthWithBorders = srcWidth + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder + detxyBottomBorder;

```

The ROI concept can be used to reconstruct large images by blocks or 'tiles'.

To accomplish this, each the source images into blocks with overlapping borders, similar to what is considered in the description of the function [ippiWTFwd](#). Each component must be subdivided into the same pattern of rectangular blocks.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is <i>NULL</i> .
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if step through any buffer is less than or equal to zero.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid specification structure is passed.

## Example

The example below shows how to use the function *ippiWTInv\_32f\_C1R*.

```
void func_wavelet()
{
    IppiWTFwdSpec_32f_C1R* pSpecFwd;
    IppiWTInvSpec_32f_C1R* pSpecInv;
    int specSizeFwd, specSizeInv;
    Ipp32f pTapsLow[3] = {0.25, 0.5, 0.25};
    int lenLow = 3;
    int anchorLow = 1;
    Ipp32f pTapsHigh[3] = { 0.75, -0.25, -0.125};
    int lenHigh = 3;
    int anchorHigh = 1;
    Ipp32f pSrc[8*8] = { 0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
                         0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
                         0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
                         11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                         11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                         0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
                         0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
                         0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0};

    Ipp32f pSrcB[9*9];
    int srcStepB = 9*sizeof(Ipp32f);
    IppiSize roiSizeB = {9, 9};
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};
    Ipp32f pDetailXDst[4*4];
    Ipp32f pDetailYDst[4*4];
    Ipp32f pDetailXYDst[4*4];
    Ipp32f pApproxDst[4*4];
    IppiSize dstRoiSize = {4, 4};
    int bufSizeFwd, bufSizeInv;
    Ipp8u* pBufferFwd;
    Ipp8u* pBufferInv;
    Ipp32f pDstInv[8*8];
    Ipp32f pAppB[5*5];
    Ipp32f pXB[5*5];
    Ipp32f pYB[5*5];
    Ipp32f pXYB[5*5];
    int StepB = 5*sizeof(Ipp32f);
    IppiSize roiInvSize = {4, 4};
    IppiSize roiInvSizeB = {5, 5};
    int stepDstInv = 8*sizeof(Ipp32f);
    int approxStep, detailXStep, detailYStep, detailXYStep;
    approxStep = detailXStep = detailYStep = detailXYStep = 4*sizeof(Ipp32f);
    //adds border to the source image
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pSrc, srcStep, roiSize, (Ipp32s*)pSrcB, srcStepB,
    roiSizeB, 1, 1);
    //performs forward wavelet transform
    ippiWTFwdGetSize_32f(1, lenLow, anchorLow, lenHigh, anchorHigh, &specSizeFwd, &bufSizeFwd);
    pSpecFwd = (IppiWTFwdSpec_32f_C1R*)ippMalloc(specSizeFwd);
    pBufferFwd = (Ipp8u*)ippMalloc(bufSizeFwd);
    ippiWTFwdInit_32f_C1R( pSpecFwd, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh,
    anchorHigh);
    ippiWTFwd_32f_C1R (pSrcB + roiSizeB.width + 1, srcStepB, pApproxDst, approxStep,
    pDetailXDst, detailXStep, pDetailYDst, detailYStep, pDetailXYDst, detailXYStep,
    dstRoiSize, pSpecFwd, pBufferFwd);
}
```

```

printf_32f_2D("After WTFwd ->\n pApproxDst", pApproxDst, dstRoiSize, approxStep,
ippStsNoErr);
printf_32f_2D("pDetailXDst", pDetailXDst, dstRoiSize, detailXStep, ippStsNoErr);
printf_32f_2D("pDetailYDst", pDetailYDst, dstRoiSize, detailYStep, ippStsNoErr);
printf_32f_2D("pDetailXYDst", pDetailXYDst, dstRoiSize, detailXYStep, ippStsNoErr);

//-----
ippiWTInvGetSize_32f(1, lenLow, anchorLow, lenHigh, anchorHigh, &specSizeInv, &bufSizeInv);
pSpecInv = (IppiWTInvSpec_32f_C1R*)ippMalloc(specSizeInv, &bufSizeInv);
pBufferInv = (Ipp8u*)ippMalloc(bufSizeInv);
ippiWTInvInit_32f_C1R(pSpecInv, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh, anchorHigh);
//adds border to four images obtained after ippiWTFwd
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pApproxDst, approxStep, dstRoiSize, (Ipp32s*)pAppB,
StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXDst, detailXStep, dstRoiSize, (Ipp32s*)pXB,
StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailYDst, detailYStep, dstRoiSize, (Ipp32s*)pYB,
StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXYDst, detailXYStep, dstRoiSize, (Ipp32s*)pXYB,
StepB, roiInvSizeB, 0, 0);
//performs inverse wavelet transform
ippiWTInv_32f_C1R(pAppB, StepB, pXB, StepB, pYB, StepB, pXYB, StepB, roiInvSize, pDstInv,
stepDstInv, pSpecInv, pBufferInv);

printf_32f_2D("After WTFinv ->\n pDstInv", pDstInv, roiSize, stepDstInv, ippStsNoErr);

ippFree(pSpecFwd);
ippFree(pSpecInv);
ippFree(pBufferFwd);
ippFree(pBufferInv);
}

```

**Result:**

```

After WTFwd ->
pApproxDst
0.0 2.8 8.3 0.0
2.8 4.9 9.0 2.8
8.3 9.0 10.4 8.3
0.0 2.8 8.3 0.0
pDetailXDst
 0.0 1.0 3.1 0.0
 8.3 7.3 5.2 8.3
-4.2 -2.1 2.1 -4.2
 0.0 1.0 3.1 0.0
pDetailYDst
 0.0 8.3 -4.2 0.0
 1.0 7.3 -2.1 1.0
 3.1 5.2 2.1 3.1
 0.0 8.3 -4.2 0.0
pDetailXYDst
 0.0 3.1 -1.6 0.0
 3.1 0.0 4.7 3.1
-1.6 4.7 -4.7 -1.6
 0.0 3.1 -1.6 0.0
After WTFinv ->
pDstInv
 0.0 2.8 -0.3 -0.6 2.1 1.1 0.0 0.0

```

2.8	5.9	2.7	4.9	3.4	5.4	2.8	5.1
-0.3	2.7	-0.6	-1.2	2.2	0.7	-0.3	-0.5
-0.6	4.9	-1.2	-2.2	3.9	1.2	-0.6	-1.0
2.1	3.4	2.2	3.9	1.8	3.7	2.1	3.8
1.1	5.4	0.7	1.2	3.7	3.2	1.1	1.9
0.0	2.8	-0.3	-0.6	2.1	1.1	0.0	0.0
0.0	5.1	-0.5	-1.0	3.8	1.9	0.0	0.0

## See Also

[WTInvGetSize](#) Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

[WTInvInit](#) Initializes the inverse wavelet transform specification structure.

[WTFwd](#) Performs one-level wavelet decomposition of an image.

# Computer Vision

15

This chapter provides some background for the computer vision concepts used in the Intel® IPP library as well as detailed descriptions of the Intel IPP image processing functions for computer vision. These functions are combined in groups by their functionality.

## Using `ippiAdd` for Background Differencing

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term “background” stands for a set of motionless image pixels – that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, for example, with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location  $S(x, y)$ , using the function `ippiAdd`, and the sum of squares of the values  $Sq(x, y)$ , using the function `ippiAddSquare`. Then mean is calculated as

$$m(x, y) = \frac{S(x, y)}{N} ,$$

where  $N$  is the number of collected frames, and standard deviation as

$$stddev(x, y) = \sqrt{\left(\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2\right)}$$

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition  $\text{abs}(p(x, y) - m(x, . y)) < C * stddev(x, y)$ , where  $C$  is a constant, is met. If  $C$  is equal to 3, it satisfies the “three sigmas” rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function `ippiAddWeighted`. Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions `ippiAbsDiff` in chapter 5 and `ippiThreshold` in chapter 7), optical flow, and some other operations.

Relevant addition functions used for background differencing include:

`Add_8u32f_C1IR`, `Add_8s32f_C1IR`, `Add_32f_C1IR`,  
`Add_8u32f_C1IMR`, `Add_8s32f_C1IMR`, `Add_32f_C1IMR` (see `Add`),  
and also all flavors of `AddSquare`, `AddProduct`, and `AddWeighted`.

## Feature Detection Functions

This section describes feature detection functions.

The set of the Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- $D_x$  and  $D_y$  are the first  $x$  and  $y$  derivatives, respectively.
- $D_{xx}$  and  $D_{yy}$  are the second  $x$  and  $y$  derivatives, respectively.
- $D_{xy}$  is the partial  $x$  and  $y$  derivative.
- $D_{xxx}$  and  $D_{yyy}$  are the third  $x$  and  $y$  derivatives, respectively.
- $D_{xxy}$  and  $D_{xyy}$  are the third partial  $x$  and  $y$  derivatives.

## Corner Detection

The Sobel and Scharr first derivative operators are to be used to take the  $x$  and  $y$  derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A  $2 \times 2$  matrix of sums of the  $x$  and  $y$  derivatives is created as follows:

$$A = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

Solving

$$\det(C - \lambda I) = 0 ,$$

where  $\lambda$  is a column vector of the eigen values and  $I$  is the identity matrix, gives the eigen values. For the  $2 \times 2$  matrix of the equation above, the solutions may be written in a closed form:

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2} .$$

If  $\lambda_1, \lambda_2 > t$ , where  $t$  is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

## FastNGetSize

*Computes the size of the FastN context structure.*

### Syntax

```
IppStatusippiFastNGetSize(IppiSize srcSize, int circleRadius, int N, int orientationBins, int option, IppDataType dataType, int numChannels, int* pSpecSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>srcSize</i>	Size of the source ROI, in pixels.
----------------	------------------------------------

*circleRadius*

Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.

*N*

Critical number of serial pixels on circle for defining a corner. The ranges are as follows:

<i>circleRadius</i>	<i>N</i>
1	$5 \leq N \leq 8$
2	$7 \leq N \leq 12$
3	$9 \leq N \leq 16$
5	$15 \leq N \leq 28$
7	$21 \leq N \leq 40$
9	$27 \leq N \leq 52$

*orientationBins*

The number of bins to define direction. Supported values are from 2 to 64.

*option*

Mode of processing. Supported values:

- `IPP_FASTN_CIRCLE`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`

*dataType*

Data type of the source and destination images. Supported value is `ipp8u`.

*numChannels*

Number of channels in the images. Supported value is 1.

*pSpecSize*

Pointer to the computed size of the specification structure.

## Description

This function computes the size of the FastN context structure for the `FastN` function. The result is stored in *pSpecSize*.

Use the computed *pSpecSize* value to allocate memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the *Support Functions* section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the `FastN` function description.

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error.

`ippStsNullPtrErr` Indicates an error when *pSpecSize* is `NULL`.

`ippStsSizeErr` Indicates an error when *srcSize* is less than, or equal to zero.

`ippDataTypeErr` Indicates an error when *dataType* has an illegal value.

`ippNumChannelsErr` Indicates an error when *numChannels* has an illegal value.

ippOutOfRangeErr	Indicates an error when <i>orientationBins</i> or <i>N</i> has an illegal value.
ippBadArgErr	Indicates an error when <i>option</i> or <i>circleRadius</i> has an illegal value.

## See Also

[FastN](#) Detects corners in an image using the FastN algorithm.

## FastNGetBufferSize

*Computes the size of the work buffer for the [FastN](#) function.*

---

### Syntax

```
IppStatusippiFastNGetBufferSize(IppiFastNSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSpec</i>	Pointer to the FastN specification structure.
<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>pBufSize</i>	Pointer to the computed size of the work buffer.

### Description

This function computes the size of the work buffer for the [FastN](#) function. The result is stored in *pBufSize*.

Use the computed *pBufSize* value to allocate memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the *Support Functions* section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the [FastN](#) function description.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when <i>pSpec</i> or <i>pBufSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>dstRoiSize</i> is less than, or equal to zero.

## See Also

[FastN](#) Detects corners in an image using the FastN algorithm.

## FastNInit

*Initializes the FastN context structure.*

### Syntax

```
IppStatusippiFastNInit(IppiSize srcSize, int circleRadius, int N, int orientationBins,
int option, Ipp32f threshold, IppDataType dataType, int numChannels, IppiFastNSpec*
pSpec);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>srcSize</i>	Size of the source ROI, in pixels.														
<i>circleRadius</i>	Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.														
<i>N</i>	Critical number of serial pixels on circle for defining a corner. The ranges are as follows:														
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th><i>circleRadius</i></th> <th><i>N</i></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>5 ≤ N ≤ 8</td> </tr> <tr> <td>2</td> <td>7 ≤ N ≤ 12</td> </tr> <tr> <td>3</td> <td>9 ≤ N ≤ 16</td> </tr> <tr> <td>5</td> <td>15 ≤ N ≤ 28</td> </tr> <tr> <td>7</td> <td>21 ≤ N ≤ 40</td> </tr> <tr> <td>9</td> <td>27 ≤ N ≤ 52</td> </tr> </tbody> </table>	<i>circleRadius</i>	<i>N</i>	1	5 ≤ N ≤ 8	2	7 ≤ N ≤ 12	3	9 ≤ N ≤ 16	5	15 ≤ N ≤ 28	7	21 ≤ N ≤ 40	9	27 ≤ N ≤ 52
<i>circleRadius</i>	<i>N</i>														
1	5 ≤ N ≤ 8														
2	7 ≤ N ≤ 12														
3	9 ≤ N ≤ 16														
5	15 ≤ N ≤ 28														
7	21 ≤ N ≤ 40														
9	27 ≤ N ≤ 52														
<i>orientationBins</i>	The number of bins to define direction. Supported values are from 2 to 64.														
<i>option</i>	Mode of processing. Supported values: <ul style="list-style-type: none"> <li>• IPP_FASTN_CIRCLE</li> <li>• (IPP_FASTN_CIRCLE   IPP_FASTN_ORIENTATION)</li> <li>• (IPP_FASTN_CIRCLE   IPP_FASTN_SCORE_MODE0)</li> <li>• (IPP_FASTN_CIRCLE   IPP_FASTN_ORIENTATION   IPP_FASTN_SCORE_MODE0)</li> <li>• (IPP_FASTN_CIRCLE   IPP_FASTN_SCORE_MODE0   IPP_FASTN_NMS)</li> <li>• (IPP_FASTN_CIRCLE   IPP_FASTN_ORIENTATION   IPP_FASTN_SCORE_MODE0   IPP_FASTN_NMS)</li> </ul>														
<i>threshold</i>	Threshold value to detect critical pixels. The value must be more than, or equal to zero.														
<i>dataType</i>	Data type of the source and destination images. Supported value is ipp8u.														
<i>numChannels</i>	Number of channels in the images. Supported value is 1.														

*pSpec* Pointer to the specification structure.

## Description

This function initializes the FastN context structure for the [FastN](#) function.

For an example on how to use this function, refer to the example provided with the [FastN](#) function description.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when <i>pSpec</i> is NULL.
ippStsSizeErr	Indicates an error when <i>srcSize</i> is less than, or equal to zero.
ippDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.
ippOutOfRangeErr	Indicates an error when <i>orientationBins</i> or <i>N</i> has an illegal value.
ippBadArgErr	Indicates an error when <i>option</i> or <i>circleRadius</i> has an illegal value.
ippThresholdErr	Indicates an error when <i>threshold</i> is negative.

## See Also

**FastN** Detects corners in an image using the FastN algorithm.

FastN

*Detects corners in an image using the FastN algorithm.*

## Syntax

```
IppStatusippiFastN_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstCorner, int dstCornerStep, Ipp8u* pDstScore, int dstScoreStep, int* pNumCorner, IppiPoint srcRoiOffset, IppiSize dstRoiSize, IppiFastNSpec* pSpec, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDstCorner</i>	Pointer to the destination image with corners.

---

<i>dstCornerStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image with corners.
<i>pDstScore</i>	Pointer to the destination image with scores.
<i>dstScoreStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image with scores.
<i>pNumCorner</i>	Pointer to the calculated number of corners.
<i>srcRoiOffset</i>	Offset in the source image.
<i>dstRoiSize</i>	Size of the destination ROI, in pixels.
<i>pSpec</i>	Pointer to the specification structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The `ippiFastN` function implements the FastN corner detection algorithm. This function detects corners in the source image, calculates orientation and score of corners.

The figures below show pixels location for different radius values.

Radius = 1

Px3	Px4	Px5
Px2	pxc	Px6
Px1	Px0	Px7

Radius = 2

	Px5	Px6	Px7	
Px4				Px8
Px3		pxc		Px9
Px2				Px10
	Px1	Px0	Px11	

Radius = 3

		Px7	Px8	Px9		
	Px6				Px10	
Px5						Px11
Px4			pxc			Px12
Px3						Px13
	Px2				Px14	
		Px1	Px0	Px15		

Radius = 5

			Px12	Px13	Px14	Px15	Px16			
		Px11						Px17		
	Px10								Px18	
Px9										Px19
Px8										Px20
Px7				pxc						Px21
Px6										Px22
Px5										Px23
	Px4								Px24	
		Px3							Px25	
			Px2	Px1	Px0	Px27	Px26			

Radius = 7

					Px18	Px19	Px20	Px21	Px22				
			Px16	Px17						Px23	Px24		
		Px15										Px25	
	Px14												Px26
	Px13												Px27
Px12													Px28
Px11													Px29
Px10										pxc			
Px9													Px30
Px8													Px31
	Px7												Px32
	Px6												Px33
		Px5											Px34
			Px4	Px3								Px35	
					Px2	Px1	Px0	Px39	Px38	Px37			Px36

Radius = 9

					Px21	Px22	Px23	Px24	Px25	Px26	Px27	Px28	Px29	Px30	Px31		
		Px20															Px32
		Px19															Px33
		Px18															
		Px17															
Px16																	
Px15																	
Px14																	
Px13																	
Px12																	
Px11																	
Px10																	
Px9																	
Px8			Px7														Px45
			Px6		Px5	Px4											Px46
					Px3	Px2	Px1	Px0	Px51	Px50	Px49						Px47

The function defines a pixel as a corner if the value of  $N$  consecutive pixels located on the circle with the center at the current pixel is greater (less) than the value of the current - central pixel. Differences between the values of pixels on the circle and the central pixel value must be greater than the *threshold* value. In this case, the corresponding pixel of the *pDstCorner* image is set to the following values in binary format:

- If greater - 01xxxxxx
- If less - 10xxxxxx

If the *IPP\_FASTN\_ORIENTATION* mode is not set, *xxxxxx* = 000000, otherwise, *xxxxxx* = the number of sector (bin) to which the corner directs. The *orientationBin* parameter defines the number of sectors that is computed from the bottom clockwise. If the source pixel is not a corner, the corresponding pixel of *pDstCorner* is set to zero.

If the `IPP_FASTN_SCORE_MODE0` mode is set, the corresponding pixel of `pDstCorner` is set to the corner score. Score is a maximum among minimal differences with `threshold` calculated for every N consecutive pixels. If the source pixel is not a corner, the corresponding pixel of `pDstScore` is set to zero.

If the `IPP_FASTN_NMS` mode is set, the corners that have a corner with the greater score among neighboring pixels are cancelled.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when:
	<ul style="list-style-type: none"> <li><code>pSrc</code>, <code>pDstCorner</code>, <code>pNumCorner</code>, <code>pSpec</code>, or <code>pBuffer</code> is <code>NULL</code></li> <li><code>pDstScore</code> is <code>NULL</code> if <code>option</code> is not equal to <code>IPP_FASTN_SCORE_MODE0</code></li> </ul>
<code>ippStsSizeErr</code>	Indicates an error when <code>dstRoiSize</code> is less than, or equal to zero.
<code>ippDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.
<code>ippOutOfRangeErr</code>	Indicates an error when <code>orientationBins</code> or <code>N</code> has an illegal value.
<code>ippBadArgsErr</code>	Indicates an error when <code>option</code> or <code>circleRadius</code> has an illegal value.
<code>ippThresholdErr</code>	Indicates an error when <code>threshold</code> is negative.

## Example

To better understand usage of the `ippiFastN` function, refer to the `FastN.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

`FastNGetSize` Computes the size of the FastN context structure.

`FastNInit` Initializes the FastN context structure.

`FastNGetBufferSize` Computes the size of the work buffer for the `FastN` function.

`FastN2DToVec` Converts corners from two-dimensional image to an array of structures.

## FastN2DToVec

*Converts corners from two-dimensional image to an array of structures.*

## Syntax

```
IppStatusippiFastN2DToVec_8u(const Ipp8u* pSrcCorner, int srcCornerStep, const Ipp8u* pSrcScore, int srcScoreStep, IppiCornerFastN* pDst, IppisizesrcRoiSize, int maxLen, int* pNumCorners, IppiFastNSpec* pSpec);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrcCorner</i>	Pointer to the source image with corners.
<i>srcCornerStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image with corners.
<i>pSrcScore</i>	Pointer to the source image with scores.
<i>srcScoreStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image with score.
<i>pDst</i>	Pointer to the destination vector of structures.
<i>srcRoiSize</i>	Size of the source ROI, in pixels.
<i>maxLen</i>	Length of the array of structures.
<i>pNumCorners</i>	Pointer to the number of corners in the source image.
<i>pSpec</i>	Pointer to the specification structure.

## Description

This function converts two-dimensional image with corners to an array of structures. The result is stored in *pDst*.

For an example on how to use this function, refer to the example provided with the [FastN](#) function description.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error when:
	<ul style="list-style-type: none"> <li><i>pSrcCorner</i>, <i>pDst</i>, <i>pNumCorners</i>, or <i>pSpec</i> is NULL</li> <li><i>pSrcScore</i> is NULL if <i>option</i> is not equal to <code>IPP_FASTN_SCORE_MODE0</code></li> </ul>
<i>ippStsSizeErr</i>	Indicates an error when <i>srcRoiSize</i> is less than, or equal to zero.
<i>ippContextMatchErr</i>	Indicates an error when the specification structure does not match the operation.
<i>ippDataTypeErr</i>	Indicates an error when <i>dataType</i> has an illegal value.
<i>ippNumChannelsErr</i>	Indicates an error when <i>numChannels</i> has an illegal value.
<i>ippOutOfRangeErr</i>	Indicates an error when <i>orientationBins</i> or <i>N</i> has an illegal value.
<i>ippBadArgsErr</i>	Indicates an error when <i>option</i> or <i>circleRadius</i> has an illegal value.
<i>ippThresholdErr</i>	Indicates an error when <i>threshold</i> is negative.

## See Also

[FastN](#) Detects corners in an image using the FastN algorithm.

## HarrisCornerGetBufferSize

*Calculates the size of the temporary buffer for the ippiHarrisCorner function.*

## Syntax

```
IppStatusippiHarrisCornerGetBufferSize(IppiSize roiSize, IppiMaskSize filterMask,  
Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>filterMask</i>	Size of the derivative filter aperture. Possible values are <code>ippMskSize3x3</code> and <code>ippMskSize5x5</code> .
<i>avgWndSize</i>	Linear size of a neighborhood block for averaging.
<i>dataType</i>	Data type of the source image. Possible values are <code>ipp8u</code> and <code>ipp32f</code> .
<i>numChannels</i>	Number of channels in the images. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size (in bytes) of the external work buffer.

## Description

This function calculates the size of the temporary buffer needed for the [HarrisCorner](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"><li>• when <i>roiSize</i> is less than, or equal to zero</li><li>• when <i>avgWndSize</i> is equal to zero</li></ul>
<code>ippStsMaskSizeErr</code>	Indicates an error when <i>filterMask</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[HarrisCorner](#) Implements Harris corner detection algorithm.

## HarrisCorner

Implements Harris corner detection algorithm.

## Syntax

```
IppStatusippiHarrisCorner_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatusippiHarrisCorner_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.								
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.								
<i>pDst</i>	Pointer to the destination image.								
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.								
<i>roiSize</i>	Size of the source and destination image ROI in pixels.								
<i>filterType</i>	Type of the derivative operator. Possible values are: ippKernelSobel              Sobel filter ippKernelScharr              Scharr filter ippKernelCentralDiff         Central differences operator								
<i>filterMask</i>	Size of the derivative filter aperture. The list of possible values depends on the <i>filterType</i> value:								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Filter Type</th> <th style="text-align: center;">Filter Mask</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">ippKernelSobel</td> <td style="text-align: center;">ippMskSize3x3, ippMskSize5x5</td> </tr> <tr> <td style="text-align: center;">ippKernelScharr</td> <td style="text-align: center;">ippMskSize3x3</td> </tr> <tr> <td style="text-align: center;">ippKernelCentralDiff</td> <td style="text-align: center;">ippMskSize3x3</td> </tr> </tbody> </table>		Filter Type	Filter Mask	ippKernelSobel	ippMskSize3x3, ippMskSize5x5	ippKernelScharr	ippMskSize3x3	ippKernelCentralDiff	ippMskSize3x3
Filter Type	Filter Mask								
ippKernelSobel	ippMskSize3x3, ippMskSize5x5								
ippKernelScharr	ippMskSize3x3								
ippKernelCentralDiff	ippMskSize3x3								
<i>avgWndSize</i>	Linear size of a neighborhood block for averaging.								
<i>k</i>	Harris detector free coefficient.								
<i>scale</i>	Destination image scale factor.								
<i>borderType</i>	Type of border. Possible values are: ippBorderConst              Values of all border pixels are set to constant.								

ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderRepl` or `ippBorderConst` and the following flags:

- `ippBorderInMemTop`
- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

*borderValue*

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type. The `ippiHarrisCorner` function uses the specified constant value only at the first stage of the algorithm. At the third stage, the function uses zero constant border.

*pBuffer*

Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the [HarrisCornerGetBufferSize](#) function.

## Description

This function goes through the following stages to implement the Harris corner detection algorithm:

1. Computes  $I^{(x, y)}_x$  and  $I^{(x, y)}_y$  gradients for each  $(x, y)$  pixel of the image. The function computes gradients using the derivative operator specified by the `filterType` and `filterMask` parameters.
2. Computes products of the gradients at each  $(x, y)$  pixel of the image:

$$I_{xx}^{(x, y)} = I_x^{(x, y)} * I_x^{(x, y)}, \quad I_{yx}^{(x, y)} = I_{xy}^{(x, y)} = I_x^{(x, y)} * I_y^{(x, y)}, \quad I_{yy}^{(x, y)} = I_y^{(x, y)} * I_y^{(x, y)}$$

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the `avgWndSize` value.

$$S_{xx}^{(x, y)} = \sum_{y'} \sum_{x'} I_{xx}^{(x', y')}, \quad S_{xy}^{(x, y)} = \sum_{y'} \sum_{x'} I_{xy}^{(x', y')}, \quad S_{yy}^{(x, y)} = \sum_{y'} \sum_{x'} I_{yy}^{(x', y')}$$

4. Defines 2x2 gradient covariance matrix  $H^{(x, y)}$  over a rectangular neighborhood block for each  $(x, y)$  pixel of the image.

$$H^{(x, y)} = \begin{pmatrix} S_{xx}^{(x, y)} & S_{xy}^{(x, y)} \\ S_{yx}^{(x, y)} & S_{yy}^{(x, y)} \end{pmatrix}$$

5. Computes the corner response at each pixel of the image:

$$dst(x, y) = \det H^{(x,y)} - k * (\text{tr}H^{(x,y)})^2$$

where

*k* is the Harris detector free parameter

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the *borderType* parameter.

The *scale* parameter is applied to the output image.

Before using this function, compute the size of the temporary work buffer using the [HarrisCornerGetBufferSize](#) function.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when <i>pSrc</i> , <i>pDst</i> , or <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>when <i>roiSize</i> is less than, or equal to zero</li> <li>when <i>avgWndSize</i> is equal to zero</li> </ul>
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating point images.
ippStsFilterTypeErr	Indicates an error when <i>filterType</i> has an illegal value.
ippStsMaskSizeErr	Indicates an error when <i>filterMask</i> has an illegal value.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> has a negative value.
ippStsInplaceModeNotSupportedErr	Indicates an error when <i>pSrc</i> and <i>pDst</i> point to the same image.

## Example

The code example below demonstrates how to use the [ippiHarrisCorner\\_8u32f\\_C1R](#) and [ippiHarrisCornerGetBufferSize](#) functions.

```

...
int     bufSize = 0;
Ipp8u* pBuffer  = 0;
Ipp32u numChannels = 1;
IppStatus status = ippStsNoErr;
IppiBorderType borderType = ippBorderRepl;
IppiDifferentialKernel filterType = ippFilterSobel;
IppiMaskSize filterMask = ippMskSize5x5;
Ipp32u avgWndSize = 3;
Ipp32f scale = 1.0f;

/* Computes the temporary work buffer size */
status = ippiHarrisCornerGetBufferSize(roiSize, filterMask, avgWndSize, ipp8u, numChannels,
&bufSize);

/* Memory allocation */

```

```

if (status != ippStsNoErr) pBuffer = ippsMalloc_8u(bufSize);

if (pBuffer != NULL)
{
    /* Harris Corner processing */
    status =ippiHarrisCorner_8u32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize, filterType,
filterMask, avgWndSize, 0.04f,
scale, borderType, 0, pBuffer);

    ippsFree(pBuffer);
}
...

```

## See Also

[HarrisCornerGetBufferSize](#) Calculates the size of the temporary buffer for the `ippiHarrisCorner` function.

## Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [Canny86]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

### Stage 1: Differentiation

The image data is differentiated in  $x$  and  $y$  directions. From the computed  $x$  and  $y$  gradient components, Canny edge detector functions calculate the magnitude and angle of the gradient vector.

---

#### NOTE

The `ippiSobel` functions perform the first stage and Canny edge detector functions use their output.

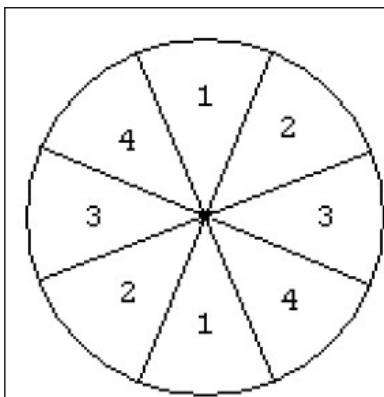
---

### Stage 2: Non-Maximum Suppression

With the rate of intensity change found at each point in the image, edges must be placed at the points of maximum values of gradient magnitude. It is preferable to suppress non-maximum points that are perpendicular to the edge direction, rather than parallel to the edge direction, as the edge is strong along an extended contour.

The algorithm starts off with sorting the direction of gradient to one of four sectors shown in the figure below.

#### Gradient Sectors



The algorithm passes a  $3 \times 3$  neighborhood across the magnitude array. At each point, the center element of the neighborhood is compared with its two neighbors along the line of the gradient given by the sector value. If the central value is not greater than the neighbors, it is suppressed.

### Stage 3: Edge Thresholding

The Canny operator uses the so-called “hysteresis” thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit, it is immediately accepted. If the value lies below the low threshold, it is rejected. Points which lie between the two limits are accepted if they are connected to pixels which are also accepted. The likelihood of streaking is small, since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J.Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

[Example](#) shows how to use the Intel IPP functions for the Canny edge detector.

### CannyBorderGetSize

*Calculates the size of the temporary buffer for the ippiCannyBorder function.*

### Syntax

```
IppStatus ippiCannyBorderGetSize(IppiSize roiSize, IppiDifferentialKernel filterType,
IppiMaskSize mask, IppDataType dataType, int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the image ROI in pixels.
<i>filterType</i>	Type of the filter to be applied. Possible values are <code>ippFilterSobel</code> and <code>ippFilterScharr</code> .
<i>mask</i>	The size of the mask. Possible values are <code>ippMskSize3x3</code> and <code>ippMskSize5x5</code> .
<i>dataType</i>	Data type of the image. Possible value is <code>ipp8u</code> .
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

### Description

This function calculates the size of the temporary buffer needed for the [CannyBorder](#) function.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when the <i>pBufferSize</i> pointer is <b>NULL</b> .
ippStsMaskSizeErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsSizeErr	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.

## See Also

[CannyBorder](#) Implements Canny algorithm for edge detection.

## CannyBorder

*Implements Canny algorithm for edge detection.*

## Syntax

```
IppStatusippiCannyBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp32f lowThresh, Ipp32f highThresh, IppNormType norm, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.i.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.i.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>filterType</i>	Type of the filter to be applied. Possible values are <code>ippFilterSobel</code> and <code>ippFilterScharr</code> .
<i>mask</i>	The size of the mask. Possible values are <code>ippMskSize3x3</code> and <code>ippMskSize5x5</code> for <code>ippFilterSobel</code> , and <code>ippMskSize3x3</code> for <code>ippFilterScharr</code> .
<i>borderType</i>	Type of border. Possible values are:
	<code>ippBorderConst</code> Values of all border pixels are set to constant.
	<code>ippBorderRepl</code> Border is replicated from the edge pixels.

<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
	Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderRepl</code> and <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> .
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>lowThresh</code>	Lower threshold for edges detection.
<code>highThresh</code>	Upper threshold for edges detection.
<code>norm</code>	Normalization mode. Possible values are <code>ippNormL1</code> and <code>ippNormL2</code> .
<code>pBuffer</code>	Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the <a href="#">CannyBorderGetSize</a> function.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds edges in the ROI of the source image with the user-defined border types using the Canny edge detector algorithm. The output image is stored in `pDst`.

Before using this function, compute the size of the temporary work buffer using the [CannyBorderGetSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>mask</code> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> or <code>dstStep</code> is less than <code>roi.width*pixelSize</code> .
<code>ippStsBadArgErr</code>	Indicates an error when <code>lowThresh</code> is negative, or <code>highThresh</code> is less than <code>lowThresh</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when one of the step values is not divisible by 2 for <code>16s</code> images, and by 4 for <code>32f</code> images.

## Example

To better understand usage of the `ippiCannyBorder` function, refer to the `CannyBorder.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[CannyBorderGetSize](#) Calculates the size of the temporary buffer for the `ippiCannyBorder` function.

## CannyGetSize

*Calculates size of temporary buffer for the `ippiCanny` function.*

## Syntax

```
IppStatusippiCannyGetSize(IppiSize roiSize, int* pBufferSize);
IppStatusippiCannyGetSize_L(IppiSizeL roi, IppSizeL* bufferSize);
```

## Include Files

ippcv.h

Flavors with the \_L suffix: ippcv\_L.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

*roiSize, roi* Size of the image ROI, in pixels.

*pBufferSize, bufferSize* Pointer to the computed size of the temporary buffer.

## Description

This function calculates the size of a temporary buffer for the [ippiCanny](#) function.

## Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error condition if *pBufferSize* pointer is NULL.

ippStsSizeErr Indicates an error condition if *pRoiSize* has a field with zero or negative value.

## Canny

*Implements Canny algorithm for edge detection.*

---

## Syntax

```
IppStatusippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy, int
srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold,
Ipp32f highThreshold, Ipp8u* pBuffer);
```

```
IppStatusippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy, int
srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold,
Ipp32f highThreshold, Ipp8u* pBuffer);
```

## Platform-aware functions

```
IppStatusippiCanny_16s8u_C1R_L(Ipp16s* pSrcDx, IppSizeL srcDxStep, Ipp16s* pSrcDy,
IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f
lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);
```

```
IppStatusippiCanny_32f8u_C1R_L(Ipp32f* pSrcDx, IppSizeL srcDxStep, Ipp32f* pSrcDy,
IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f
lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

Flavors with the `_L` suffix: `ippcv_1.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrcDx</code>	Pointer to the source image ROI <i>x</i> -derivative.
<code>srcDxStep</code>	Distance in bytes between starts of consecutive lines in the source image <code>pSrcDx</code> .
<code>pSrcDy</code>	Pointer to the source image ROI <i>y</i> -derivative.
<code>srcDyStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image <code>pSrcDy</code> .
<code>pDstEdges</code>	Pointer to the output array of the detected edges.
<code>dstEdgeStep</code>	Distance, in bytes, between the starting points of consecutive lines in the output image.
<code>roiSize</code>	Size of the source image ROI in pixels.
<code>lowThreshold</code>	Lower threshold for edges detection.
<code>highThreshold</code>	Upper threshold for edges detection.
<code>norm</code>	Normalization type; supported values: <code>ippNormL1</code> and <code>ippNormL2</code> .
<code>pBuffer</code>	Pointer to the pre-allocated temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds edges in the source image ROI and stores them into the output image `pDstEdges` using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiCannyGetSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDxStep</code> , <code>srcDyStep</code> or <code>dstEdgeStep</code> is less than <code>roi.width * &lt;pixelSize&gt;</code> .
<code>ippStsBadArgErr</code>	Indicates an error when <code>lowThresh</code> is negative or <code>highThresh</code> is less than <code>lowThresh</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 2 for <code>16s</code> images, and by 4 for <code>32f</code> images.

## Example

To better understand usage of the `ippiCanny` function, refer to the `Canny.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## EigenValsVecsGetBufferSize

*Calculates size of temporary buffer for the function  
ippiEigenValsVecs.*

---

### Syntax

```
IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
```

```
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>apertureSize</i>	Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the size of a temporary buffer to be used by the function [ippiEigenValsVecs](#).

---

#### Caution

The parameters *apertureSize* and *avgWindow* must be the same for both functions [ippiEigenValsVecsGetBufferSize](#) and [ippiEigenValsVecs](#).

---

[Example 14-2](#) shows how to use the function [ippiEigenValsVecsGetBufferSize](#).

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiWidth</i> has zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value.

## EigenValsVecsBorder

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

---

## Syntax

```
IppStatusippiEigenValsVecsBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*  
pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int  
avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);  
  
IppStatusippiEigenValsVecsBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*  
pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int  
avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pEigenVV</i>	Image to store the results.
<i>eigStep</i>	Distance, in bytes, between the starting points of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are:  ippKernelSobel                    Sobel kernel 3x3 or 5x5 ippKernelSobelNeg                Negative Sobel kernel 3x3 or 5x5 ippKernelScharr                 Scharr kernel 3x3
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.
<i>border</i>	Type of image border. Possible values:  ippBorderConst                 Values of all border pixels are set to a constant. ippBorderRepl                  Border is replicated from the edge pixels.
<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The *apertureSize* parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.

---

### **Caution**

If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

---

The function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The image *eigenVV* has the following format. For every pixel of the source image it contains six floating-point values -  $\lambda_1, \lambda_2, x_1, y_1, x_2, y_2$ . These values are defined as follows:

$\lambda_1, \lambda_2$	Eigen values of the above matrix ( $\lambda_1 \geq \lambda_2 \geq 0$ ).
$x_1, y_1$	Coordinates of the normalized eigen vector corresponding to $\lambda_1$ .
$x_2, y_2$	Coordinates of the normalized eigen vector corresponding to $\lambda_2$ .

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary work buffer. Before using this function, compute the size of the buffer using the [ippiEigenValsVecsGetBufferSize](#) function.

---

### **Caution**

The parameters *apertureSize* and *avgWindow* must be the same for both functions [ippiEigenValsVecsGetBufferSize](#) and [ippiEigenValsVecsBorder](#).

---

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value; or if <i>kernType</i> has a wrong value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width*pixelSize</i> , or <i>eigStep</i> is less than <i>roiSize.width*sizeof(Ipp32f)*6</i> .

---

ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.
ippStsBorderErr	Indicates an error if <i>borderType</i> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[EigenValsVecsGetBufferSize](#) Calculates size of temporary buffer for the function `ippiEigenValsVecs`.

## EigenValsVecs

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

## Syntax

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV,
int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int
avgWindow, Ipp8u* pBuffer);

IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV,
int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int
avgWindow, Ipp8u* pBuffer);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pEigenVV</i>	Image to store the results.
<i>eigStep</i>	Distance, in bytes, between the starting points of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are:
	ippKernelSobel              Sobel kernel 3x3 or 5x5
	ippKernelSobelNeg          Negative Sobel kernel 3x3 or 5x5
	ippKernelScharr            Scharr kernel 3x3
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The *apertureSize* parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.

---

### **Caution**

If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

---

The function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The image *eigenVV* has the following format. For every pixel of the source image it contains six floating-point values -  $\lambda_1, \lambda_2, x_1, y_1, x_2, y_2$ . These values are defined as follows:

$\lambda_1, \lambda_2$	Eigen values of the above matrix ( $\lambda_1 \geq \lambda_2 \geq 0$ ).
$x_1, y_1$	Coordinates of the normalized eigen vector corresponding to $\lambda_1$ .
$x_2, y_2$	Coordinates of the normalized eigen vector corresponding to $\lambda_2$ .

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiEigenValsVecsGetBufferSize](#).

---

### **Caution**

The parameters *apertureSize* and *avgWindow* must be the same for both functions [ippiEigenValsVecsGetBufferSize](#) and [ippiEigenValsVecs](#).

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value; or if <i>kernType</i> has wrong value.

---

ippStsStepErr	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width*pixelSize</i> , or <i>eigStep</i> is less than <i>roiSize.width*sizeof(Ipp32f)*6</i> .
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.

## Example

To better understand usage of the `ippiEigenValsVecs` function, refer to the `EigenValsVecs.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>:

## MinEigenValGetBufferSize

*Calculates size of temporary buffer for the function  
ippiMinEigenVal.*

---

## Syntax

```
IppStatusippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int
avgWindow, int* pBufferSize);

IppStatusippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize,
int avgWindow, int* pBufferSize);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>apertureSize</i>	Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBufferSize</i>	Pointer to the variable that returns the size of the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function calculates the size of a temporary buffer to be used by the function `ippiMinEigenVal`.

### Caution

The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiWidth</i> has a zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value.

## MinEigenValBorder

*Calculates the minimal eigen value of image blocks for corner detection.*

## Syntax

```
IppStatusippiMinEigenValBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*  
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int  
apertureSize, int avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u*  
pBuffer);
```

```
IppStatusippiMinEigenValBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*  
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int  
apertureSize, int avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u*  
pBuffer);
```

## Include Files

`ippcv.h`

# Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pMinEigenVal</i>	Pointer to the image that is filled with the minimal eigen values.
<i>minValStep</i>	Distance, in bytes, between the starting points of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are:  ippKernelSobel                    Sobel kernel 3x3 or 5x5 ippKernelSobelNeg                Negative Sobel kernel 3x3 or 5x5 ippKernelScharr                 Scharr kernel 3x3
<i>apertureSize</i>	Size of the derivative operator, in pixels; possible values are 3 or 5.
<i>avgWindow</i>	Size of the averaging window, in pixels; possible values are 3 or 5.
<i>borderType</i>	Type of image border. Possible values:  ippBorderConst                Values of all border pixels are set to a constant.

---

	ippBorderRepl	Border is replicated from the edge pixels.
	ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.
<i>borderValue</i>		Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<i>pBuffer</i>		Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the `kernType` parameter. The `apertureSize` parameter specifies the size of the Sobel kernel. If `apertureSize` is set to 3, the function uses 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter `apertureSize` must be set to 3 if the Scharr kernel is used.

---

### Caution

If the parameter `apertureSize` is set to 5 for operation with the Scharr kernel, the function returns error status.

---

The function computes the minimal eigen value  $\lambda$  ( $\lambda \geq 0$ ) of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The function requires a temporary work buffer. Before using this function, compute the size of the work buffer using the [ippiMinEigenValGetBufferSize](#) function.

---

### Caution

The parameters `apertureSize` and `avgWindow` must be the same for both functions [ippiMinEigenValGetBufferSize](#) and [ippiMinEigenValBorder](#).

---

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value, or if <code>apertureSize</code> or <code>avgWindow</code> has an illegal value; or if <code>kernType</code> has wrong value.
ippStsStepErr	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width*pixelSize</code> , or <code>eigenvvStep</code> is less than <code>roiSize.width*sizeof(Ipp32f)</code> .

<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.
<code>ippStsBorderErr</code>	Indicates an error if <i>borderType</i> has an illegal value.

## See Also

## Regions of Interest in Intel IPP

**MinEigenValGetBufferSize** Calculates size of temporary buffer for the function `ippiMinEigenVal`.

## MinEigenVal

*Calculates the minimal eigen value of image blocks for corner detection.*

## Syntax

```
IppStatusippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*  
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int  
apertureSize, int avgWindow, Ipp8u* pBuffer);
```

```
IppStatusippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*  
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int  
apertureSize, int avgWindow, Ipp8u* pBuffer);
```

## Include Files

`ippcv.h`

# Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h,ippi.h

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMinEigenVal</i>	Pointer to the image that is filled with the minimal eigen values.
<i>minValStep</i>	Distance in bytes between starts of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are:  ippKernelSobel                    Sobel kernel 3x3 or 5x5 ippKernelSobelNeg                Negative Sobel kernel 3x3 or 5x5 ippKernelScharr                 Scharr kernel 3x3
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.
<i>avgWindow</i>	Size of the averaging window in pixels, possible values are 3 or 5.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)). This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . When using the `MinEigenVal` function, you need to verify that the processing pixels beyond the ROI are available in memory. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the `kernType` parameter. The size of the Sobel kernel may be specified by the parameter `apertureSize`. If this parameter is set to 3, the function uses 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter `apertureSize` must be set to 3 if the Scharr kernel is used.

### **Caution**

If the parameter `apertureSize` is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value  $\lambda (\lambda \geq 0)$  of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The function requires a temporary working buffer; its size should be computed previously by calling the function [`ippiMinEigenValGetBufferSize`](#).

### **Caution**

The parameters `apertureSize` and `avgWindow` must be the same for both functions [`ippiMinEigenValGetBufferSize`](#) and [`ippiMinEigenVal`](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value, or if <code>apertureSize</code> or <code>avgWindow</code> has an illegal value; or if <code>kernType</code> has wrong value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width*pixelSize</code> , or <code>eigenvvStep</code> is less than <code>roiSize.width*sizeof(Ipp32f)</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

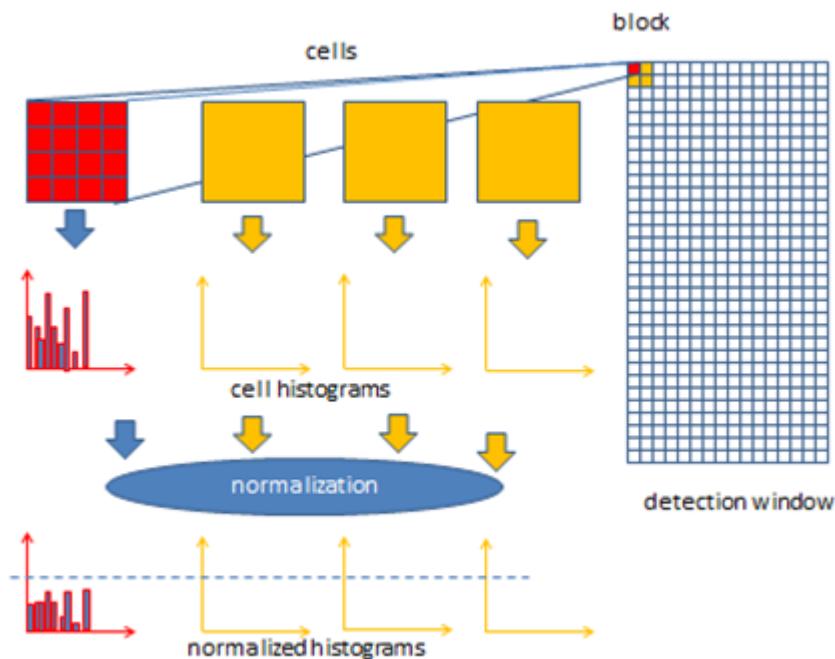
## Histogram of Oriented Gradients (HOG) Descriptor

Histogram of oriented gradients (HOG) is a feature descriptor used to detect objects in computer vision and image processing. The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI).

Implementation of the HOG descriptor algorithm is as follows:

1. Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.
2. Discretize each cell into angular bins according to the gradient orientation.
3. Each cell's pixel contributes weighted gradient to its corresponding angular bin.
4. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.
5. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

The following figure demonstrates the algorithm implementation scheme:



Computation of the HOG descriptor requires the following basic configuration parameters:

- Masks to compute derivatives and gradients
- Geometry of splitting an image into cells and grouping cells into a block
- Block overlapping
- Normalization parameters

According to [Dalal05] the recommended values for the HOG parameters are:

- 1D centered derivative mask [-1, 0, +1]
- Detection window size is 64x128
- Cell size is 8x8
- Block size is 16x16 (2x2 cells)

Intel® IPP implementation does not assume any default fixed set of parameters values. The [IppiHOGConfig](#) structure defines HOG parameters used in Intel IPP functions.

There are some limitations to the values of basic configuration parameters:

```
#define IPP_HOG_MAX_CELL    (16) /* max size of cell */
#define IPP_HOG_MAX_BLOCK    (64) /* max size of block */
#define IPP_HOG_MAX_BINS     (16) /* max number of bins */
```

## See Also

[Structures and Enumerators](#)

## HOGGetSize

*Computes the size of the HOG context structure.*

### Syntax

```
IppStatusippiHOGGetSize(const IppiHOGConfig* pConfig, int* pHOGSpecSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

*pConfig* Pointer to the HOG context structure.

*pHOGSpecSize* Pointer to the size of the HOG context structure, in bytes.

### Description

This function checks the parameters of the HOG configuration and computes the size, in bytes, of the HOG context structure *pHOGSpecSize*.

For an example on how to use this function, refer to the example provided with the [HOG](#) function description.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when one of the <i>winSize</i> fields in the <i>pConfig</i> parameter has a zero or negative value.
ippStsNotSupportedMod	Indicates an error in HOG configuration:
eErr	<ul style="list-style-type: none"> <li>• <i>cellSize</i> is less than 2, or more than IPP_HOG_MAX_CELL</li> <li>• <i>cellSize</i> is more than <i>blockSize</i>, or <i>blockSize</i> is more than IPP_HOG_MAX_BLOCK</li> <li>• <i>blockSize</i> is not a multiple of <i>cellSize</i></li> <li>• Block does not have 2x2 cell geometry</li> <li>• <i>blockStride</i> is not a multiple of <i>cellSize</i></li> <li>• Detection window size is not a multiple of <i>blockStride</i></li> <li>• <i>nbins</i> is less than 2, or more than IPP_HOG_MAX_BINS</li> <li>• <i>sigma</i> or <i>threshold</i> value is less than, or equal to zero</li> </ul>

### See Also

[HOG](#) Computes the HOG descriptor.

### HOGInit

*Initializes the HOG context structure.*

### Syntax

```
IppStatusippiHOGInit(const IppiHOGConfig* pConfig, IppiHOGSpec* pHOGSpec);
```

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

*pConfig* Pointer to the HOG context structure.

*pHOGSpec* Pointer to the HOG context structure.

## Description

This function checks the parameters of the HOG configuration and initializes the HOG context structure.

For an example on how to use this function, refer to the example provided with the [HOG](#) function description.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when one of the <i>winSize</i> fields in the <i>pConfig</i> parameter has a zero or negative value.
ippStsNotSupportedMod	Indicates an error in HOG configuration:
eErr	<ul style="list-style-type: none"> <li>• <i>cellSize</i> is less than 2, or more than IPP_HOG_MAX_CELL</li> <li>• <i>cellSize</i> is more than <i>blockSize</i>, or <i>blockSize</i> is more than IPP_HOG_MAX_BLOCK</li> <li>• <i>blockSize</i> is not a multiple of <i>cellSize</i></li> <li>• Block does not have 2x2 cell geometry</li> <li>• <i>blockStride</i> is not a multiple of <i>cellSize</i></li> <li>• Detection window size is not a multiple of <i>blockStride</i></li> <li>• <i>nbins</i> is less than 2, or more than IPP_HOG_MAX_BINS</li> <li>• <i>sigma</i> or <i>threshold</i> value is less than, or equal to zero</li> </ul>

## See Also

[HOG](#) Computes the HOG descriptor.

## [HOGGetBufferSize](#)

Computes the size of the work buffer for the [ippiHOG](#) function.

---

## Syntax

```
IppStatusippiHOGGetBufferSize(const IppiHOGSpec* pHOGSpec, IppiSize roiSize, int* pBufferSize);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pHOGSpec</i>	Pointer to the HOG context structure.
<i>roiSize</i>	Maximum size of the source image ROI, in pixels.
<i>pBufferSize</i>	Pointer to the size of the work buffer, in bytes.

## Description

This function computes the size of the work buffer for the [HOG](#) function.

For an example on how to use this function, refer to the example provided with the [HOG](#) function description.

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error when one of the specified pointers is NULL.
<i>ippStsContextmatchErr</i>	Indicates an error when the context parameter does not match the operation.

## See Also

[HOG](#) Computes the HOG descriptor.

## [HOGGetDescriptorSize](#)

Computes the size of the HOG descriptor.

## Syntax

```
IppStatusippiHOGGetDescriptorSize(const IppiHOGSpec* pHOGSpec, int* pWinDescriptorSize);
```

## Include Files

ippi.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pHOGSpec</i>	Pointer to the HOG context structure.
<i>pWinDescriptorSize</i>	Pointer to the size of the descriptor window, in bytes.

## Description

This function computes the size of the buffer for a single detection window. If the subsequent call(s) of the [HOG](#) function target processing of multiple detection windows, the size of the buffer must be increased proportionally.

For an example on how to use this function, refer to the example provided with the [HOG](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsContextmatchErr</code>	Indicates an error when the context parameter does not match the operation.

## See Also

**HOG** Computes the HOG descriptor.

## HOG

*Computes the HOG descriptor.*

---

### Syntax

```
IppStatusippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize roiSize,
const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGSpec,
IppiBorderType borderID, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u32f_C1R	16u32f_C1R	16s32f_C1R	32f_C1R
-----------	------------	------------	---------

```
IppStatusippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize roiSize,
const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGCtx,
IppiBorderType borderID, Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u32f_C3R	16u32f_C3R	16s32f_C3R	32f_C3R
-----------	------------	------------	---------

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<code>roiSize</code>	Size of the source image ROI, in pixels.
<code>pLocation</code>	Pointer to the array with detection window locations.
<code>nLocations</code>	Number of locations.
<code>pDst</code>	Pointer to the HOG descriptor.
<code>pHOGCtx</code> , <code>pHOGSpec</code>	Pointer to the HOG context/specification structure.
<code>borderID</code>	Type of border. Possible values are:

<code>ippBorderConst</code>	Values of all border pixels are set to a constant.
<code>ippBorderRepl</code>	Border is replicated from the edge pixels.
<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
<code>ippBorderMirror</code>	Border pixels are mirrored from the source image boundary pixels.
	Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderRepl</code> and <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> .
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
<code>pBuffer</code>	Pointer to the work buffer.

## Description

This function computes the HOG descriptor over defined locations of the detection window. Flavors with the C1 suffix operate on one-channel (gray) images, and C3 flavors operate on color images.

Before using this function, compute the size of the context structure, work buffer, and descriptor using the `HOGGetSize`, `HOGGetBufferSize`, and `HOGGetDescriptorSize` functions, respectively. To initialize the HOG context structure, use the `HOGInit` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsContextmatchErr</code>	Indicates an error when the context parameter does not match the operation.
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> is less than, or equal to zero.
<code>ippStsNotEvenStepErr</code>	Indicates an error when <code>srcStep</code> is not divisible by input data type size.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderID</code> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> is less than, or equal to zero.

## Example

To better understand usage of the `ippiHOG` function, refer to the `HOG.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

`HOGGetSize` Computes the size of the HOG context structure.

`HOGGetBufferSize` Computes the size of the work buffer for the `ippiHOG` function.

`HOGGetDescriptorSize` Computes the size of the HOG descriptor.

`HOGInit` Initializes the HOG context structure.

## Hough Transform

The Hough transform is a general technique that allows to detect the flat curves in binary images [Gon93]. The current version of Intel IPP implements the following:

- Detection of the straight lines that are defined by the parametric equation:  

$$r = x \cos(\theta) + y \sin(\theta)$$
, where  $r$  and  $\theta$  are the length and angle from the origin of a normal to the line, respectively.
- Detection of lines using the probabilistic Hough transform algorithm [Matas00].

## HoughLineGetSize

*Computes the size of the working buffer for the straight lines detection.*

---

### Syntax

```
IppStatusippiHoughLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int maxLineCount, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

### Parameters

<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).
<i>maxLineCount</i>	Number of elements of the destination buffer.
<i>pBufSize</i>	Pointer to the size of the working buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the temporary working buffer that is required for the function [ippiHoughLine](#).

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if <i>pBufSize</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> or <i>delta</i> has a field with zero or negative value.

## HoughLine

*Detects straight lines in the source image.*

### Syntax

```
IppStatusippiHoughLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
IppPointPolar delta, int threshold, IppPointPolar* pLine, int maxLineCount, int*
pLineCount, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).
<i>threshold</i>	Minimum number of points that are required to detect the line.
<i>pLine</i>	Pointer to the destination buffer for lines.
<i>pLineCount</i>	Number of detected lines. If the value is more than <i>maxLineCount</i> , the function returns the <code>ippStsDstSizeLessExpected</code> status.
<i>maxLineCount</i>	Number of elements of the destination buffer.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

In the binarised source image *pSrc*, the function performs detection of the straight line defined by the [equation](#) given at the beginning of section Hough Transform. The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer *pBuffer*, which size should be computed previously using the function [ippiHoughLineGetSize](#).

### Caution

The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value; or if <code>maxLineCount</code> is less than or equal to 0.
ippStsStepErr	Indicates an error condition if <code>srcStep</code> has an illegal value.
ippStsBadArgErr	Indicates an error condition if <code>threshold</code> is less than or equal to 0; or <code>delta.rho</code> is less than 0, or greater than sum of the width and height of the ROI; or <code>delta.theta</code> is less than 0, or greater than $p$ .
ippStsDstSizeLessExpected	Indicates a warning if number of the detected lines is greater than the size of the destination buffer <code>maxLineCount</code> .

## HoughLine\_Region

Detects straight lines with the specified range of parameters in the source image.

---

### Syntax

```
IppStatusippiHoughLine_Region_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize  
roiSize, IppPointPolar* pLine, IppPointPolar dstRoi[2], int maxLineCount, int*  
pLineCount, IppPointPolar delta, int threshold, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ipppvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ipppvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of source image ROI, in pixels.
<i>pLine</i>	Pointer to the destination array of detected lines.
<i>dstRoi</i>	Specifies the range of parameters of straight lines to be detected.
<i>pLineCount</i>	Pointer to the number of detected lines.
<i>delta</i>	Step of discretization ( <code>delta.rho</code> - radial increment, <code>delta.theta</code> - angular increment).
<i>threshold</i>	Minimum number of points that are required to detect the line.
<i>maxLineCount</i>	Maximum number of lines in the destination buffer.
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

In the binary source image *pSrc*, this function performs detection of the straight line defined by the [equation](#) given at the beginning of section Hough Transform. Only lines *line[n]* with the parameters satisfying the following conditions are detected:

```
dstRoi[0].rho ≤ line[n].rho ≤ dstRoi[1].rho;
dstRoi[0].theta ≤ line[n].theta ≤ dstRoi[1].theta;
```

where *n* = 0..*pLineCount*.

The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer *pBuffer*, which size should be computed previously using the function [ippiHoughLineGetSize](#).

---

### Caution

The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

---

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or if <i>maxLineCount</i> is less than or equal to 0.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> has an illegal value.
ippStsBadArgErr	Indicates an error condition if <i>threshold</i> is less than or equal to 0; or <i>delta.rho</i> is less than 0, or greater than sum of the width and height of the ROI; or <i>delta.theta</i> is less than 0, or greater than <i>p</i> ; or some filed of <i>dstRoi[0]</i> is greater than the corresponding filed of <i>dstRoi[1]</i> .
ippStsDstSizeLessExpected	Indicates a warning if number of the detected lines is greater than the size of the destination buffer <i>maxLineCount</i> .

## HoughProbLineGetSize

*Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.*

---

## Syntax

```
IppStatusippiHoughProbLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int*  
pSpecSize, int* pBufSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).
<i>pSpecSize</i>	Size of the <code>IppiHoughProbSpec</code> structure.
<i>pBufSize</i>	Pointer to the size of the working buffer.

## Description

This function operates with ROI.

This function computes the size of the temporary working buffer and the `IppiHoughProbSpec` specification structure for the `ippiHoughProbLine` function.

For an example on how to use this function, see the example provided with the `ippiHoughProbLine` function description.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if <i>pBufSize</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> or <i>delta</i> has a field with a zero or negative value.

## See Also

[Regions of Interest in Intel IPP](#)

[HoughProbLine](#) Detects lines in the source image using the probabilistic Hough transform.

## HoughProbLineInit

*Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.*

## Syntax

```
IppStatus ippiHoughProbLineInit_8u32f_C1R(IppiSize roiSize, IppPointPolar delta,
IppHintAlgorithm hint, IppiHoughProbSpec* pSpec);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Size of the source image ROI, in pixels.
<i>delta</i>	Step of discretization ( <i>delta.rho</i> - radial increment, <i>delta.theta</i> - angular increment).

---

<i>hint</i>	Suggests using specific code for calculations.
<i>pSpec</i>	Pointer to the <code>IppiHoughProbSpec</code> structure.

## Description

This function operates with ROI.

This function initializes the `IppiHoughProbSpec` specification structure that is required for the `ippiHoughProbLine` function.

For an example on how to use this function, see the example provided with the `ippiHoughProbLine` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pBufSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>delta</code> has a field with zero or negative value.

## See Also

[Regions of Interest in Intel IPP](#)

[HoughProbLine](#) Detects lines in the source image using the probabilistic Hough transform.

## HoughProbLine

*Detects lines in the source image using the probabilistic Hough transform.*

---

## Syntax

```
IppStatus ippiHoughProbLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
int threshold, int lineLength, int lineGap, IppiPoint* pLine, int maxLineCount, int*
pLineCount, Ipp8u* pBuffer, const IppiHoughProbSpec* pSpec);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>threshold</i>	Minimum number of points that are required to detect the line.
<i>lineLength</i>	Minimum length of the line.
<i>lineGap</i>	Maximum length of the gap between lines.

<i>pLine</i>	Pointer to the detected line: <i>pLine</i> [2*n] indicates beginning of the line, <i>pLine</i> [2*n+1] indicates end of the line.
<i>pLineCount</i>	Number of detected lines.
<i>maxLineCount</i>	Number of elements in the destination buffer.
<i>pBuffer</i>	Pointer to the working buffer.
<i>pSpec</i>	Pointer to the specification structure.

## Description

This function operates with ROI.

This function detects line segments of the binary *pSrc* image using the probabilistic Hough transform. Before using this function, compute the size of the working buffer and specification structure using the [ippiHoughProbGetSize](#) function and initialize the structure using the [ippiHoughProbLineInit](#) function.

This function implements the probabilistic Hough transform algorithm for line detection, described in [\[Matas00\]](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or if <i>maxLineCount</i> is less than or equal to 0.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> has an illegal value.
<i>ippStsBadArgErr</i>	Indicates an error condition if <i>threshold</i> is less than or equal to 0.
<i>ippStsDstSizeLessExpected</i>	Indicates a warning if number of the detected lines is greater than the size of the destination buffer <i>maxLineCount</i> .

## Example

To better understand usage of the [ippiHoughProbLine](#) function, refer to the `HoughProbLine.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

### Regions of Interest in Intel IPP

[HoughProbLineGetSize](#) Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.

[HoughProbLineInit](#) Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.

## LineSuppressionGetBufferSize

*Computes the size of the external buffer for the [ippiLineSuppression](#) function.*

## Syntax

```
IppStatusippiLineSuppressionGetBufferSize(IppiSize roiSize, IppiMaskSize filterMask,  
Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI, in pixels.
<i>filterMask</i>	Size of the derivative filter aperture. Possible values are <code>ippMskSize3x3</code> and <code>ippMskSize5x5</code> .
<i>avgWndSize</i>	Linear size of a neighborhood block for averaging.
<i>dataType</i>	Data type of the source image.
<i>numChannels</i>	Number of channels in the images. Possible value is 1.
<i>pBufferSize</i>	Pointer to the size, in bytes, of the external work buffer.

## Description

This function calculates the size of the temporary buffer for the [ippiLineSuppression](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>when <code>roiSize</code> is less than, or equal to zero</li> <li>when <code>avgWndSize</code> is equal to zero</li> </ul>
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>filterMask</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error when <code>dataType</code> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when <code>numChannels</code> has an illegal value.

## See Also

[LineSuppression](#) Implements the line suppression algorithm.

## LineSuppression

*Implements the line suppression algorithm.*

## Syntax

```
IppStatusippiLineSuppression_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp8u* pFeature, int featureStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float threshold, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image.								
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.								
<i>pFeature</i>	Pointer to the feature points image.								
<i>featureStep</i>	Distance, in bytes, between the starting points of consecutive lines in the feature points image.								
<i>pDst</i>	Pointer to the destination image.								
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.								
<i>roiSize</i>	Size of the source and destination image ROI, in pixels.								
<i>filterType</i>	Type of the derivative operator. Possible values are: <table border="0"> <tbody> <tr> <td>ippKernelSobel</td><td>Sobel filter</td></tr> <tr> <td>ippKernelScharr</td><td>Scharr filter</td></tr> <tr> <td>ippKernelCentralDiff</td><td>Central differences operator</td></tr> </tbody> </table>	ippKernelSobel	Sobel filter	ippKernelScharr	Scharr filter	ippKernelCentralDiff	Central differences operator		
ippKernelSobel	Sobel filter								
ippKernelScharr	Scharr filter								
ippKernelCentralDiff	Central differences operator								
<i>filterMask</i>	Size of the derivative filter aperture. The list of possible values depends on the <i>filterType</i> value: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Filter Type</th><th>Filter Mask</th></tr> </thead> <tbody> <tr> <td>ippKernelSobel</td><td>ippMskSize3x3, ippMskSize5x5</td></tr> <tr> <td>ippKernelScharr</td><td>ippMskSize3x3</td></tr> <tr> <td>ippKernelCentralDiff</td><td>ippMskSize3x3</td></tr> </tbody> </table>	Filter Type	Filter Mask	ippKernelSobel	ippMskSize3x3, ippMskSize5x5	ippKernelScharr	ippMskSize3x3	ippKernelCentralDiff	ippMskSize3x3
Filter Type	Filter Mask								
ippKernelSobel	ippMskSize3x3, ippMskSize5x5								
ippKernelScharr	ippMskSize3x3								
ippKernelCentralDiff	ippMskSize3x3								
<i>avgWndSize</i>	Linear size of a neighborhood block for averaging.								
<i>threshold</i>	Line suppression threshold.								
<i>borderType</i>	Type of border. Possible values are: <table border="0"> <tbody> <tr> <td>ippBorderConst</td><td>Values of all border pixels are set to a constant.</td></tr> <tr> <td>ippBorderRepl</td><td>Border is replicated from the edge pixels.</td></tr> <tr> <td>ippBorderInMem</td><td>Border is obtained from the source image pixels in memory.</td></tr> </tbody> </table> Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderRepl</code> or <code>ippBorderConst</code> and the following flags: <ul style="list-style-type: none"> <li>• <code>ippBorderInMemTop</code></li> </ul>	ippBorderConst	Values of all border pixels are set to a constant.	ippBorderRepl	Border is replicated from the edge pixels.	ippBorderInMem	Border is obtained from the source image pixels in memory.		
ippBorderConst	Values of all border pixels are set to a constant.								
ippBorderRepl	Border is replicated from the edge pixels.								
ippBorderInMem	Border is obtained from the source image pixels in memory.								

- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

`borderValue`

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

`pBuffer`

Pointer to the work buffer. To calculate the size of the temporary buffer, use the [LineSuppressionGetBufferSize](#) function.

## Description

The `ippiLineSuppression` function implements the line suppression algorithm. This function uses two input images: the original image and feature points image containing the lines (edges) and corners. This function performs the following steps:

1. Computes  $I^{(x, y)}_x$  and  $I^{(x, y)}_y$  gradients for each  $(x, y)$  pixel of the image. The function computes gradients using the derivative operator specified by the `filterType` and `filterMask` parameters.
2. Computes products of the gradients at each  $(x, y)$  pixel of the image:

$$I_{xx}^{(x, y)} = I_x^{(x, y)} * I_x^{(x, y)}, \quad I_{yx}^{(x, y)} = I_{xy}^{(x, y)} = I_x^{(x, y)} * I_y^{(x, y)}, \quad I_{yy}^{(x, y)} = I_y^{(x, y)} * I_y^{(x, y)}$$

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the `avgWndSize` value.

$$S_{xx}^{(x, y)} = \sum_{y'} \sum_{x'} I_{xx}^{(x', y')}, \quad S_{xy}^{(x, y)} = S_{yx}^{(x, y)} = \sum_{y'} \sum_{x'} I_{xy}^{(x', y')}, \quad S_{yy}^{(x, y)} = \sum_{y'} \sum_{x'} I_{yy}^{(x', y')}$$

4. Defines 2x2 gradient covariance matrix  $H^{(x, y)}$  over a rectangular neighborhood block for each  $(x, y)$  pixel of the image.

$$H^{(x, y)} = \begin{pmatrix} S_{xx}^{(x, y)} & S_{xy}^{(x, y)} \\ S_{yx}^{(x, y)} & S_{yy}^{(x, y)} \end{pmatrix}$$

5. For each  $(x, y)$  pixel of the image checks the condition below. If the condition is true, the considered point is not a feature point.

$$\frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 * \lambda_2} = \frac{(tr H^{(x, y)})^2}{\det H^{(x, y)}} > threshold$$

where

- `threshold` is the line suppression threshold value passed to the function
- $\lambda_1 * \lambda_2$  are eigenvalues of the matrix  $H^{(x, y)}$ . If both eigenvalues have large positive values, the point belongs to a corner. If  $\lambda_1$  is much bigger than  $\lambda_2$ , the function clears out the candidate point.

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the `borderType` parameter.

Before using this function, compute the size of the external work buffer using the [LineSuppressionGetBufferSize](#) function.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pSrc</i> , <i>pFeature</i> , <i>pDst</i> , or <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error in the following cases: <ul style="list-style-type: none"> <li>• when <i>roiSize</i> is less than, or equal to zero</li> <li>• when <i>avgWndSize</i> is equal to zero</li> </ul>
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating point images.
ippStsFilterTypeErr	Indicates an error when <i>filterType</i> has an illegal value.
ippStsMaskSizeErr	Indicates an error when <i>filterMask</i> has an illegal value.
ippStsBorderErr	Indicates an error when <i>borderType</i> has an illegal value.
ippStsStepErr	Indicates an error when <i>srcStep</i> or <i>dstStep</i> has a negative value.
ippStsInplaceModeNotSupportedErr	Indicates an error when <i>pFeature</i> and <i>pDst</i> point to the same image.

## Example

To better understand usage of the `ippiLineSuppression` function, refer to the `LineSuppression.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[LineSuppressionGetBufferSize](#) Computes the size of the external buffer for the `ippiLineSuppression` function.

## Distance Transform Functions

---

This section describes the distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

### DistanceTransform

*Calculates distance to the closest zero pixel for all non-zero pixels of source image.*

---

#### Syntax

##### Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

```
IppStatusippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppSize roiSize, Ipp32s* pMetrics);
```

Supported values for mod:

8u_C1R	8u16u_C1R
--------	-----------

```
IppStatusippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppSize roiSize, Ipp32f* pMetrics);
```

```
IppStatusippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppSize roiSize, Ipp32f* pMetrics);
```

### Case 2: In-place operations

```
IppStatusippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppSize
roiSize, Ipp32s* pMetrics);
```

```
IppStatusippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppSize
roiSize, Ipp32s* pMetrics);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination distance image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMetrics</i>	Pointer to the array that specifies used metrics.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3x3 mask and three values for a 5x5 mask.

[Figure "3x3 Mask"](#) shows the result of the distance transform of a 7x7 image with zero point in the center. This example corresponds to a 3x3 mask. Two numbers specify metrics in case of the 3x3 mask:

- distance between two pixels that share an edge,

- distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

### 3x3 Mask

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

Figure “5x5 Mask” shows the distance transform for the same image, but for a 5x5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

### 5x5 Mask

4	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width*pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.
ippStsCoeffErr	Indicates an error condition if at least one element of <i>pMetrics</i> array has zero or negative value.

### Example

To better understand usage of the `ippiDistanceTransform` function, refer to the `DistanceTransform.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

Result:

```
1 2 3 4 5 6 7
1 0 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 0 5 6 7    src
```

```

1 2 3 4 5 6 7
1 2 3 4 5 0 7
1 2 3 4 5 6 7

2 2 2 4 6 6 6
2 0 2 4 4 4 6
2 2 2 2 2 4 6
4 4 2 0 2 4 4    dst
6 4 2 2 2 2 2
6 4 4 4 2 0 2
6 6 6 4 2 2 2

```

## GetDistanceTransformMask

*Returns an optimal mask for a given type of metrics and given mask size.*

### Syntax

```
IppStatusippiGetDistanceTransformMask_<mod>(int kerSize, IppiNorm norm, Ipp<datatype>* pMetrics);
```

Supported values for `mod`:

32s	32f
-----	-----

```
IppStatusippiGetDistanceTransformMask(int maskType, Ipp32f* pMetrics);
```

### Include Files

`ippcv.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>kerSize</code>	Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.
<code>norm</code>	Specifies the type of metrics. Possible values are:
	<code>ippiNormInf(0)</code> $L_{\infty}$ , $\Delta = \max( x_1 - x_2 ,  y_1 - y_2 )$ ,
	<code>ippiNormL1(1)</code> $L_1$ , $\Delta =  x_1 - x_2  +  y_1 - y_2 $ ,
	<code>ippiNormL2(2)</code> $L_2$ , $\Delta = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
<code>maskType</code>	Distance type.
<code>pMetrics</code>	Pointer to the output array to store metrics parameters. The array contains the following number of elements: 2                                  for 3x3 mask, 3                                  for 5x5 mask.

## Description

This function fills up the output array with metrics parameters for the given type of metrics and size of mask. The function returns the following results:

(1, 1)	$L_\infty$ , 3x3 mask,
(1, 2)	$L_1$ , 3x3 mask,
(2, 3)	$L_2$ , 3x3 mask, 32s data type,
(0.955, 1.3693)	$L_2$ , 3x3 mask, 32f data type,
(1, 1, 2)	$L_\infty$ , 5x5 mask,
(1, 2, 3)	$L_1$ , 5x5 mask,
(4, 6, 9)	$L_2$ , 5x5 mask, 32s data type,
(1.0, 1.4, 2.1969)	$L_2$ , 5x5 mask, 32f data type.

For more information, see [Bor86].

---

### NOTE

For compatibility with the previous versions of the library the earlier function `ippiGetDistanceTransformMask` replaced by the function `ippiGetDistanceTransformMask_32f` in the current version is also supported.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pMetrics</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>kerSize</code> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>kerSize</code> or <code>norm</code> has a wrong value.

## FastMarchingGetBufferSize

Computes the size of the working buffer for the peak search.

---

## Syntax

```
IppStatusippiFastMarchingGetBufferSize_8u32f_C1R(IppiSize roiSize, int* pBufferSize);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>roiSize</i>	Maximum image size (in pixels).
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiFastMarching` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width that is equal to or less than the parameter `roiSize` specified for the function `ippiFastMarching`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> is less than 1.

## FastMarching

*Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.*

## Syntax

```
IppStatusippiFastMarching_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f radius, Ipp8u* pBuffer);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>radius</i>	Radius of the neighborhood of the marked area.
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see Regions of Interest in Intel IPP ).

This function computes the distance from the closest zero pixel to each image pixel according to the Fast Marching Method (FMM) [ [Telea04](#) ]. The FMM distance for area  $\Omega$  with the border  $\partial\Omega$  is a solution of the equations:

$$(|\nabla T(x, y)| = 1), \{x, y\} \in \Omega$$

$$(T(x, y) = 0), \{x, y\} \in \partial\Omega$$

The resulting distance complies with the equation

$$T(x, y) = \min \left( \frac{T(u_1, v_1) + T(u_2, v_2) + \sqrt{2 - (T(u_1, v_1) - T(u_2, v_2))^2}}{2}, \min(T(u_1, v_1), T(u_2, v_2)) + 1 \right)$$

Here  $\{u_1, v_1\}$  and  $\{u_2, v_2\}$  are coordinates for pair of pixels adjacent to the pixel with  $\{x, y\}$  coordinates.

The area  $\Omega$  is defined by the non-zero pixel of the image  $pSrc$ . If  $radius$  is positive, then the FMM distance with the negative sign is calculated in Euclidean  $radius$ -neighborhood of  $\Omega$ .

The function requires the working buffer  $pBuffer$  whose size should be computed by the function [FastMarchingGetBufferSize](#) beforehand.

[Figure "Result of the FFM Method](#) shows the result of the fast marching method for the 7x9 image with centered 3x5 non-zero mask and  $radius=1$ .

## Result of the FFM Method

0.0000	0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000	0.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
-1.0000	-1.e-10	0.9659	1.6730	1.9579	1.6730	0.9659	-1.e-10	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if $roiSize$ has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if $srcStep$ or $dstStep$ is less than $roiSize.width * <pixelSize>$ .
ippStsNotEvenStepErr	Indicates an error condition if the step value is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if $radius$ is negative.

## TrueDistanceTransformGetBufSize

*Calculates the size of the temporary working buffer for the function `ippiTrueDistanceTransform`.*

### Syntax

```
IppStatus ippiTrueDistanceTransformGetBufferSize_8u32f_C1R(IppiSize roiSize, int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

*roiSize* Size of the image ROI in pixels.

*pBufferSize* Pointer to the computed size of the buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the size of the work buffer required for the [TrueDistanceTransform](#) function.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is <i>NULL</i> .
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width*pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition if step value is not divisible by 2 for <i>16u</i> images, and by 4 for <i>32f</i> images.

## TrueDistanceTransform

*Calculates the Euclidian distance to the closest zero pixel for all non-zero pixels of the source image.*

### Syntax

```
IppStatus ippiTrueDistanceTransform_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination distance image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>pBuffer</code>	Pointer to the temporary working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the Euclidian distance to the closest zero pixel for all non-zero pixels of the source image [\[Felz04\]](#).

The figure below shows the result of the integer version of the true distance transform of a 7x7 image with zero point in the center and with the scale factor -5.

136	115	101	96	101	115	136
115	91	72	64	72	91	115
101	72	45	36	45	72	101
96	64	36	0	36	64	96
101	72	45	36	45	72	101
115	91	72	64	72	91	115
136	115	101	96	101	115	136

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if step value is not divisible by 2 for <code>16u</code> images, and by 4 for <code>32f</code> images.

## Image Gradients

---

## GradientColorToGray

*Converts a color gradient image to grayscale.*

### Syntax

```
IppStatusippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
```

Supported values for mod:

8u\_C3C1R      16u\_C3C1R      32f\_C3C1R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>norm</i>	Type of norm to form the mask for dilation; following values are possible: ippiNormInf      Infinity norm. ippiNormL1      L1 norm. ippiNormL2      L2 norm.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates the grayscale gradient image *pDst* from the source three-channel gradient image *pSrc*. The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

$$dst_{i,j} = \begin{cases} \max\{|src_{i,j,0}|, |src_{i,j,1}|, |src_{i,j,2}|\} & \text{norm} = \text{ippiNormInf} \\ |src_{i,j,0}| + |src_{i,j,1}| + |src_{i,j,2}| & \text{norm} = \text{ippiNormL1} \\ \sqrt{|src_{i,j,0}|^2 + |src_{i,j,1}|^2 + |src_{i,j,2}|^2} & \text{norm} = \text{ippiNormL2} \end{cases}$$

For integer flavors the result is scaled to the full range of the destination data type.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if <i>norm</i> has an illegal value.

## GradientVectorGetBufferSize

Computes the size of the work buffer for the  
ippiGradientVector{Sobel|Scharr|Prewitt}  
functions.

---

### Syntax

```
IppStatusippiGradientVectorGetBufferSize(IppiSize roiSize, IppiMaskSize mask,  
IppDataType dataType, int numChannels, int* pBufferSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>roiSize</i>	Size of the destination ROI in pixels.
<i>mask</i>	Predefined mask of <a href="#">IppiMaskSize</a> type.
<i>dataType</i>	Data type of the source image.
<i>numChannels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the computed size of the external work buffer.

### Description

The `ippiGradientVectorGetBufferSize` function computes the size (in bytes) of the external work buffer needed for the `ippiGradientVector{Sobel|Scharr|Prewitt}` functions and stores the result in the *pBufferSize* parameter.

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when one of the fields of <i>roiSize</i> has a zero or negative value.
ippStsMaskSizeErr	Indicates an error when <i>mask</i> has an illegal value.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsNumChannelsErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

### Structures and Enumerators

[GradientVectorPrewitt](#) Computes gradient vectors of an image using the Prewitt operator.

[GradientVectorScharr](#) Computes gradient vectors of an image using the Scharr operator.

[GradientVectorSobel](#) Computes gradient vectors of an image using the Sobel operator.

## GradientVectorPrewitt

*Computes gradient vectors of an image using the Prewitt operator.*

### Syntax

```
IppStatusippiGradientVectorPrewitt_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R 16s32f\_C1R 16u32f\_C1R 32f\_C1R

```
IppStatusippiGradientVectorPrewitt_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype>
borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C3C1R 16s32f\_C3C1R 16u32f\_C3C1R 32f\_C3C1R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pGx</i>	Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
<i>gxStep</i>	Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
<i>pGy</i>	Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
<i>gyStep</i>	Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
<i>pMag</i>	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
<i>magStep</i>	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
<i>pAngle</i>	Pointer to the destination image ROI for the angle (in polar gradient form).
<i>angleStep</i>	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
<i>dstRoiSize</i>	Size of the source and destination image ROI in pixels.
<i>maskSize</i>	Predefined mask of <a href="#">IppiMaskSize</a> type.
<i>normType</i>	Normalization mode of <a href="#">IppNormType</a> type.
<i>borderType</i>	Type of border. Possible values are:
	<i>ippBorderConst</i> Values of all border pixels are set to constant.
	<i>ippBorderRepl</i> Border is replicated from the edge pixels.
	<i>ippBorderInMem</i> Border is obtained from the source image pixels in memory.
	<i>ippBorderMirro</i> r Border pixels are mirrored from the source image boundary pixels.
<i>borderValue</i>	Constant value to assign to pixels in the constant border (not applicable for other border types).
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

Before using this function, compute the size of the work buffer using the [ippiGradientVectorGetBufferSize](#) function.

### NOTE

Any of the *pGx*, *pGy*, *pMag*, and *pAngle* output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

### Single-channel image (C1) input :

If input is a single-channel image, the [ippiGradientVectorPrewitt](#) function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (*pGx* and *pGy*) and/or polar (*pMag* and *pAngle*) form, or any combination of these possible outputs.

Cartesian projections  $G_x$  and  $G_y$  are stored in the *pGx* and *pGy* buffers, respectively. The formulas below describe the algorithm for the 3x3 Prewitt operator:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * A$$

where

- $A$  is the source image
- $*$  means two-dimensional convolution
- $G_x$  and  $G_y$  are X and Y components of the gradient

The magnitude of the gradient is computed according to the *normType* value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between  $G_x$  and  $G_y$  is computed by the formula:

$$\text{angle} = \arctan \frac{G_y}{G_x}$$

### Color image (C3) input :

If input is a color image, the `ippiGradientVectorPrewitt` function computes the spatial image derivatives  $G_x$  and  $G_y$  for each channel of the image using the specified differential operator. For each pixel  $(x, y)$  this function chooses the derivatives for which  $L2(G_x, G_y)$  is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the [GradientVectorSobel](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> or <code>pBuffer</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>maskSize</code> has an illegal value.
<code>ippStsStepErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> has a zero or negative value</li> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)</li> </ul>
<code>ippStsBadArgErr</code>	Indicates an error when <code>normType</code> has an illegal value.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an incorrect value.

## See Also

[Structures and Enumerators](#)

[Regions of Interest in Intel IPP](#)

[GradientVectorGetBufferSize](#) Computes the size of the work buffer for the `ippiGradientVector{Sobel|Scharr|Prewitt}` functions.

## GradientVectorScharr

*Computes gradient vectors of an image using the Scharr operator.*

---

### Syntax

```
IppStatus ippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s\_C1R 16s32f\_C1R 16u32f\_C1R 32f\_C1R

```
IppStatusippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C3C1R 16s32f\_C3C1R 16u32f\_C3C1R 32f\_C3C1R

## Include Files

ippi.h

## Domain Dependencies

Headers:ippcore.h,ippvm.h,ipps.h

Libraries:ippcore.lib,ippvm.lib,ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pGx</i>	Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
<i>gxStep</i>	Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
<i>pGy</i>	Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
<i>gyStep</i>	Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
<i>pMag</i>	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
<i>magStep</i>	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
<i>pAngle</i>	Pointer to the destination image ROI for the angle (in polar gradient form).
<i>angleStep</i>	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
<i>dstRoiSize</i>	Size of the source and destination image ROI in pixels.
<i>maskSize</i>	Predefined mask of <a href="#">IppiMaskSize</a> type.
<i>normType</i>	Normalization mode of <a href="#">IppNormType</a> type.

<i>borderType</i>	Type of border. Possible values are:
ippBorderConst	Values of all border pixels are set to constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.
ippBorderMirro r	Border pixels are mirrored from the source image boundary pixels.
<i>borderValue</i>	Constant value to assign to pixels in the constant border (not applicable for other border types).
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

Before using this function, compute the size of the work buffer using the [ippiGradientVectorGetBufferSize](#) function.

### NOTE

Any of the *pGx*, *pGy*, *pMag*, and *pAngle* output parameters can be `NULL`. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

### Single-channel image (C1) input :

If input is a single-channel image, the [ippiGradientVectorScharr](#) function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (*pGx* and *pGy*) and/or polar (*pMag* and *pAngle*) form, or any combination of these possible outputs.

Cartesian projections  $G_x$  and  $G_y$  are stored in the *pGx* and *pGy* buffers, respectively. The formulas below describe the algorithm for the 3x3 Scharr operator:

$$G_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} * A \quad G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} * A$$

where

- $A$  is the source image
- $*$  means two-dimensional convolution
- $G_x$  and  $G_y$  are X and Y components of the gradient

The magnitude of the gradient is computed according to the *normType* value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between  $G_x$  and  $G_y$  is computed by the formula:

$$\text{angle} = \arctan \frac{G_y}{G_x}$$

### Color image (C3) input :

If input is a color image, the `ippiGradientVectorScharr` function computes the spatial image derivatives  $G_x$  and  $G_y$  for each channel of the image using the specified differential operator. For each pixel  $(x, y)$  this function chooses the derivatives for which  $L2(G_x, G_y)$  is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the `GradientVectorSobel` function description.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> or <code>pBuffer</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>maskSize</code> has an illegal value.
<code>ippStsStepErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> has a zero or negative value</li> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)</li> </ul>
<code>ippStsBadArgErr</code>	Indicates an error when <code>normType</code> has an illegal value.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an incorrect value.

### See Also

[Structures and Enumerators](#)  
[Regions of Interest in Intel IPP](#)

[GradientVectorGetBufferSize](#) Computes the size of the work buffer for the `ippiGradientVector{Sobel|Scharr|Prewitt}` functions.

## GradientVectorSobel

*Computes gradient vectors of an image using the Sobel operator.*

---

### Syntax

```
IppStatusippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C1R 16s32f\_C1R 16u32f\_C1R 32f\_C1R

```
IppStatusippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype>
borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s\_C3C1R 16s32f\_C3C1R 16u32f\_C3C1R 32f\_C3C1R

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

*pSrc*

Pointer to the source image ROI.

*srcStep*

Distance, in bytes, between the starting points of consecutive lines in the source image.

*pGx*

Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).

*gxStep*

Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.

*pGy*

Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).

*gyStep*

Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.

---

<i>pMag</i>	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
<i>magStep</i>	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
<i>pAngle</i>	Pointer to the destination image ROI for the angle (in polar gradient form).
<i>angleStep</i>	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
<i>dstRoiSize</i>	Size of the source and destination image ROI in pixels.
<i>maskSize</i>	Predefined mask of <a href="#">IppiMaskSize</a> type.
<i>normType</i>	Normalization mode of <a href="#">IppNormType</a> type.
<i>borderType</i>	Type of border. Possible values are:  ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl        Border is replicated from the edge pixels. ippBorderInMem       Border is obtained from the source image pixels in memory. ippBorderMirro r                    Border pixels are mirrored from the source image boundary pixels.
<i>borderValue</i>	Constant value to assign to pixels in the constant border (not applicable for other border types).
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

Before using this function, compute the size of the work buffer using the [GradientVectorGetBufferSize](#) function.

---

### NOTE

Any of the *pGx*, *pGy*, *pMag*, and *pAngle* output parameters can be `NULL`. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

### Single-channel image (C1) input :

If input is a single-channel image, the [ippiGradientVectorSobel](#) function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (*pGx* and *pGy*) and/or polar (*pMag* and *pAngle*) form, or any combination of these possible outputs.

Cartesian projections  $G_x$  and  $G_y$  are stored in the  $pGx$  and  $pGy$  buffers, respectively. The formulas below describe the algorithm for the 3x3 Sobel operator:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

for the 5x5 Sobel operator:

$$G_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 0 & 8 & 12 & 8 & 2 \\ 2 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A$$

where

- $A$  is the source image
- $*$  means two-dimensional convolution
- $G_x$  and  $G_y$  are X and Y components of the gradient

The magnitude of the gradient is computed according to the *normType* value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between  $G_x$  and  $G_y$  is computed by the formula:

$$\text{angle} = \arctan \frac{G_y}{G_x}$$

**Color image (C3) input :**

If input is a color image, the `ippiGradientVectorSobel` function computes the spatial image derivatives  $G_x$  and  $G_y$  for each channel of the image using the specified differential operator. For each pixel  $(x, y)$  this function chooses the derivatives for which  $L2(G_x, G_y)$  is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSrc</code> or <code>pBuffer</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when one of the fields of <code>dstRoiSize</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error when <code>maskSize</code> has an illegal value.
<code>ippStsStepErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> has a zero or negative value</li> <li>• <code>srcStep</code>, <code>gxStep</code>, <code>gyStep</code>, <code>magStep</code>, or <code>angleStep</code> is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)</li> </ul>
<code>ippStsBadArgErr</code>	Indicates an error when <code>normType</code> has an illegal value.
<code>ippStsBorderErr</code>	Indicates an error when <code>borderType</code> has an incorrect value.

## Example

To better understand usage of the `ippiGradientVectorSobel` function, refer to the `GradientVectorSobel1.c` and `GradientVectorSobel2.c` examples in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Structures and Enumerators](#)

[Regions of Interest in Intel IPP](#)

[GradientVectorGetBufferSize](#) Computes the size of the work buffer for the `ippiGradientVector{Sobel|Scharr|Prewitt}` functions.

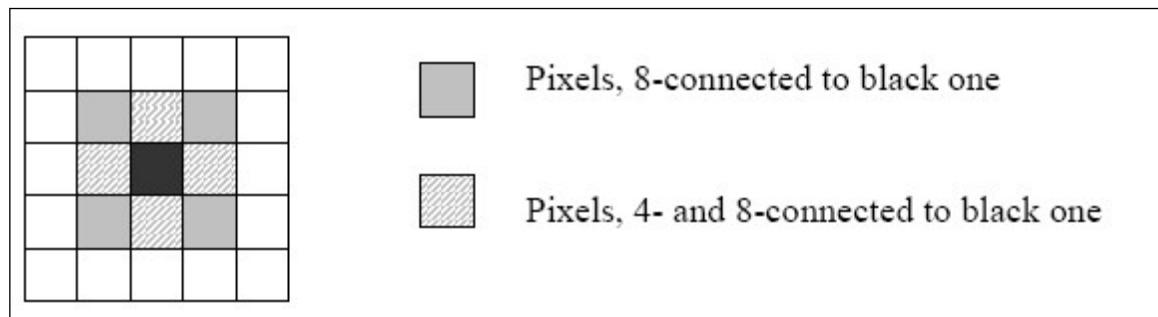
## Flood Fill Functions

This section describes functions performing flood filling of connected areas. *Flood filling* means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point ("seed") and continues until it reaches the image ROI boundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:

- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes `4Con`, or

- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes 8Con.

### Pixels Connectivity Patterns



These functions can be used for:

- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

## FloodFillGetSize

*Calculates size of temporary buffer for flood filling operation.*

### Syntax

```
IppStatusippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

*roiSize* Size of the source image ROI in pixels.

*pBufSize* Pointer to the variable that returns the size of the temporary buffer.

### Description

This function calculates the size of the temporary buffer to be used by the function [ippiFloodFill](#).

### Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error condition if *pBufSize* pointer is NULL.

ippStsSizeErr Indicates an error condition if *roiSize* has a field with zero or negative value.

## FloodFillGetSize\_Grad

*Calculates size of temporary buffer for the gradient flood filling.*

### Syntax

```
IppStatusippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufSize</i>	Pointer to the variable that returns the size of the temporary buffer.

### Description

This function calculates the size of the temporary buffer to be used by the function [ippiFloodFill\\_Grad](#).

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if <i>pBufSize</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## FloodFill

*Performs flood filling of connected area.*

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatusippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize  
roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u*  
pBuffer);
```

```
IppStatusippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize  
roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u*  
pBuffer);
```

Supported values for *mod*:

8u\_C1IR    16u\_C1IR    32s\_C1IR    32f\_C1IR

## Case 2: Operations on three-channel data

```
IppStatusippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize  
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u*  
pBuffer);
```

```
IppStatusippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize  
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u*  
pBuffer);
```

Supported values for mod:

8u\_C3IR      16u\_C3IR      32f\_C3IR

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (for the in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels whose pixel values are equal to the value in the *seed* point. Values of these pixels is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [ippiFloodFillGetSize](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See [Figure Pixels Connectivity Patterns](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
--------------------	--

---

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>imageStep</i> is less than <i>pRoiSize.width*pixelsize</i> .
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the <i>seed</i> point is out of ROI.

## Example

To better understand usage of the `ippiFloodFill` function, refer to the `FloodFill.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## FloodFill\_Grad

*Performs gradient flood filling of connected area on an image.*

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C1IR      16u\_C1IR      32f\_C1IR

#### Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C3IR      16u\_C3IR      32f\_C3IR

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** ippcore.lib, ippvm.lib, ipsps.lib,ippi.lib

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfy the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where *v<sub>0</sub>* is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and *d<sub>lw</sub>*, *d<sub>up</sub>* are *minDelta*, *maxDelta*, respectively. Values of these pixels are set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [ippiFloodFillGetSize\\_Grad](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure Pixels Connectivity Patterns](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>imageStep</i> is less than <i>pRoysize.width*pixelSize</i> .

---

ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the <i>seed</i> point is out of ROI.

## Example

To better understand usage of the `ippiFloodFill_Grad` function, refer to the `FloodFill_Grad.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## FloodFill\_Range

*Performs flood filling of pixels with values in the specified range in the connected area on an image.*

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C1IR      16u\_C1IR      32f\_C1IR

#### Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u\_C3IR      16u\_C3IR      32f\_C3IR

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) (see [Flood Filling Functions](#)) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfy the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where *v*<sub>0</sub> is the pixel value of the *seed* point, and *d*<sub>lw</sub>, *d*<sub>up</sub> are *minDelta*, *maxDelta*, respectively. Values of these pixels are set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [\*ippiFloodFillGetSize\\_Grad\*](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure Pixels Connectivity Patterns](#).

## Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>imageStep</i> is less than <i>pRoiSize.width*pixelsize</i> .

---

<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <code>seed</code> point is out of ROI.

## Motion Analysis and Object Tracking

---

### FGMMGetBufferSize

*Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.*

---

### Syntax

```
IppStatusippiFGMMGetBufferSize_8u_C3R(IppiSize roi, int maxNGauss, int* pSpecSize);
```

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>roi</code>	Size of the source image ROI, in pixels.
<code>maxNGauss</code>	Maximal size of the Gaussian mixture components.
<code>pSpecSize</code>	Pointer to the size of the <code>IppFGMMSpec_8u_C3R</code> structure.

### Description

This function operates with ROI.

This function computes the size of the `IppFGMMSpec_8u_C3R` structure for the [FGMMForeground](#) and [FGMMBackground](#) functions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pSpecSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>roi</code> is less than, or equal to zero.
<code>ippStsBadArgErr</code>	Indicates an error when <code>maxNumGauss</code> is less than, or equal to zero.

### See Also

[Regions of Interest in Intel IPP](#)

[FGMMForeground](#) Performs the Gaussian mixture model foreground subtraction.

[FGMMBackground](#) Returns the updated background image.

## FGMMInit

*Initializes the state structure for the Gaussian mixture model foreground/background subtraction.*

### Syntax

```
IppStatusippiFGMMInit_8u_C3R(IppiSize roi, int maxNGauss, IppFGMModel* pModel,  
IppFGMMState_8u_C3R* pState);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.i.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.i.lib

### Parameters

<i>roi</i>	Size of the source image ROI, in pixels.
<i>maxNGauss</i>	Maximal size of the Gaussian mixture components.
<i>pModel</i>	Pointer to the <code>IppFGMModel</code> structure containing parameters for the model. If <i>pModel</i> is NULL, the default parameters are applied.
<i>pState</i>	Pointer to the <code>IppFGMMState_8u_C3R</code> state structure.

### Description

This function operates with ROI.

This function initializes the `IppFGMMState_8u_C3R` state structure for the [FGMMForeground](#) and [FGMMBackground](#) functions.

Before using this function, you need to compute the size of the state structure using the [FGMMGetBufferSize](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pModel</i> or <i>pState</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roi</i> is less than, or equal to zero.
<code>ippStsBadArgErr</code>	Indicates an error when <i>maxNGauss</i> is less than, or equal to zero.

### See Also

[Regions of Interest in Intel IPP](#)

[FGMMForeground](#) Performs the Gaussian mixture model foreground subtraction.

[FGMMBackground](#) Returns the updated background image.

[FGMMGetBufferSize](#) Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

## FGMMForeground

*Performs the Gaussian mixture model foreground subtraction.*

### Syntax

```
IppStatusippiFGMMForeground_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize roi, IppFGMMState_8u_C3R* pState, IppFGMMModel* pModel, double learning_rate);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pDst</i>	Pointer to the one-channel Ipp8u mask of foreground.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roi</i>	Size of the source image ROI, in pixels.
<i>learning_rate</i>	Speed of algorithm learning.
<i>pState</i>	Pointer to the IppFGMMState_8u_C3R state structure.
<i>pModel</i>	Pointer to the IppFGMMModel structure containing parameters for the model. If <i>pModel</i> is NULL, the parameters are the same as in a previous call.

### Description

This function operates with ROI.

This function implements the Gaussian mixture model foreground subtraction described in [ZIVKOVIC04]. The foreground mask is stored in *pDst*.

Before using this function, you need to compute the size of the IppFGMMState\_8u\_C3R state structure and initialize the structure using the [FGMMGetBufferSize](#) and [FGMMInit](#) functions, respectively.

For an example on how to use this function, refer to the example provided with the [FGMMBackground](#) function description.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when <i>pSrc</i> , <i>pDst</i> , <i>pModel</i> , or <i>pState</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error when <i>roi</i> is less than, or equal to zero.

<i>ippStsStepErr</i>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than, or equal to zero.
----------------------	--

## See Also

[Regions of Interest in Intel IPP](#)

[FGMMGetBufferSize](#) Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

[FGMMInit](#) Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

[FGMMBackground](#) Returns the updated background image.

## FGMMBackground

Returns the updated background image.

---

### Syntax

```
IppStatusippiFGMMBackground_8u_C3R(Ipp8u* pDst, int dstStep, IppiSize roi,  
IppFGMMState_8u_C3R* pState);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pDst</i>	Pointer to the three-channel background image.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>roi</i>	Size of the source image ROI, in pixels.
<i>pState</i>	Pointer to the <code>IppFGMMState_8u_C3R</code> state structure.

### Description

This function implements the Gaussian mixture model background subtraction described in [ZIVKOVIC04]. The function returns the three-channel `Ipp8u` background image.

Before using this function, you need to compute the size of the `IppFGMMState_8u_C3R` state structure and initialize the structure using the [FGMMGetBufferSize](#) and [FGMMInit](#) functions, respectively.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when <i>pDst</i> or <i>pState</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error when <i>roi</i> is less than, or equal to zero.
<i>ippStsStepErr</i>	Indicates an error when <i>dstStep</i> is less than, or equal to zero.

## Example

To better understand usage of the `ippiFGMMBackground` function, refer to the `FGMMBackground.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[FGMMGetBufferSize](#) Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

[FGMMInit](#) Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

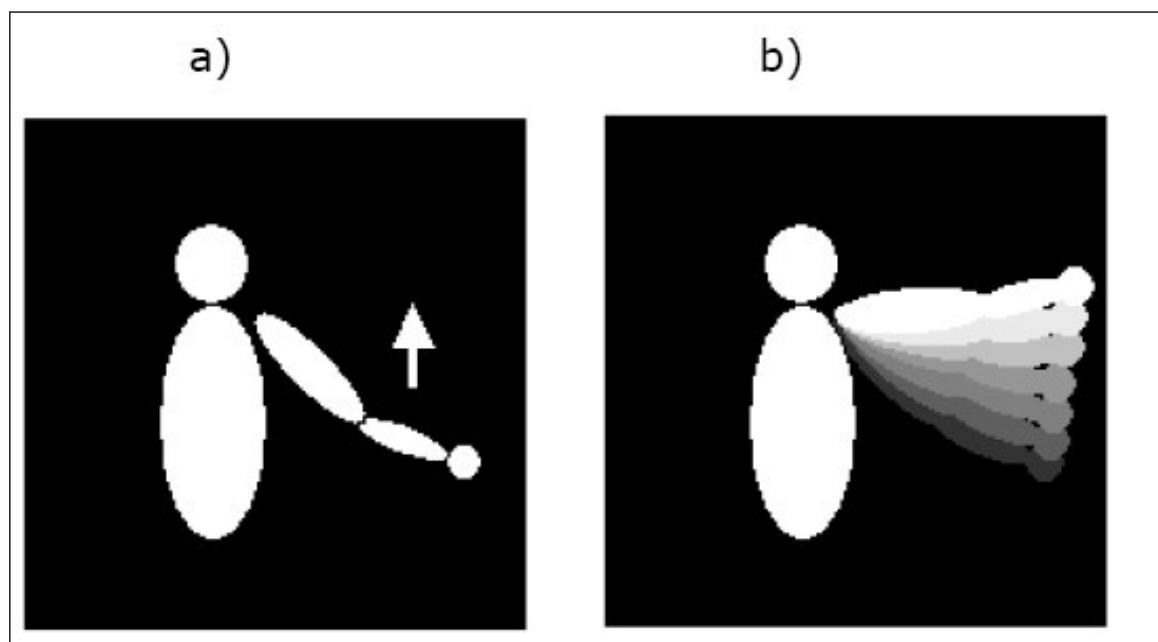
## Motion Template Functions

This section describes a motion templates function. This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [ Davis97], and [Davis99]. The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be 8u, 8s, or 32f.

## Motion Representation

[Figure Motion Image History](#) (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a “layered history” of the resulting motion. Typically, this “highest value” is just a floating-point timestamp of time since the code has been running in milliseconds. [Figure Motion Image History](#) (b) shows the result that may be called the *Motion History Image (MHI)*. The MHI in Figure Motion Image History represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

## Motion Image History



The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

## Updating MHI Images

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

### UpdateMotionHistory

*Updates motion history image using motion silhouette at given timestamp.*

---

#### Syntax

```
IppStatusippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>* pSilhouette, int
silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f timeStamp, Ipp32f
mhiDuration);
```

Supported values for `mod`:

8u32f_C1IR	16u32f_C1IR	32f_C1IR
------------	-------------	----------

#### Include Files

`ippcv.h`

#### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

#### Parameters

<code>pSilhouette</code>	Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>silhStep</code>	Distance in bytes between starts of consecutive lines in the silhouette image.
<code>pMhi</code>	Pointer to the motion history image which is both an input and output parameter.
<code>mhiStep</code>	Distance in bytes between starts of consecutive lines in the motion history image.
<code>timeStamp</code>	Timestamp in milliseconds.
<code>mhiDuration</code>	Threshold for MHI pixels. MHI motions older than this threshold are deleted.

#### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function updates the motion history image. It sets MHI pixels to the current `timeStamp` value, if their values are non-zero.

The function deletes MHI pixels, if their values are less than the `mhiDuration` timestamp, that is, the pixels are “old.”

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>mhiStep</i> or <i>silhStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if the step value is not divisible by 2 for <i>16u</i> images, and by 4 for <i>32f</i> images.
ippStsOutOfRangeErr	Indicates an error when <i>mhiDuration</i> is negative.

## Optical Flow

This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [ [Bou99](#) ].

The optical flow between two images is generally defined as an apparent motion of image brightness. Let  $I(x, y, t)$  be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates

$$u = \frac{\partial x}{\partial t}, v = \frac{\partial y}{\partial t}$$

can be found from so called *optical flow constraint equation*:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and finds the approximate solution of the above equation using the least square method.

## OpticalFlowPyrLKGetSize

*Computes the size of the pyramidal optical flow state structure.*

### Syntax

```
IppStatusippiOpticalFlowPyrLKGetSize(int winSize, IppiSize roiSize, IppDataType
dataType, IppHintAlgorithm hint, int* pStateSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>winSize</i>	Size of the search window of each pyramid level.
----------------	--

<i>roiSize</i>	Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
<i>dataType</i>	Data type of the image. Possible values: <code>ipp8u</code> , <code>ipp16u</code> , or <code>ipp32f</code> .
<i>hint</i>	Option to select the algorithmic implementation of the transform function.
<i>pStateSize</i>	Pointer to the size of the state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the *pState* structure that is required to calculate the optical flow between two images in the centered window of size *winSize*\**winSize* using the pyramidal Lucas-Kanade [Bou99] algorithm. The *hint* argument specifies the computation algorithm. The *pState* structure is used by the [ippiOpticalFlowPyrLK](#) function and can be applied to process images with size not greater than *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pStateSize</i> is NULL.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value; or when <i>winSize</i> is less than, or equal to zero.

## See Also

[Regions of Interest in Intel IPP](#)

[OpticalFlowPyrLK](#) Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

[OpticalFlowPyrLKInit](#) Initializes the state structure for optical flow calculation.

## OpticalFlowPyrLKInit

*Initializes the state structure for optical flow calculation.*

---

## Syntax

```
IppStatusippiOpticalFlowPyrLKInit_<mod>(IppiOptFlowPyrLK_<mod>** ppState, IppiSize  
roiSize, int winSize, IppHintAlgorithm hint, Ipp8u* pStateBuf);
```

Supported values for mod:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
---------------------	----------------------	----------------------

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>ppState</i>	Double pointer to the optical flow state structure to be initialized.
<i>roiSize</i>	Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
<i>winSize</i>	Size of the search window of each pyramid level.
<i>hint</i>	Option to select the algorithmic implementation of the transform function.
<i>pStateBuf</i>	Pointer to the work buffer for the state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the *ppState* structure that is required to calculate the optical flow between two images in the centered window of size *winSize*\**winSize* using the pyramidal Lucas-Kanade [Bou99] algorithm. The *hint* argument specifies the computation algorithm. The *ppState* structure is used by the [ippiOpticalFlowPyrLK](#) function and can be applied to process images with size not greater than *roiSize*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>ppState</i> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value; or when <i>winSize</i> is less than, or equal to zero.

## See Also

[Regions of Interest in Intel IPP](#)

[OpticalFlowPyrLK](#) Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

[OpticalFlowPyrLKGGetSize](#) Computes the size of the pyramidal optical flow state structure.

## OpticalFlowPyrLK

*Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.*

## Syntax

```
IppStatusippiOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2, const
IppiPoint_32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f* pError, int
numFeat, int winSize, int maxLev, int maxIter, Ipp32f threshold,
IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for *mod*:

`8u_C1R`      `16u_C1R`      `32f_C1R`

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pPyr1</i>	Pointer to the ROI in the first image pyramid structure.
<i>pPyr2</i>	Pointer to the ROI in the second image pyramid structure.
<i>pPrev</i>	Pointer to the array of initial coordinates of the feature points.
<i>pNext</i>	Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.
<i>pStatus</i>	Pointer to the array of result indicators.
<i>pError</i>	Pointer to the array of differences between neighborhoods of old and new point positions.
<i>numFeat</i>	Number of feature points.
<i>winSize</i>	Size of the search window of each pyramid level.
<i>maxLev</i>	Pyramid level to start the operation.
<i>maxIter</i>	Maximum number of algorithm iterations for each pyramid level.
<i>threshold</i>	Threshold value.
<i>pState</i>	Pointer to the pyramidal optical flow structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function implements the iterative version of the Lucas-Kanade algorithms [Bou99]. It computes with sub-pixel accuracy new coordinates of the *numFeat* feature points of two images at time *t* and *t+dt*. Their initial coordinates are places in the *pPrev* array. Computed values of new coordinates of the feature points are stored in the array *pNext*, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the *pNext* array contains the same initial coordinates as the *pPrev* array.

*pStatus* and *pError* are arrays of size *numFeat* with the respective data type.

The images are presented by the pyramid structures *pPyr1* and *pPyr2* respectively (see description of the [PyramidGetSize](#) and [ippiPyramidInit](#) functions for more details). These structures should be initialized by calling the function [PyramidGetSize](#) and [ippiPyramidInit](#) functions beforehand. The function uses the pyramidal optical flow structure *pState* that also should be previously initialized using [OpticalFlowPyrLKGetSize](#) and [OpticalFlowPyrLKInit](#).

The function starts operation on the highest pyramid level (smallest image) that is specified by the *maxLev* parameter in the centered search window which size *winSize* could not exceed the corresponding value *winSize* that is specified in [OpticalFlowPyrLKGetSize](#) and [OpticalFlowPyrLKInit](#). The operation for *i*-th feature point on the given pyramid level finishes if:

- New position of the point is found:

$$\left( \sqrt{dx^2 + dy^2} < threshold \right),$$

- Specified number of iteration *maxIter* is performed
- Intersection between the pyramid layer and the search window became empty

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level. For zero level or for third case the operation stops, the number of the corresponding level is written to the `pStatus[i]` element, the new coordinates are scaled to zero level and are written to `pNext[i]`. The square root of the average squared difference between neighborhoods of old and new positions is written to `pError[i]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>numFeat</code> or <code>winSize</code> has zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>maxLev</code> or <code>threshold</code> has negative value, or <code>maxIter</code> has zero or negative value.

## Example

To better understand usage of the `ippiOpticalFlowPyrLK` function, refer to the `OpticalFlowPyrLK.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in optical flow calculations.

---

### NOTE

All universal pyramid functions use the mirrored border.

---

[Example](#) shows how to build pyramids and calculate the optical flow for two images.

## PyramidGetSize

*Computes the size of the pyramid structure and the size of the temporary buffer for the `ippiPyramidInit` function.*

---

## Syntax

```
IppStatusippiPyramidGetSize(int* pPyrSize, int* pBufSize, int level, IppiSize roiSize,
Ipp32f rate);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pPyrSize</i>	Pointer to the size of the pyramid structure.
<i>pBufSize</i>	Pointer to the size of the external work buffer for pyramid processing.
<i>level</i>	Maximum value for the pyramid level.
<i>roiSize</i>	Size of zero level image ROI, in pixels.
<i>rate</i>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).

## Description

This function computes the size of the pyramid structure and the size of the temporary buffer, in bytes, for the [ippiPyramidInit](#) function. For an example on how to use this function, refer to the example provided with the [ippiPyramidLayerDown](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error when <i>level</i> is equal to or less than 0, or when <i>rate</i> is out of the range.

## See Also

[PyramidInit](#) Initializes the pyramid structure and calculates the ROI size for pyramid layers.  
[PyramidLayerDown](#) Creates a lower pyramid layer.

## PyramidInit

*Initializes the pyramid structure and calculates the ROI size for pyramid layers.*

---

## Syntax

```
IppStatusippiPyramidInit(IppiPyramid** pPyr, int level, IppiSize roiSize, Ipp32f rate,
Ipp8u* pPyrBuffer, Ipp8u* pBuffer);
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pPyr</i>	Pointer to the pointer to the pyramid structure.
<i>level</i>	Maximum value for the pyramid level.
<i>roiSize</i>	Size of zero level image ROI, in pixels.

---

<i>rate</i>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).
<i>pPyrBuffer</i>	Pointer to the buffer for the pyramid structure initialization.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function initializes the structure for pyramid with  $(\text{level}+1)$  levels. This structure is used by the [ippiOpticalFlowPyrLK](#) function for optical flow calculation. The [IppiPyramid](#) structure contains the following fields:

<i>pImage</i>	Pointer to the array of $(\text{level}+1)$ layer images.
<i>pStep</i>	Pointer to the array of $(\text{level}+1)$ image row step values.
<i>pRoi</i>	Pointer to the array of $(\text{level}+1)$ layer image ROIs.
<i>pRate</i>	Pointer to the array of $(\text{level}+1)$ ratios of $i$ -th levels to the zero level ( $\text{rate}^{-i}$ ).
<i>pState</i>	Pointer to the structure to compute the next pyramid layer.
<i>level</i>	Number of levels in the pyramid.

The [ippiPyramidInit](#) function fills the *pRoi* and *pRate* arrays and the *level* field. The value of the *level* field is equal to the minimum of the input value of the *level* parameter and the maximum possible number of layers of the pyramid with given *rate* and zero level size.

You need to specify other fields. To initialize the pyramid layer structure *pState*, use the [ippiPyramidLayerDownInit](#) or [ippiPyramidLayerUpInit](#) functions. To obtain the pyramid layer images, you can use the [ippiPyramidLayerDown](#) and [ippiPyramidLayerUp](#) functions.

For an example on how to use this function, refer to the example provided with the [ippiPyramidLayerDown](#) function description.

## Return Values

<a href="#">ippStsNoErr</a>	Indicates no error.
<a href="#">ippStsNullPtrErr</a>	Indicates an error when at least one of the pointers is NULL.
<a href="#">ippStsSizeErr</a>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<a href="#">ippStsBadArgErr</a>	Indicates an error when <i>level</i> is equal to or less than 0, or when <i>rate</i> is out of the range.

## See Also

[OpticalFlowPyrLK](#) Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

[PyramidLayerDownInit](#) Initializes the structure for creating a lower pyramid layer.

[PyramidLayerUpInit](#) Initializes the structure for creating an upper pyramid layer.

[PyramidLayerDown](#) Creates a lower pyramid layer.

[PyramidLayerUp](#) Creates an upper pyramid layer.

## GetPyramidDownROI

*Computes the size of the lower pyramid layer.*

## Syntax

```
IppStatusippiGetPyramidDownROI(IppiSize srcRoiSize, IppiSize* pDstRoiSize, Ipp32f
rate);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

## Parameters

<i>srcRoiSize</i>	Size of the source pyramid layer ROI in pixels.
<i>pDstRoiSize</i>	Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
<i>rate</i>	Ratio between source and destination layers ( $1 < \text{rate} \leq 10$ ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the lower pyramid layer *pDstRoiSize* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

$$\begin{aligned} pDstRoiSize.width &= \max(1, \min(\lceil srcRoiSize.width / rate \rceil, srcRoiSize.width - 1)) \\ pDstRoiSize.height &= \max(1, \min(\lceil srcRoiSize.height / rate \rceil, srcRoiSize.height - 1)) \end{aligned}$$

---

## NOTE

Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

---

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if <i>pDstRoiSize</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if <i>rate</i> is out of the range.

## GetPyramidUpROI

*Computes the size of the upper pyramid layer.*

---

## Syntax

```
IppStatusippiGetPyramidUpROI(IppiSize srcRoiSize, IppiSize* pDstRoiSizeMin, IppiSize*
pDstRoiSizeMax, Ipp32f rate);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>srcRoiSize</i>	Size of the source pyramid layer ROI in pixels.
<i>pDstRoiSizeMin</i>	Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
<i>pDstRoiSizeMax</i>	Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
<i>rate</i>	Ratio between source and destination layers ( $1 < \text{rate} \leq 10$ ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes possible sizes of the upper pyramid layer *pDstRoiSizeMin* and *pDstRoiSizeMax* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

maximum size *pDstRoiSizeMax*:

$$\begin{aligned} pDstRoiMax.width &= \max(srcRoiSize.width + 1, \lfloor srcRoiSize.width \cdot rate \rfloor) \\ pDstRoiMax.height &= \max(srcRoiSize.height + 1, \lfloor srcRoiSize.height \cdot rate \rfloor) \end{aligned}$$

minimum size *pDstRoiSizeMin*:

if the width and height of the source layer ROI is greater than 1,

$$\begin{aligned} pDstRoiMin.width &= \max(srcRoiSize.width + 1, \lfloor (srcRoiSize.width - 1) \cdot rate \rfloor) \\ pDstRoiMin.height &= \max(srcRoiSize.height + 1, \lfloor (srcRoiSize.height - 1) \cdot rate \rfloor) \end{aligned}$$

if the width and height of the source layer ROI is equal to 1,

$$\begin{aligned} pDstRoiMin.width &= 1 \\ pDstRoiMin.height &= 1 \end{aligned}$$

## NOTE

Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<i>ippStsBadArgErr</i>	Indicates an error condition if <i>rate</i> is out of the range.

## PyramidLayerDownGetSize

*Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.*

### Syntax

```
IppStatusippiPyramidLayerDownGetSize_8u_C1R(IppiSize srcRoi, Ipp32f rate, int kerSize,
int* pStateSize, int* pBufSize);

IppStatusippiPyramidLayerDownGetSize_8u_C3R(IppiSize srcRoi, Ipp32f rate, int kerSize,
int* pStateSize, int* pBufSize);

IppStatusippiPyramidLayerDownGetSize_16u_C1R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);

IppStatusippiPyramidLayerDownGetSize_16u_C3R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);

IppStatusippiPyramidLayerDownGetSize_32f_C1R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);

IppStatusippiPyramidLayerDownGetSize_32f_C3R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>srcRoi</i>	Size of the source image ROI.
<i>rate</i>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).
<i>kerSize</i>	Size of the kernel.
<i>pStateSize</i>	Pointer to the size of the pyramid layer state structure.
<i>pBufSize</i>	Pointer to the size of the external work buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the *pState* structure to build a lower pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the [ippiPyramidLayerDown](#) function and can be applied to process images with size not greater than *srcRoi*. For an example on how to use this function, refer to the example provided with the [ippiPyramidLayerDown](#) function description.

### Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.

---

ippStsSizeErr	Indicates an error when the width or height of images is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when <i>kerSize</i> is even, or equal to or less than 0; or when <i>rate</i> is out of the range.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerDown](#) Creates a lower pyramid layer.

## PyramidLayerDownInit

*Initializes the structure for creating a lower pyramid layer.*

---

## Syntax

### Case 1: Operating on integer data

```
IppStatusippiPyramidLayerDownInit_8u_C1R(IppiPyramidDownState_8u_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatusippiPyramidLayerDownInit_8u_C3R(IppiPyramidDownState_8u_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatusippiPyramidLayerDownInit_16u_C1R(IppiPyramidDownState_16u_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatusippiPyramidLayerDownInit_16u_C3R(IppiPyramidDownState_16u_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

### Case 2: Operating on floating point data

```
IppStatusippiPyramidLayerDownInit_32f_C1R(IppiPyramidDownState_32f_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatusippiPyramidLayerDownInit_32f_C3R(IppiPyramidDownState_32f_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

## Parameters

<i>ppState</i>	Pointer to the pointer to the initialized pyramid layer state structure.
<i>srcRoi</i>	Size of the source image ROI.
<i>rate</i>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).

<i>pKernel</i>	Separable symmetric kernel of odd length.
<i>kerSize</i>	Size of the kernel.
<i>mode</i>	Interpolation method, possible value is: IPPI_INTER_LINEAR      Bilinear interpolation.
<i>pStateBuf</i>	Pointer to the buffer to initialize the pyramid layer state structure.
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the *pState* structure to build a lower pyramid layer. This structure is used by the [ippiPyramidLayerDown](#) function and can be applied to process images with size not greater than *dstRoi*.

The specified kernel *pKernel* should be symmetric. If it is not symmetric, the function builds the symmetric kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs downsampling by discarding rows and columns that are not multiples of the rate value. For non-integer rates, the function uses bilinear interpolation (see [Linear Interpolation](#) for more information).

For an example on how to use this function, refer to the example provided with the [ippiPyramidLayerDown](#) function description.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when at least one of the pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error when the width or height of images is less than, or equal to zero.
<i>ippStsBadArgErr</i>	Indicates an error when <i>kerSize</i> is even, or equal to or less than 0; or when <i>rate</i> is out of the range.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerDown](#) Creates a lower pyramid layer.

[Linear Interpolation](#)

## PyramidLayerDown

*Creates a lower pyramid layer.*

---

## Syntax

```
IppStatusippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
--------	---------	---------

---

8u\_C3R      16u\_C3R      32f\_C3R

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI, in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination image ROI, in pixels.
<i>pState</i>	Pointer to the pyramid layer structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a lower pyramid layer *pDst* from the source image *pSrc*. The function applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, compute the size of the *pState* structure and work buffer using the [PyramidLayerDownGetSize](#) function and initialize the structure using the [PyramidLayerDownInit](#) function. The function can process images with *srcRoiSize* not greater than the *srcRoi* parameter specified in the [PyramidLayerDownInit](#) function.

---

### NOTE

This function uses the mirrored border.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

`ippStsBadArgErr` Indicates an error condition if `pState->rate` has wrong value.

## Example

To better understand usage of the `ippiPyramidLayerDown` function, refer to the `PyramidLayerDown.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerDownGetSize](#) Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

[PyramidLayerDownInit](#) Initializes the structure for creating a lower pyramid layer.

## PyramidLayerUpGetSize

*Computes the size of the structure for creating an upper pyramid layer and the size of the temporary buffer.*

---

## Syntax

```
IppStatus ippiPyramidLayerUpGetSize_8u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);

IppStatus ippiPyramidLayerUpGetSize_8u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);

IppStatus ippiPyramidLayerUpGetSize_16u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);

IppStatus ippiPyramidLayerUpGetSize_16u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);

IppStatus ippiPyramidLayerUpGetSize_32f_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);

IppStatus ippiPyramidLayerUpGetSize_32f_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>dstRoi</code>	Size of the destination image ROI.
<code>rate</code>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).
<code>kerSize</code>	Size of the kernel.
<code>pStateSize</code>	Pointer to the size of the pyramid layer state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the *pState* structure to build an upper pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the [ippiPyramidLayerUp](#) function and can be applied to process images with size not greater than *dstRoi*. For an example on how to use this function, refer to the example provided with the [ippiPyramidLayerDown](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when the width or height of images is less than, or equal to zero.
<code>ippStsBadArgErr</code>	Indicates an error when <i>kerSize</i> is even, or equal to or less than 0; or when <i>rate</i> is out of the range.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerUp](#) Creates an upper pyramid layer.

[PyramidLayerDown](#) Creates a lower pyramid layer.

## PyramidLayerUpInit

*Initializes the structure for creating an upper pyramid layer.*

## Syntax

### Case 1: Operating on integer data

```
IppStatusippiPyramidLayerUpInit_8u_C1R(IppiPyramidUpState_8u_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatusippiPyramidLayerUpInit_8u_C3R(IppiPyramidUpState_8u_C3R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatusippiPyramidLayerUpInit_16u_C1R(IppiPyramidUpState_16u_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatusippiPyramidLayerUpInit_16u_C3R(IppiPyramidUpState_16u_C3R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
```

```
IppStatusippiPyramidLayerUpInit_32f_C1R(IppiPyramidUpState_32f_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf, Ipp8u* pBuffer);
IppStatusippiPyramidLayerUpInit_32f_C3R(IppiPyramidDownState_32f_C3R** ppState,
```

## Include Files

`ippcv.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<code>ppState</code>	Pointer to the pointer to the initialized pyramid state structure.
<code>dstRoi</code>	Size of the destination image ROI.
<code>rate</code>	Ratio between neighbouring levels ( $1 < \text{rate} \leq 10$ ).
<code>pKernel</code>	Separable symmetric kernel of odd length.
<code>kerSize</code>	Size of the kernel.
<code>mode</code>	Interpolation method, possible value is: <code>IPPI_INTER_LINEAR</code> Bilinear interpolation.
<code>pStateBuf</code>	Pointer to the buffer to initialize the pyramid layer state structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory and initializes the `pState` structure to build an upper pyramid layer. This structure is used by the `ippiPyramidLayerUp` function and can be applied to process images with size not greater than `dstRoi`.

The specified kernel `pKernel` should be symmetric. If it is not symmetric, the function builds the symmetric kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs upsampling by inserting zero rows and columns that are not multiples of the rate value. For non-integer rates, the function uses bilinear interpolation (see [Linear Interpolation](#) for more information) to calculate kernel coefficients for pixels with non-integer coordinates.

For an example on how to use this function, refer to the example provided with the `ippiPyramidLayerDown` function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when the width or height of images is less than, or equal to zero.
<code>ippStsBadArgErr</code>	Indicates an error when <code>kerSize</code> is even, or equal to or less than 0; or when <code>rate</code> is out of the range.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerUp](#) Creates an upper pyramid layer.

[PyramidLayerDown](#) Creates a lower pyramid layer.

[Linear Interpolation](#)

## PyramidLayerUp

*Creates an upper pyramid layer.*

---

### Syntax

```
IppStatusippiPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPyramidUpState_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of source image ROI, in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of destination image ROI, in pixels.
<i>pState</i>	The pointer to the pyramid layer structure.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates an upper pyramid layer *pDst* from the source image *pSrc*. The function performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling the `ippiPyramidLayerUp` function, compute the size of the pyramid layer structure *pState* using the [PyramidLayerUpGetSize](#) function and initialize the structure using the [PyramidLayerUpInit](#) function. The function can process images with *srcRoiSize* not greater than the *roiSize* parameter specified in the [PyramidLayerUpInit](#) function.

---

#### NOTE

This function uses the mirrored border.

---

## Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is <code>NULL</code> .
ippStsSizeErr	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <code>srcStep</code> is less than <code>srcRoiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>dstRoiSize.width * &lt;pixelSize&gt;</code> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error condition if <code>pState-&gt;rate</code> has wrong value.

## See Also

[Regions of Interest in Intel IPP](#)

[PyramidLayerDownGetSize](#) Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

[PyramidLayerDownInit](#) Initializes the structure for creating a lower pyramid layer.

## Example of Using General Pyramid Functions

Refer to the following example to understand how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

[Pyramid.c](#)

## Image Inpainting

---

The functions described in this section allows to restore the unknown image portions. They could be used to repair damaged parts of images and to remove some objects from images. Fast direct methods of inpainting that allow for run-time correcting of video frames are supported.

### InpaintGetSize

*Computes the size of the state structure and work buffer for image inpainting.*

### Syntax

```
IppStatusippiInpaintGetSize(const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, int channels, int* pStateSize, int* pBufSize);
```

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pMask</i>	Pointer to the mask image ROI.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the mask image.
<i>roiSize</i>	Size of the image ROI, in pixels.
<i>radius</i>	Radius of the neighborhood used for inpainting.
<i>flags</i>	Specifies algorithm for image inpainting. Possible values: IPP_INPAINT_TELEA      Telea algorithm IPP_INPAINT_NS            Navier-Stokes equation
<i>channels</i>	Number of channels in the image.
<i>pStateSize</i>	Pointer to the size of the state structure.
<i>pBufSize</i>	Pointer to the size of the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the state structure for direct methods of image inpainting and the size of the external work buffer. Call this function before using [ippiInpaintInit](#). For an example on how to use this function, refer to the example provided with the [ippiInpaint](#) function description.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when width or height of the image is less than, or equal to zero.
<code>ippStsStepErr</code>	Indicates an error when the step in the mask image is too small.
<code>ippStsBadArgErr</code>	Indicates an error when <i>radius</i> is less than 1, or <i>flags</i> has an illegal value.
<code>ippStsNumChannelsErr</code>	Indicates an error when the specified number of image channels is invalid or not supported.

## See Also

[Regions of Interest in Intel IPP](#)

[Inpaint](#) MODIFIED API. Restores unknown image pixels.

[InpaintInit](#) Initializes the state structure for image inpainting.

## InpaintInit

Initializes the state structure for image inpainting.

## Syntax

```
IppStatusippiInpaintInit_8u_C1R(IppiInpaintState_8u_C1R** ppState, const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);
```

```
IppStatusippiInpaintInit_8u_C3R(IppiInpaintState_8u_C3R** ppState, const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>ppState</i>	Double pointer to the state structure for image inpaiting.
<i>pDist</i>	Pointer to the ROI of the image of distances.
<i>distStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image of distances.
<i>pMask</i>	Pointer to the mask image ROI.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the mask image.
<i>roiSize</i>	Size of the image ROI, in pixels.
<i>radius</i>	Radius of the neighborhood used for inpainting ( $dist \leq radius$ pixels are processed).
<i>flags</i>	Specifies algorithm for image inpainting. Possible values:
	IPP_INPAINT_TELEA      Telea algorithm
	IPP_INPAINT_NS      Navier-Stokes equation
<i>pStateBuf</i>	Pointer to the buffer for the state structure initialization.
<i>pBuf</i>	Pointer to the external work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the *ppState* structure for direct methods of image inpainting. This structure is used by the [ippiInpaint](#) function and can be applied to process images of the same size *roiSize*.

Zero pixels of the *pMask* image correspond to the known image pixels, non-zero pixels - to the unknown image pixels that should be restored. The distance image *pDist* specifies the order of pixel inpainting. Values of unknown pixels are restored in ascending order depending on their distances. The *radius* parameter specifies the radius of the circular neighborhood that affects the restoration of the central pixel. The *flag* parameter specifies the method of direct inpainting. Two methods are supported: Telea algorithm [[Telea04](#)] and Navier-Stokes equation [[Bert01](#)].

For an example on how to use this function, refer to the example provided with the [ippiInpaint](#) function description.

---

**NOTE** The image ROI must not exceed the maximum width and height of *roiSize* specified in the initialization function.

---

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width or height of the image is less than, or equal to zero.
ippStsStepErr	Indicates an error when the step of the mask or distance image ROI is too small.
ippStsNotEvenStepErr	Indicates an error when the step value is not divisible by the <i>pDist</i> element.
ippStsBadArgErr	Indicates an error when <i>radius</i> is less than 1, or <i>flags</i> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[Inpaint](#) MODIFIED API. Restores unknown image pixels.

[InpaintGetSize](#) Computes the size of the state structure and work buffer for image inpainting.

## Inpaint

MODIFIED API. Restores unknown image pixels.

### Syntax

```
IppStatusippiInpaint_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);

IppStatusippiInpaint_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pState</i>	The pointer to the inpainting structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

**Important** The API of this function has been modified in Intel IPP 9.0 release.

Before using this function, compute the size of the state structure and work buffer using [InpaintGetSize](#) and initialize the structure using [InpaintInit](#).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reconstructs damaged part of the image, or removes a selected object (see [Figure "Image Inpainting"](#)). The image part to restore is defined by the mask that is created when the inpainting structure *pState* is initialized by the [InpaintInit](#) function. Different distant transforms can be used, but the Fast Marching method ([ippiFastMarching](#)) provides the best results. The order of pixel restoration is defined by the distance through the initialization the inpainting structure *pState* by the [InpaintInit](#) function. Pixels are restored in according to the growing of their distance value. When a pixel is inpainted, it is treated as the known one.

Two algorithms of direct inpainting are supported (controlled by the parameter *flags* of the [InpaintInit](#) function):

- image restoration of the unknown pixel by the weighted sum of approximations by known pixels in the neighborhood (*flags* = [IPP\\_INPAINT\\_TELEA](#)) [[Telea04](#)],
- image restoration based on the Navier-Stokes equations (*flags* = [IPP\\_INPAINT\\_NS](#)) [[Bert01](#)].

The inpainting structure *pState* can be used to perform restoration of several different images of the same size *roiSize*.

### Image Inpainting



### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with zero or negative value, or if differs from the corresponding parameter that is specified when the inpainting structure is initialized by <a href="#">InpaintInit</a> .

`ippStsStepErr` Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.

## Example

To better understand usage of the `ippiInpaint` function, refer to the `Inpaint.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

`InpaintGetSize` Computes the size of the state structure and work buffer for image inpainting.  
`InpaintInit` Initializes the state structure for image inpainting.

# Image Segmentation

This section describes the functions that perform image segmentation using different techniques. These functions allow to extract parts of the image that can be associated with objects of the real world. Watershed and gradient segmentation are region-based methods to split image into the distinctive areas.

Background/foreground segmentation allows for distinguishing between moving objects and stable areas of the background.

## LabelMarkersGetBufferSize

*Computes the size of the working buffer for the marker labeling.*

### Syntax

```
IppStatus ippiLabelMarkersGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_8u32s_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_16u_C1R(IppiSize roiSize, int* pBufSize);
```

### Include Files

`ippcv.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
<code>pBufSize</code>	Pointer to the computed size of the working buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the `ippiLabelMarkers` function. The buffer with the length `pBufSize[0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter `roiSize`.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer <i>pBufSize</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## LabelMarkers

*Labels* markers in image with different values.

---

### Syntax

```
IppStatusippiLabelMarkers_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize,
int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);

IppStatusippiLabelMarkers_8u32s_C1R(Ipp8u* pSrcMarker, int srcMarkerStep, Ipp32s*
pDstMarker, int dstMarkerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm
norm, int* pNumber, Ipp8u* pBuffer);

IppStatusippiLabelMarkers_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize,
int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.i.lib

### Parameters

<i>pMarker</i>	Pointer to the source and destination image ROI (for in-place flavors).
<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>pSrcMarker</i>	Pointer to the source image ROI (for not-in-place flavors).
<i>srcMarkerStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image (for not-in-place flavors).
<i>pDstMarker</i>	Pointer to the source and destination image ROI (for not-in-place flavors).
<i>dstMarkerStep</i>	Distance, in bytes, between the starting points of consecutive lines in the destination image (for not-in-place flavors).
<i>minLabel</i>	Minimal value of the marker label ( $0 < \text{minLabel} \leq \text{maxLabel}$ ).
<i>maxLabel</i>	Maximal value of the marker label ( $\text{minLabel} \leq \text{maxLabel} < 255$ for 8-bit markers, and $\text{minLabel} \leq \text{maxLabel} < 65535$ for 16-bit markers, and $\text{minLabel} \leq \text{maxLabel} < (2^{31}-1)$ for 32-bit markers).
<i>roiSize</i>	Size of the source and destination image ROI in pixels.

*norm*                      Specifies type of the norm to form the mask for marker propagation:

`ippiNormInf`      Infinity norm (8-connectivity);  
`ippiNormL1`      L1 norm (4-connectivity).

*pNumber* Pointer to the number of markers.

*pBuffer* Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function labels markers in the destination image with different integer values. Each connected set of non-zero image pixels is treated as the separate marker. 4- or 8-connectivity can be used depending on the norm type. All pixels belonging to the same marker are set to the same value from the interval  $[minLabel, maxLabel]$ . Two markers can be labeled with the same value if the number of connected components exceeds  $minLabel - maxLabel + 1$ . The image with labeled markers can be used as the seed image for segmentation by functions `ippiSegmentWatershed` or `ippiSegmentGradient` functions.

The function requires the working buffer `pBuffer` whose size should be computed by the function `ippiLabelMarkersGetBufferSize` beforehand.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <i>markerStep</i> is less than <i>roiSize.width * pixelSize</i> .
ippStsNotEvenStepErr	Indicates an error condition if <i>markerStep</i> , <i>srcMarkerStep</i> , or <i>dstMarkerStep</i> is not divisible by respective <i>&lt;pixelSize&gt;</i> .
ippStsBadArgErr	Indicates an error condition if one of the <i>minLabel</i> , <i>maxLabel</i> , and <i>norm</i> has an illegal value.

## Example

To better understand usage of the `ippiLabelMarkers` function, refer to the `LabelMarkers.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## MarkSpecklesGetBufferSize

*Computes the size of the external work buffer for speckle marking.*

## Syntax

```
IppStatusippiMarkSpecklesGetBufferSize(IppiSize roiSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>dataType</i>	Data type of the source and destination image.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pBufferSize</i>	Pointer to the computed size (in bytes) of the external work buffer.

## Description

This function computes the size of the external work buffer for the [MarkSpeckles](#) function.

For an example on how to use this function, refer to the example provided with the [MarkSpeckles](#) function description.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when <i>pBufferSize</i> is NULL.
ippStsSizeErr	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
ippStsDataTypeErr	Indicates an error when <i>dataType</i> has an illegal value.
ippStsNumChannelErr	Indicates an error when <i>numChannels</i> has an illegal value.

## See Also

[MarkSpeckles](#) Marks small noise blobs (speckles) in an image.

## MarkSpeckles

Marks small noise blobs (speckles) in an image.

## Syntax

```
IppStatusippiMarkSpeckles_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize  
roiSize, Ipp<datatype> speckleVal, int maxSpeckleSize, Ipp<datatype> maxPixDiff,  
IppiNorm norm, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1IR        16u\_C1IR        16s\_C1IR        32f\_C1IR

## Include Files

ippcv.h

## Domain Dependencies

**Headers:**ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:**ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination image.
<i>srcDstStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>speckleVal</i>	Value to set to the speckles.
<i>maxSpeckleSize</i>	Maximum size of the image component to consider it as a speckle.
<i>maxPixDiff</i>	Maximum difference between neighboring disparity pixels to put them into the same component.
<i>norm</i>	Type of the norm to form the mask for marker propagation. Possible value is: ippiNormL1                            L1 norm (4-connectivity)
<i>pBuffer</i>	Pointer to the work buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function marks small noise blobs (speckles) in the source image.

The *pSrcDst* parameter points to the processed source and destination image ROI.

The function finds small connected components and set them to the *speckleVal* value. This function marks only components with size that is less than, or equal to *maxSpeckleSize*. Pixels of the image belong to the same connected component if the difference between adjacent pixels (considering 4-connected adjacency) is less than, or equal to the *maxSpeckleSize* value.

---

### NOTE

This release does not support 8-connectivity.

---

The function does not process the pixels of the image that already have the *speckleVal* value.

Before using the `ippiMarkSpeckles` function, compute the size of the external buffer using the [MarkSpecklesGetBufferSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrcDst</i> or <i>pBufferSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.

ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsNormErr	Indicates an error when <i>norm</i> has an incorrect or not supported value.

## Example

To better understand usage of the `ippiMarkSpeckles` function, refer to the `MarkSpeckles.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

[User-defined Border Types](#)

[FilterGaussianGetBufferSize](#) Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

[FilterGaussianInit](#) Initializes the Gaussian context structure.

## SegmentWatershedGetBufferSize

*Computes the size of the working buffer for the watershed segmentation.*

---

## Syntax

```
IppStatus ippiSegmentWatershedGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiSegmentWatershedGetBufferSize_8u16u_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiSegmentWatershedGetBufferSize_32f16u_C1R(IppiSize roiSize, int*
pBufSize);
```

## Include Files

`ippcv.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

*roiSize* Size of the source image ROI in pixels.

*pBufSize* Pointer to the computed size of the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the [ippiSegmentWatershed](#) function. The buffer with the length *pBufSize[0]* can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
-------------	--

---

ippStsNullPtrErr	Indicates an error condition if the pointer <i>pBufSize</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## SegmentWatershed

Performs marker-controlled watershed segmentation of an image.

---

### Syntax

```
IppStatusippiSegmentWatershed_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);

IppStatusippiSegmentWatershed_8u16u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp16u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);

IppStatusippiSegmentWatershed_32f16u_C1IR(const Ipp32f* pSrc, int srcStep, Ipp16u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.								
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.								
<i>pMarker</i>	Pointer to the ROI of the source and destination image of markers.								
<i>markerStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image with markers.								
<i>roiSize</i>	Size of the source and destination image ROI, in pixels.								
<i>norm</i>	Specifies the type of the norm to form the mask for marker propagation: <table> <tr> <td>ippiNormInf</td><td>Infinity norm (8-connectivity, 3x3 rectangular mask)</td></tr> <tr> <td>ippiNormL1</td><td>L1 norm (4-connectivity, 3x3 cross mask)</td></tr> <tr> <td>ippiNormL2</td><td>Approximation of L2 norm (8-connectivity, 3x3 mask) [Bor86]</td></tr> <tr> <td>ippiNormFM</td><td>Fast marching distance [Telea04]</td></tr> </table>	ippiNormInf	Infinity norm (8-connectivity, 3x3 rectangular mask)	ippiNormL1	L1 norm (4-connectivity, 3x3 cross mask)	ippiNormL2	Approximation of L2 norm (8-connectivity, 3x3 mask) [Bor86]	ippiNormFM	Fast marching distance [Telea04]
ippiNormInf	Infinity norm (8-connectivity, 3x3 rectangular mask)								
ippiNormL1	L1 norm (4-connectivity, 3x3 cross mask)								
ippiNormL2	Approximation of L2 norm (8-connectivity, 3x3 mask) [Bor86]								
ippiNormFM	Fast marching distance [Telea04]								
<i>flag</i>	Specifies the algorithm of segmentation. The value is a logical sum of the following mandatory values: <table> <tr> <td>IPP_SEGMENT_QUEUE</td><td>Priority queue is used to define the order of pixel processing.</td></tr> </table>	IPP_SEGMENT_QUEUE	Priority queue is used to define the order of pixel processing.						
IPP_SEGMENT_QUEUE	Priority queue is used to define the order of pixel processing.								

`IPP_SEGMENT_DISTANCE` Distance transform algorithm is used for segmentation.

and the following optional values:

`IPP_SEGMENT_BORDER_4` Pixels of the 4-connectivity border between image segments are marked with the `IPP_MAX_8U` (255) value.

`IPP_SEGMENT_BORDER_8` Pixels of the 8-connectivity border between image segments are marked with the `IPP_MAX_8U` (255) value.

*pBuffer* Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs marker-controlled watershed segmentation of the source image. Non-zero pixels of the *pMarker* image belong to water source markers. Marker values propagate through the whole image according to the watershed algorithm. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Watershed segmentation is preferable for images with local minimums, for example, gradient images. Image markers generally correspond to these local minimums and can be created, for example, manually or using morphological reconstruction.

The parameter *flag* specifies how watershed segmentation is performed. This parameter is a logical sum of two values among the following supported values:

- Mandatory values specifying the algorithm of segmentation:

<code>IPP_SEGMENT_QUEUE</code>	Classic watershed segmentation scheme with the priority queue [ <a href="#">Vincent91</a> ]
--------------------------------	--

<code>IPP_SEGMENT_DISTANCE</code>	Watershed segmentation by calculating the topographic distance for each pixel [ <a href="#">Lotufo00</a> ], [ <a href="#">Meyer94</a> ]
-----------------------------------	---

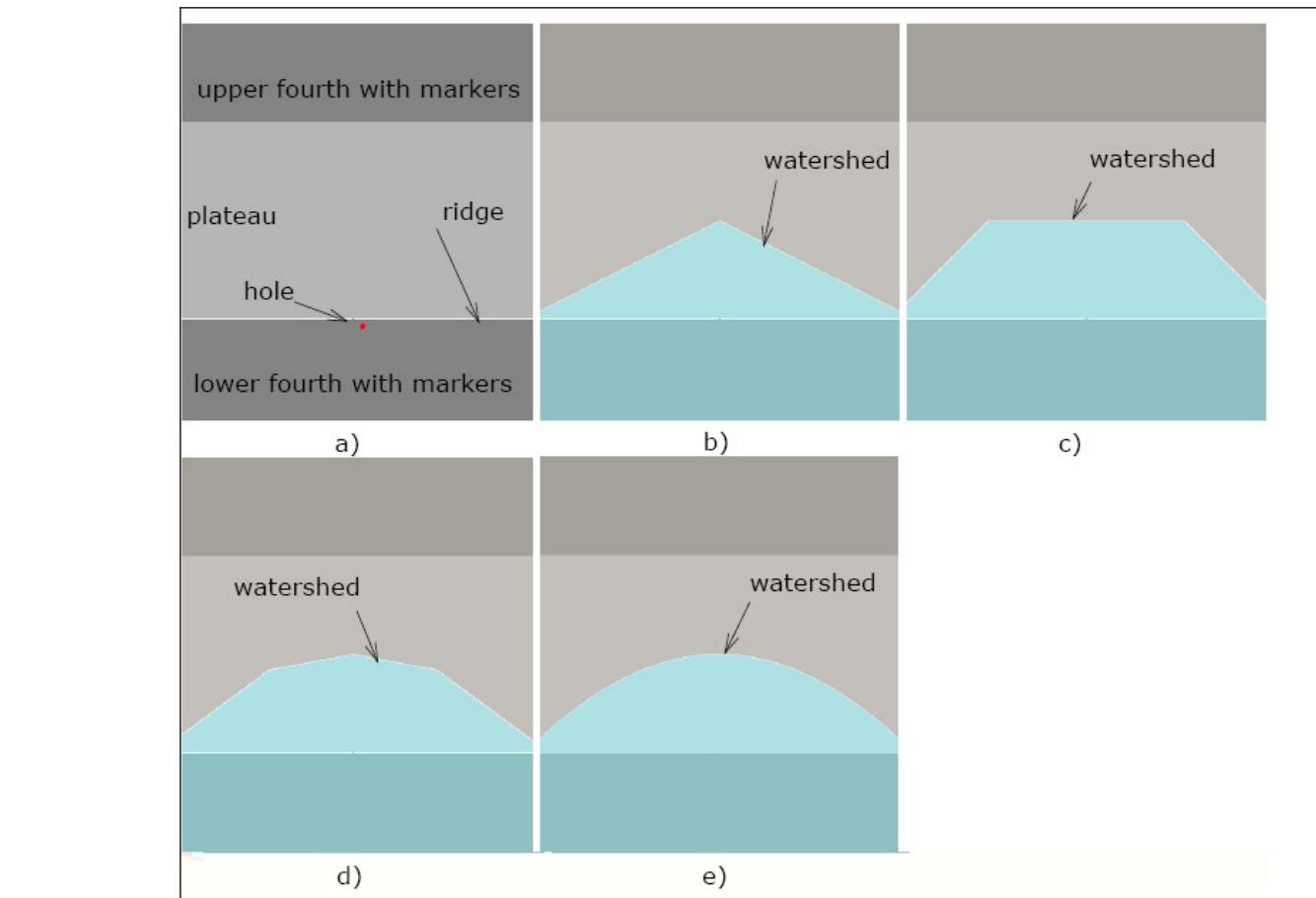
- Optional additional values of the *flag*: `IPP_SEGMENT_BORDER_4` and `IPP_SEGMENT_BORDER_8` specify the border of the segments. All pixels adjacent to the differently marked pixels are considered as border pixels, and their values are set to `IPP_MAX_8U` (255) for 8-bit markers, or `IPP_MAX_16U` (65535) for 16-bit markers. In this case, the value `IPP_MAX_8U` (`IPP_MAX_16U`) should not be used to mark segments. If these optional values are not specified, segments are formed without borders.

The function requires the working buffer *pBuffer*, which size should be computed by the function `ippiSegmentWatershedGetBufferSize` beforehand.

[Figure “Watershed Segmentation with Different Norms”](#) shows the plateau filling through the watershed segmentation with different values of the *norm* parameters. Initial image (a) has the labeled with markers upper and lower fourths with low pixel value, the central plateau between them, the ridge between the

plateau and the lower forth with one pixel hole in the center of it. The following pictures are segmentation results: b) - for L1 norm (block distance), c)- Linf norm (chessboard distance), d) - approximate L2 (Euclidian) norm [Bor86], e) Fast Marching distance.

### Watershed Segmentation with Different Norms



### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the <i>srcStep</i> or <i>markerStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippNotEvenStsStepErr	Indicates an error condition if one of the <i>srcStep</i> or <i>markerStep</i> for 16-bit integer images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if <i>norm</i> has an illegal value.

## SegmentGradientGetBufferSize

*Computes the size of the working buffer for the gradient segmentation.*

---

### Syntax

```
IppStatusippiSegmentGradientGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);  
IppStatusippiSegmentGradientGetBufferSize_8u_C3R(IppiSize roiSize, int* pBufSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBufSize</i> , <i>pBufferSize</i>	Pointer to the computed size of the working buffer.

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for the [ippiSegmentGradient](#) function. The buffer with the length *pBufSize*[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer <i>pBufSize</i> is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## SegmentGradient

*Performs image segmentation by region growing to the least gradient direction.*

---

### Syntax

```
IppStatusippiSegmentGradient_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,  
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);  
IppStatusippiSegmentGradient_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,  
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

### Include Files

ippcv.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMarker</i>	Pointer to the ROI of the source and destination image of markers.
<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the image of markers.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>norm</i>	Specifies type of the norm to form the mask for marker propagation:  ippiNormInf                  Infinity norm (8-connectivity, 3x3 rectangular mask); ippiNormL1                  L1 norm (4-connectivity, 3x3 cross mask);
<i>flags</i>	optional flag:  IPP_SEGMENT_BORDER_4        pixels of the 4-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254). IPP_SEGMENT_BORDER_8        pixels of the 8-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs image segmentation by region growing with markers. Non-zero pixels of *pMarker* image belong to initial image regions. Marker values propagate through the whole image in the direction of the least value of the absolute value of the image gradient. For 3-channel image the gradient is calculated as the maximum of channel gradients. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Gradient segmentation is generally done for an image without explicit calculation of the image gradient. [Meyer92]

If `IPP_SEGMENT_BORDER` flag is defined, then the pixels adjacent to differently marked pixels are assumed to be border pixels and are set to a special value (254). This value must not be used to mark segments in this case.

Another special value (255) is used inside the function and can not be used to mark segment in any case.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiSegmentGradientGetBufferSize` beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the <i>srcStep</i> or <i>markerStep</i> is less than <i>roiSize.width * &lt;pixelsize&gt;</i> .
ippStsBadArgErr	Indicates an error condition if <i>norm</i> has an illegal value.

## BoundSegments

*Marks pixels belonging to segment boundaries.*

---

### Syntax

```
IppStatusippiBoundSegments_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize,
Ipp8u val, IppiNorm norm);

IppStatusippiBoundSegments_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize,
Ipp16u val, IppiNorm norm);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<i>pMarker</i>	Pointer to the ROI of the source and destination image of markers.
<i>markerStep</i>	Distance in bytes between starts of consecutive lines in the image of markers.
<i>roiSize</i>	Size of the source and destination image ROI in pixels.
<i>val</i>	Value of the boundary pixel.
<i>norm</i>	Specifies type of the norm for pixel neighborhood: ippiNormInf                    Infinity norm (8-connectivity); ippiNormL1                    L1 norm (4-connectivity).

### Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function detects segment boundaries in the *pMarker* image and sets border pixels to the value *val*. A segment is the set of connected pixels of the *pMarker* image with the same value not equal to *val*. After boundaries are marked, the *pMarker* image does not contain any pair of adjacent in *norm* pixels with the same value that is not equal to *val*.

### Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
-------------	--

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if <code>markerStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
ippNotEvenStsStepErr	Indicates an error condition if <code>markerStep</code> for 16-bit integer images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if <code>norm</code> has an illegal value.

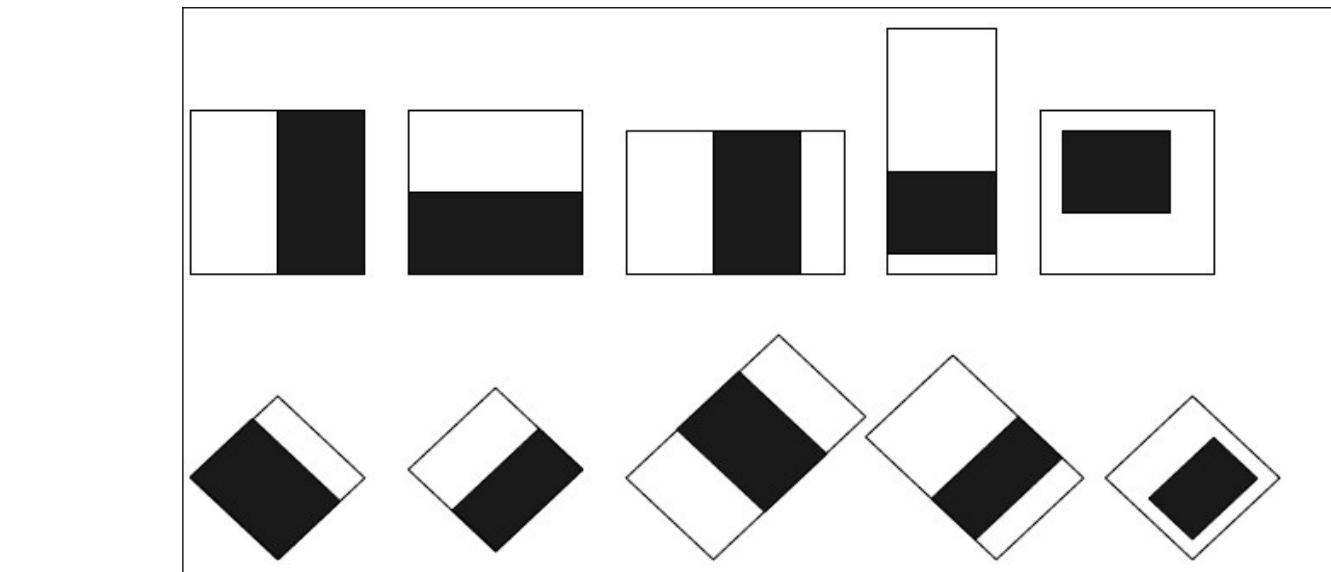
## Pattern Recognition

---

### Object Detection Using Haar-like Features

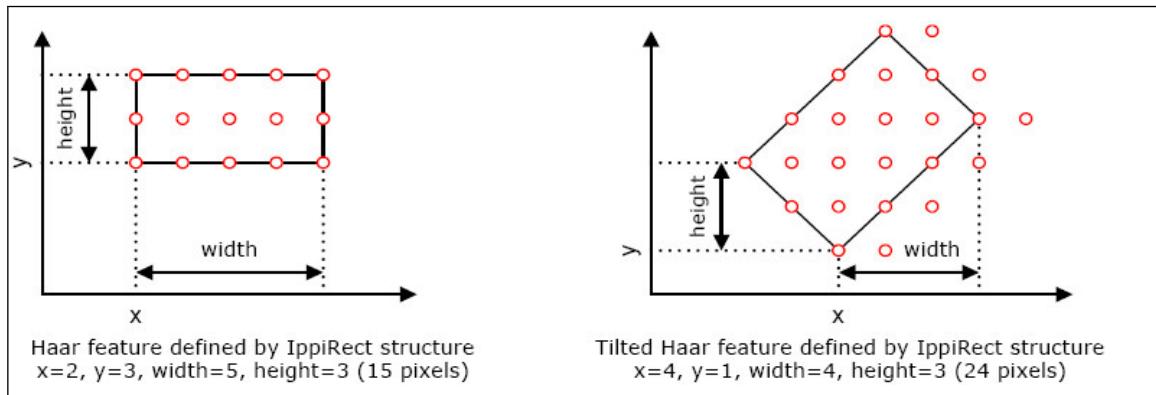
The object detector described in [ Viola01] and [ Lein02] is based on Haar classifiers. Each classifier uses  $k$  rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. [Figure "Types of Haar Features"](#) shows different types of Haar features.

#### Types of Haar Features



In the Intel IPP Haar features are represented using IppRect structure. Figure “Representing Haar Features” shows how it can be done for common and tilted features.

### Representing Haar Features



When the classifier  $K_t$  is applied to the pixel  $(i, j)$  of the image  $A$ , it yields the value  $\text{val1}(t)$  if

$$\sum_{i=1}^k w_i \cdot \sum_{u=i+R_1Y} \sum_{v=j+R_1YX} A_{uv} < \text{norm}(i, j) \cdot \text{threshold}(t)$$

and  $\text{val2}(t)$  otherwise.

Here  $w_i$  is a feature weight,  $\text{norm}(i, j)$  is the norm factor (generally the standard deviation on the rectangle containing all features),  $\text{threshold}(t)$ ,  $\text{val1}(t)$  and  $\text{val2}(t)$  are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences called *stages* (*classification stages*). The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

### HaarClassifierGetSize

*Computes the size of the structure for standard Haar classifiers.*

### Syntax

```
IppStatusippiHaarClassifierGetSize(IppDataType dataType, IppiSize roiSize, const int* pNum, int length, int* pSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

dataType	Data type of the source image. Possible values: ipp32f, ipp32s.
roiSize	Maximal size of the source image ROI, in pixels.

---

<i>pNum</i>	Pointer to the array of Haar classifier lengths.
<i>length</i>	Number of classifiers in the stage.
<i>pSize</i>	Pointer to the size of Haar classifier structure.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the *pState* structure that is required to calculate the sequence of Haar classifiers - classification stage. The *i*-th classifier in the stage has *pNum[i]* rectangular features.

The length of the *pThreshold*, *pVal1*, and *pVal2* vectors used in the [ippiHaarClassifierInit](#) function is equal to *length*.

The length of the *pFeature* and *pWeight* vectors is equal to:

$$\sum_{i=0}^{\text{length}-1} pNum[i]$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when: <ul style="list-style-type: none"> <li><i>length</i> or one of the <i>pNum[i]</i> values is less than, or equal to zero</li> <li><i>roiSize</i> has a field with a zero or negative value</li> </ul>
<code>ippStsBadArgErr</code>	Indicates an error when one of the features is defined incorrectly.
<code>ippStsDataTypeErr</code>	Indicates an error when <i>dataType</i> has an illegal value.

## See Also

[Regions of Interest in Intel IPP](#)

[HaarClassifierInit](#) Initializes the structure for standard Haar classifiers.

[ApplyHaarClassifier](#) Applies a Haar classifier to an image.

## HaarClassifierInit

Initializes the structure for standard Haar classifiers.

## Syntax

```
IppStatusippiHaarClassifierInit_32f(IppiHaarClassifier_32f** ppState, const IppiRect*  
pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const  
Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatusippiHaarClassifierInit_32s(IppiHaarClassifier_32s** ppState, const IppiRect*  
pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const  
Ipp32s* pVal2, const int* pNum, int length);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>ppState</i>	Double pointer to the Haar classifier structure.
<i>pFeature</i>	Pointer to the array of features.
<i>pWeight</i>	Pointer to the array of feature weights.
<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1</i> , <i>pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of Haar classifier lengths.
<i>length</i>	Number of classifiers in the stage.

## Description

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The *i*-th classifier in the stage has *pNum[i]* rectangular features. Each feature is defined by a certain rectangle with horizontal and vertical sides. The length of the *pThreshold*, *pVal1*, and *pVal2* vectors is equal to *length*. The length of *pFeature* and *pWeight* is equal to:

$$\text{length} - 1$$

$$\sum_{i=0}^{\text{length}-1} \text{pNum}[i]$$

Result of applying classifiers to the image is computed using [the formula in "Object Detection Using Haar-like Features"](#).

All features of the classifier initialized by the `ippiHaarClassifierInit` function have vertical and horizontal sides (left part of [Figure "Representing Haar Features"](#)). Some of these features then can be tilted using the `ippiTiltHaarFeatures` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .

---

ippStsSizeErr	Indicates an error when <i>length</i> or one of the <i>pNum[i]</i> values is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when one of the features is defined incorrectly.

## See Also

[HaarClassifierGetSize](#) Computes the size of the structure for standard Haar classifiers.

[ApplyHaarClassifier](#) Applies a Haar classifier to an image.

[Object Detection Using Haar-like Features](#)

[TiltHaarFeatures](#) Modifies a Haar classifier by tilting specified features.

## GetHaarClassifierSize

*Returns the size of the Haar classifier.*

---

### Syntax

```
IppStatusippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState, IppiSize* pSize);
```

```
IppStatusippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState, IppiSize* pSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ipp.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

### Parameters

*pState* Pointer to the Haar classifier structure.

*pSize* Pointer to the size of Haar classifier structure.

### Description

This function computes the minimum size of the window containing all features of the Haar classifier described by the *pState*.

### Return Values

ippStsNoErr	Indicates no error.
-------------	---------------------

ippStsNullPtrErr	Indicates an error condition if the <i>pState</i> pointer is NULL.
------------------	--

## TiltedHaarClassifierInit

*Initializes the structure for tilted Haar classifiers.*

---

### Syntax

```
IppStatusippiTiltedHaarClassifierInit_32f(IppiHaarClassifier_32f* pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatusippiTiltedHaarClassifierInit_32s(IppiHaarClassifier_32s* pState, const
IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s*
pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pState</i>	Double pointer to the Haar classifier structure.
<i>pFeature</i>	Pointer to the array of features.
<i>pWeight</i>	Pointer to the array of feature weights.
<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1</i> , <i>pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of Haar classifier lengths.
<i>length</i>	Number of classifiers in the stage.

## Description

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The *i*-th classifier in the stage has *pNum[i]* rectangular features. Each feature is defined by a certain rectangle with sides tilted by 45 degrees. You should specify the points with minimum and maximum row numbers. The length of the *pFeature*, *pFeature*, *pWeight*, *pVal1*, and *pVal2* vectors is equal to:

$$\text{length} - 1$$

$$\sum_{i=0}^{\text{length}-1} \text{pNum}[i]$$

Result of applying classifiers to the image is computed using [the formula in "Object Detection Using Haar-like Features"](#).

All features of the classifier initialized by the `ippiTiltedHaarClassifierInit` function have tilted sides (right part of [Figure "Representing Haar Features"](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when:

- *length* or one of the *pNum[i]* values is less than, or equal to zero
- Sum of all elements of *pNum* is not equal to *length*

**ippStsBadArgErr** Indicates an error when one of the features is defined incorrectly.

## See Also

[ApplyHaarClassifier](#) Applies a Haar classifier to an image.

[Object Detection Using Haar-like Features](#)

[TiltHaarFeatures](#) Modifies a Haar classifier by tilting specified features.

## TiltHaarFeatures

Modifies a Haar classifier by tilting specified features.

## Syntax

```
IppStatusippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag,  
IppiHaarClassifier_32f* pState);
```

```
IppStatusippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag,  
IppiHaarClassifier_32s* pState);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pMask</i>	Pointer to the mask vector.
<i>flag</i>	Flag to choose the direction of feature tilting.
<i>pState</i>	Pointer to the Haar classifier structure.

## Description

This function tilts specified features of the Haar classifier. Before using this function, compute the size of the Haar classifier state structure using [HaarClassifierGetSize](#) and initialize the structure using [TiltedHaarClassifierInit](#). Non-zero elements of previously prepared vector *pMask* indicates the features that are tilted. The *flag* parameter specifies how the features are tilted:

- if *flag* is equal to 0, the feature is tilted around the left top corner clockwise
- if *flag* is equal to 1, the feature is tilted around the bottom left corner counter-clockwise

This mixed classifier containing both common and tilted features can be used by the function [ippiApplyMixedHaarClassifier](#).

## Return Values

<b>ippStsNoErr</b>	Indicates no error.
<b>ippStsNullPtrErr</b>	Indicates an error when one of the specified pointers is NULL.
<b>ippStsBadArgErr</b>	Indicates an error when the classifier is tilted already.

## See Also

[HaarClassifierGetSize](#) Computes the size of the structure for standard Haar classifiers.  
[TiltedHaarClassifierInit](#) Initializes the structure for tilted Haar classifiers.

## ApplyHaarClassifier

*Applies a Haar classifier to an image.*

---

## Syntax

```
IppStatusippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const
Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatusippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const
Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatusippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const
Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the source image of integrals.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image.
<i>pNorm</i>	Pointer to the ROI in the source image of norm factors.
<i>normStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image of the norm factors.
<i>pMask</i>	Pointer to the source and destination image of classification decisions.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image of classification decisions.
<i>pPositive</i>	Pointer to the number of positive decisions.
<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the Haar classifier structure.
<i>scaleFactor</i>	Scale factor (see <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the Haar classifier to pixels of the source image ROI *pSrc*. The source image should be in the integral representation, it can be obtained by calling one of the [integral functions](#) beforehand. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^k (pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l]) \cdot w_l$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of *l*-th rectangle of the feature, and  $w_l$  is the feature weight. For  $i = 0..roiSize.height - 1, j = 0..roiSize.width - 1$  all pixels referred in the above formula should be allocated in memory.

The input value of *pPositive[0]* is used as a hint to choose the calculation algorithm. If it is greater than or equal to *roiSize.width\*roiSize.height*, the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the *pPositive[0]*.

Before using this function, you need to compute the size of the state structure using [HaarClassifierGetSize](#) and initialize the structure using [HaarClassifierInit](#) or [TiltedHaarClassifierInit](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when one of the image step values is less than <i>roiSize.width*pixelSize</i> .
<code>ippStsNorEvenStepErr</code>	Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.

## See Also

[HaarClassifierGetSize](#) Computes the size of the structure for standard Haar classifiers.

[HaarClassifierInit](#) Initializes the structure for standard Haar classifiers.

[TiltedHaarClassifierInit](#) Initializes the structure for tilted Haar classifiers.

## ApplyMixedHaarClassifier

Applies a mixed Haar classifier to an image.

### Syntax

```
IppStatusippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const
Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int
maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f*
pState);

IppStatusippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int
maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f*
pState);
```

```
IppStatusippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int
maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s*
pState, int scaleFactor);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the source image of integrals.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image of integrals.
<i>pTilt</i>	Pointer to the ROI in the source image of tilted integrals.
<i>tiltStep</i>	Distance, in bytes, between the starting points of consecutive lines in the source image of tilted integrals.
<i>pNorm</i>	Pointer to the ROI in the source image of norm factors.
<i>normStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image of the norm factors.
<i>pMask</i>	Pointer to the source and destination image of classification decisions.
<i>maskStep</i>	Distance, in bytes, between the starting points of consecutive lines in the image of classification decisions.
<i>pPositive</i>	Pointer to the number of positive decisions.
<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the mixed Haar classifier structure.
<i>scaleFactor</i>	Scale factor (see Integer <a href="#">Integer Result Scaling</a> ).

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the mixed Haar classifier *pState* to the ROI of the source images *pSrc* and *pTilt*. The mixed Haar classifier is a classifier initialized by [HaarClassifierInit](#) and then modified by the [TiltHaarFeatures](#) function. The source images must be in the integral representation, they can be obtained by calling one of the [integral functions](#) beforehand. Common features are applied to the *pSrc* image, and tilted features are applied to the *pTilt* image. The sum of pixels on feature rectangles is computed as:

$$\begin{aligned}
 & \sum_{l=1}^k (pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l]) \cdot w_l \\
 & \text{or} \\
 & \sum_{l=1}^k (pTilt[i+y_l, j+x_l] - pTilt[i+Y_l, j+x_l] - pTilt[i+y_l, j+X_l] + pTilt[i+Y_l, j+X_l]) \cdot w_l
 \end{aligned}$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of  $l$ -th rectangle of the feature , and  $w_l$  is the feature weight. For  $i = 0..roiSize.height - 1, j = 0..roiSize.width - 1$  all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height` the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of `pMask` image. If the sum is less than `threshold` than the negative decision is made and the value of the corresponding pixel of the `pMask` image is set to zero. The number of positive decisions is assigned to the `pPositive[0]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize</code> has a field with a zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when one of the image step values is less than <code>roiSize.width*pixelSize</code> .
<code>ippStsNorEvenStepErr</code>	Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.

## See Also

[Regions of Interest in Intel IPP](#)

[HaarClassifierInit](#) Initializes the structure for standard Haar classifiers.

[TiltHaarFeatures](#) Modifies a Haar classifier by tilting specified features.

## Local Binary Pattern (LBP) Operator

The local binary pattern (LBP) operator transforms an image into an array, or to an image with integer labels. Integer labels describe small-scale view of the image. For grayscale images, these labels represent a texture descriptor of the image. Integer labels statistics are used for image analysis. Changes of the monotonic gray level do not affect the LBP operator.

Intel® IPP functions described in this section use LBP operators with mask size 3x3 and 5x5.

The `LBPIImageMode` functions support four modes of LBP calculation set by the `mode` parameter. The `LBPIImage` functions compute LBP similar to the `LBPIImageMode` functions with the `mode` value equal to 1.

The LBP operator with 3x3 mask uses neighborhood consisting of eight pixels, as shown in the figures below.

`mode=0`:

1	8	7
2	A	6
3	4	5

Anchor Point

*mode=1:*

1	2	3
8	A	4
7	6	5

*mode=2:*

8	7	6
1	A	5
2	3	4

*mode=3:*

2	3	4
1	A	5
8	7	6

The LBP operator with 5x5 mask uses neighborhood consisting of 16 pixels, as shown in the figures below.

*mode=0:*

0	15	14	13	0
1	16	0	12	11
2	0	A	0	10
3	4	0	8	9
0	5	6	7	0

Anchor Point

mode=1:

0	3	4	5	0
1	2	0	6	7
16	0	A	0	8
15	14	0	10	9
0	13	12	11	0

mode=2:

0	14	13	12	0
16	15	0	11	10
1	0	A	0	9
2	3	0	7	8
0	4	5	6	0

mode=3:

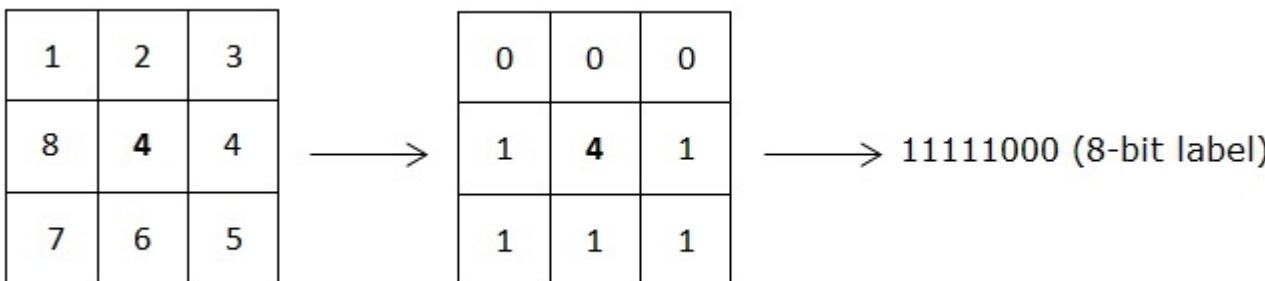
0	4	5	6	0
2	3	0	7	8
1	0	A	0	9
16	15	0	11	10
0	14	13	12	0

In the above figures:

- Numbers indicate the position of the corresponding bit in a resulting label
- The **A** letter indicates the anchor point position.

The LBP operator does the following when processing an image:

- Compares each pixel neighboring to the anchor with the anchor pixel in accordance with the neighboring pixel order. If the neighboring pixel value is more than, or equal to the anchor point value, the result is 1. If the neighboring pixel value is less than the anchor point value, the result is 0.
- Puts the result of comparison to the corresponding bit of the resulting label, as shown in the figure below.



## LBPIImageMode

Calculates LBP of the image according to the specified mode.

### Syntax

```
IppStatusippiLBPIImageMode3x3_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```

Supported values for `mod`:

8u\_C1R                    32f8u\_C1R

```
IppStatusippiLBPIImageMode5x5_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```

---

8u_C1R	8u16u_C1R	32f8u_C1R	32f16u_C1R
--------	-----------	-----------	------------

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.	
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.	
<code>pDst</code>	Pointer to the destination image ROI.	
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.	
<code>dstRoiSize</code>	Size of the destination ROI, in pixels.	
<code>mode</code>	Mode for LBP calculation. Supported values are 0, 1, 2, 3.	
<code>borderType</code>	Type of border. Possible values are:	
	<code>ippBorderRepl</code>	Border is replicated from the edge pixels.
	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
	Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderRepl</code> and <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> .	
<code>borderValue</code>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.	

## Description

These functions operate with ROI (see [Regions of Interest in Intel IPP](#)).

The `ippiLBPIImageMode3x3` and `ippiLBPIImageMode5x5` functions calculate LBP of the `pSrc` image ROI according to the `mode` value. The result is stored in the `pDst` destination image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <code>dstRoiSize</code> has a field with a zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error when <code>border</code> has an illegal value.

## Example

To better understand usage of the `ippiLBPIImageMode` function, refer to the `LBPIImageMode.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)

### LBPIImageHorizCorr

*Calculates a correlation between two LBPs.*

---

## Syntax

```
IppStatusippiLBPIImageHorizCorr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize
dstRoisize, int horShift, IppiBorderType borderType, const Ipp<datatype>* borderValue);
```

Supported values for `mod`:

8u_C1R	16u_C1R
--------	---------

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc1</code> , <code>pSrc2</code>	Pointers to the source image ROI.	
<code>src1Step</code> , <code>src2Step</code>	Distance, in bytes, between the starting points of consecutive lines in the source image.	
<code>pDst</code>	Pointer to the destination image ROI.	
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in the destination image.	
<code>dstRoisize</code>	Size of the destination ROI in pixels.	
<code>horShift</code>	Horizontal shift of the <code>pSrc2</code> image.	
<code>borderType</code>	Type of border. Possible values are:	
	<code>ippBorderRepl</code>	Border is replicated from the edge pixels.
	<code>ippBorderInMem</code>	Border is obtained from the source image pixels in memory.
	Mixed borders are also supported. They can be obtained by the bitwise operation OR between <code>ippBorderRepl</code> and <code>ippBorderInMemTop</code> , <code>ippBorderInMemBottom</code> , <code>ippBorderInMemLeft</code> , <code>ippBorderInMemRight</code> .	

---

<i>borderValue</i>	Constant value to assign to pixels of the constant border. This parameter is applicable only to the <code>ippBorderConst</code> border type.
--------------------	--

## Description

This function operates with ROI.

This function calculates the difference between two LBP images. The result is stored in the *pDst* destination image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error when <i>border</i> has an illegal value.

## Example

To better understand usage of the `ippiLBPIImageHorizCorr` function, refer to the `LBPIImageHorizCorr.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

[Regions of Interest in Intel IPP](#)  
[Borders in Neighborhood Operations](#)  
[User-defined Border Types](#)

# Camera Calibration and 3D Reconstruction

---

## Correction of Camera Lens Distortion

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths *fx*, , and principal point coordinates *cx*, *cy*. The distortion is characterized by two coefficients of radial distortions *k<sub>1</sub>*, *k<sub>2</sub>* and two coefficients of tangential distortions *p<sub>1</sub>*, *p<sub>2</sub>*.

The undistorted coordinates *x<sub>u</sub>* and *y<sub>u</sub>* of point with coordinates (*x<sub>d</sub>*, *y<sub>d</sub>*) are computed in accordance with the following formulas:

$$x_u = x_d \cdot (1 + k_1 r^2 + k_2 r^4) + 2p_1 x_d y_d + p_2 \cdot (r^2 + 2x_d^2)$$

$$y_u = y_d \cdot (1 + k_1 r^2 + k_2 r^4) + 2p_2 x_d y_d + p_1 \cdot (r^2 + 2y_d^2)$$

Here  $r^2 = x_d^2 + y_d^2$ ,  $x_d = (j-cx)/fx$ ,  $y_d = (i-cy)/fy$ ; *i* and *j* are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

## UndistortGetSize

*Computes the size of the external buffer.*

### Syntax

```
IppStatusippiUndistortGetSize(IppiSize roiSize, int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

*roiSize* Size of source and destination images ROI in pixels.

*pBufferSize* Pointer to the computed value of the buffer size.

### Description

This function computes the size of the temporary external buffer that is used by the functions [ippiUndistortRadial](#). The buffer of the computed size can be used to process smaller images as well.

### Return Values

ippStsNoErr Indicates no error.

ippStsNullPtrErr Indicates an error if *pBufferSize* is NULL.

ippStsSizeErr Indicates an error condition if *roiSize* has a field with zero or negative value.

## UndistortRadial

*Corrects radial distortions of the single image.*

### Syntax

```
IppStatusippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f , Ipp32f cx,  
Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
```

Supported values for mod:

8u\_C1R      16u\_C1R      32f\_C1R

8u\_C3R      16u\_C3R      32f\_C3R

### Include Files

ippcv.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pSrc</i>	Pointer to the ROI in the source distorted image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination corrected image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>fx</i>	Focal lengths along the x axis.
<ify< i=""></ify<>	Focal lengths along the y axis.
<i>cx</i>	x-coordinate of the principal point.
<i>cy</i>	y-coordinate of the principal point.
<i>k1</i>	First coefficient of radial distortion.
<i>k2</i>	Second coefficient of radial distortion.
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function corrects radial distortions of the single source image *pSrc* and stores corrected image in the *pDst*. Correction is performed accounting camera parameters *fx*, *fy*, *cx*, *cy* and radial distortion parameters *k1*, *k2*. The function can also pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function [ippiUndistortGetSize](#). If a null pointer is passed, slower computations without an external buffer will be performed.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<i>ippStsNotEvenStepErr</i>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<i>ippStsBadArgErr</i>	Indicates an error if <i>fx</i> or <i>fy</i> is equal to 0.

## CreateMapCameraUndistort

*Creates look-up tables of coordinates of corrected image.*

## Syntax

```
IppStatusippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep, Ipp32f* pyMap,
int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1,
Ipp32f k2, Ipp32f p1, Ipp32f p2, Ipp8u* pBuffer);
```

## Include Files

ippcv.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

## Parameters

<i>pxMap</i>	Pointer to the destination x coordinate look-up buffer.
<i>xStep</i>	Distance in bytes between starts of consecutive lines in the <i>pxMap</i> image.
<i>pyMap</i>	Pointer to the destination y coordinate look-up buffer.
<i>yStep</i>	Distance in bytes between starts of consecutive lines in the <i>pyMap</i> image.
<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>fx</i>	Focal lengths along the <i>x</i> axis.
<ify< i=""></ify<>	Focal lengths along the <i>y</i> axis.
<i>cx</i>	<i>x</i> -coordinate of the principal point.
<i>cy</i>	<i>y</i> -coordinate of the principal point.
<i>k1</i>	First coefficient of radial distortion.
<i>k2</i>	Second coefficient of radial distortion.
<i>p1</i>	First coefficient of tangential distortion.
<i>p2</i>	Second coefficient of tangential distortion.
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with ROI (see Regions of Interest in Intel IPP ).

This function creates the look-up tables of *x*- and *y*-coordinates *pxMap* and *pyMap* respectively. These coordinates are computed in accordance with camera parameters *fx*, *fy*, *cx*, *cy*, and distortion parameters *k1*, *k2*, *p1*, *p2*. The created tables can be used by the Intel IPP function [ippiRemap](#) to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function [ippiUndistortGetSize](#). If a null pointer is passed, slower computations without an external buffer will be performed.

## Return Values

ippStsNoErr	Indicates no error.
-------------	---------------------

ippStsNullPtrErr	Indicates an error if <i>pxMap</i> or <i>pyMap</i> is <i>NULL</i> .
ippStsSizeErr	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if : <i>xStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> , or <i>yStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error when <i>fx</i> or <ify< i=""> is equal to 0.</ify<>

## Example

To better understand usage of the `ippiCreateMapCameraUndistort` function, refer to the `CreateMapCameraUndistort.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

# 3D Data Processing Functions

This section describes the Intel® Integrated Performance Primitives (Intel® IPP) functions that perform 3D data transforms - resizing, affine transform, and remapping, as well as functions for 3D data linear filtering.

## **CopyConstBorder**

*Copies pixel values between two 3D images and adds border pixels with a constant value.*

### Syntax

```
IppStatus ipprCopyConstBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
IpprVolume srcRoiVolume, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume
dstRoiVolume, int topBorderHeight, int leftBorderWidth, int forwardBorderDepth, const
Ipp8u* value);

IppStatus ipprCopyConstBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp16u* value);

IppStatus ipprCopyConstBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16s* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp16s* value);

IppStatus ipprCopyConstBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp32f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp32f* value);

IppStatus ipprCopyConstBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp64f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp64f* value);
```

### Platform-aware functions

```
IppStatus ipprCopyConstBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp8u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp8u* value);

IppStatus ipprCopyConstBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp16u* value);

IppStatus ipprCopyConstBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16s* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp16s* value);
```

```
IppStatus ipprCopyConstBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp32f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp32f* value);

IppStatus ipprCopyConstBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp64f* value);
```

## Include Files

ippi.h

ippi\_1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

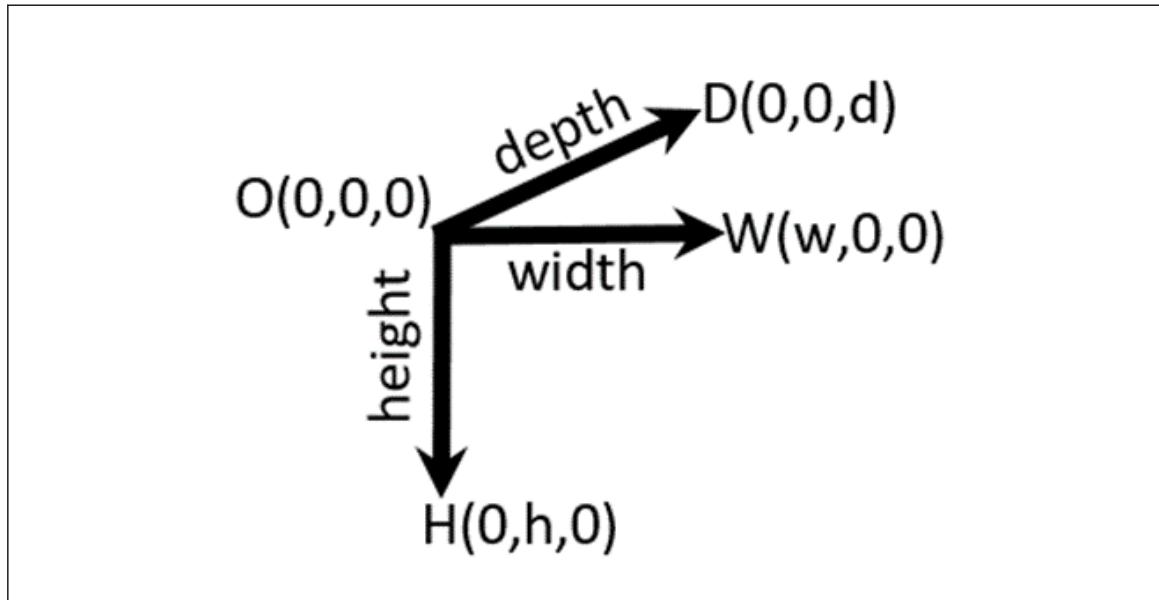
## Parameters

<i>pSrc</i>	Array of pointers to the planes in the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<i>srcRoiVolume</i>	Volume of the source ROI in pixels.
<i>pDst</i>	Array of pointers to the planes in the destination volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<i>dstRoiVolume</i>	Volume of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.
<i>forwardBorderDepth</i>	Depth of the forward border in pixels.
<i>value</i>	Constant value to assign to the border pixels.

## Description

This function operates with VOI. This function copies the source image *pSrc* with the volume *srcRoiVolume* to the destination image *pDst* with the volume *dstRoiVolume* and creates a border outside the copied area. The function sets pixel values of the border to the specified constant value that is passed by the *value* argument.

The image below shows the mapping of the parameters *topBorderHeight*, *leftBorderWidth*, and *forwardBorderDepth* onto the dimensions of the three-dimensional space.



## Return Values

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>value</code> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcPlaneStep</code> , <code>srcStep</code> , <code>dstPlaneStep</code> , or <code>dstStep</code> has a field with negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>leftBorderWidth</code> , <code>topBorderHeight</code> , or <code>forwardBorderDepth</code> has a field with negative value.

## See Also

[CopyReplicateBorder](#) Copies pixel values between two 3D images and adds replicated border pixels.

[Structures and Enumerators for Platform-Aware Functions](#)

## CopyReplicateBorder

*Copies pixel values between two 3D images and adds replicated border pixels.*

### Syntax

```
IppStatus ipprCopyReplicateBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp8u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16s* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp32f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp64f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);
```

### Platform-aware functions

```
IppStatus ipprCopyReplicateBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp8u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16s* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp32f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

```
IppStatus ipprCopyReplicateBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

### Include Files

`ippi.h`

`ippi_1.h`

### Domain Dependencies

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

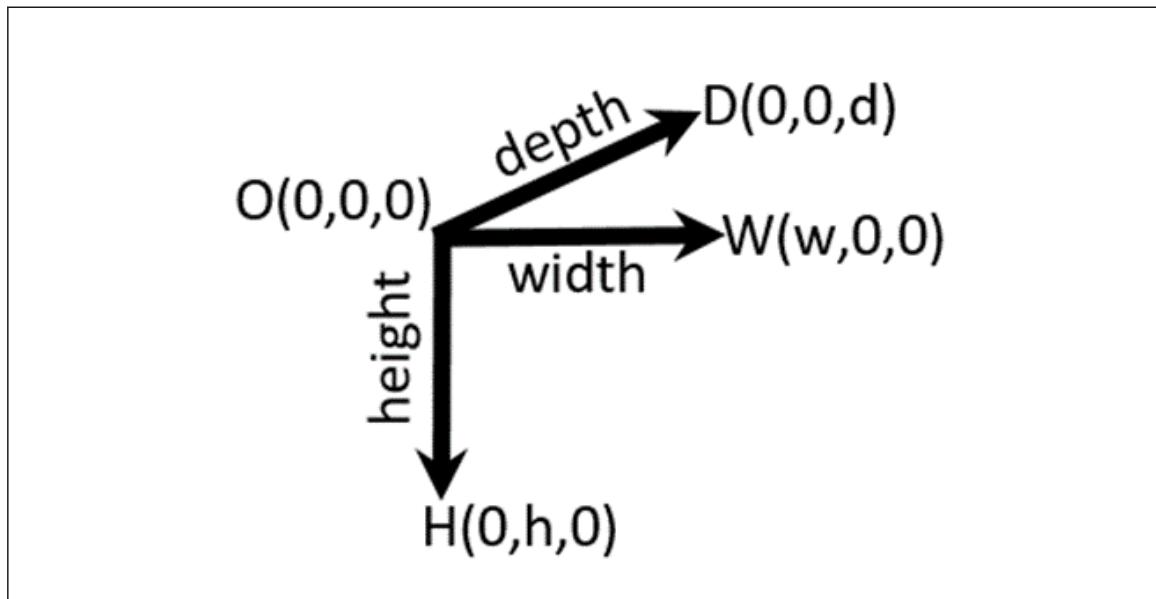
<code>pSrc</code>	Array of pointers to the planes in the source volume.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<code>srcPlaneStep</code>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<code>srcRoiVolume</code>	Volume of the source ROI in pixels.
<code>pDst</code>	Array of pointers to the planes in the destination volume.

<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<i>dstRoiVolume</i>	Volume of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.
<i>forwardBorderDepth</i>	Depth of the forward border in pixels.

## Description

This function operates with VOI. This function copies the source image *pSrc* with the volume *srcRoiVolume* to the destination image *pDst* with the volume *dstRoiVolume*. The function fills pixels ('border') outside the copied area in the destination image with the values of the source image pixels.

The image below shows the mapping of the parameters *topBorderHeight*, *leftBorderWidth*, and *forwardBorderWidth* onto the dimensions of the three-dimensional space.



## Return Values

<i>ippStsNullPtrErr</i>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcPlaneStep</i> value is less than <i>srcStep</i> value or if <i>dstPlaneStep</i> value is less than <i>dstStep</i> value.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>leftBorderWidth</i> , <i>topBorderHeight</i> or <i>forwardBorderDepth</i> has a field with negative value.

## See Also

[CopyConstBorder](#) Copies pixel values between two 3D images and adds border pixels with a constant value.

[Structures and Enumerators for Platform-Aware Functions](#)

## Filter

*Filters a volume using a general cuboidal kernel.*

### Syntax

```
IppStatus ipprFilter_16s_C1PV(const Ipp16s* const pSrc[], int srcStep, const Ipp16s* pDst[], int dstStep, IpprVolume dstVolume, const Ipp32s* pKernel, IpprVolume kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ipprFilter_16s_C1V(const Ipp16s* pSrc, int srcStep, int srcPlaneStep, Ipp16s* pDst, int dstStep, int dstPlaneStep, IpprVolume dstVolume, const Ipp32s* pKernel, IpprVolume kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>pSrc</i>	Array of pointers to the planes in the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for the <i>16s_C1V</i> flavor).
<i>pDst</i>	Array of pointers to the planes in the destination volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for the <i>16s_C1V</i> flavor).
<i>dstVolume</i>	Size of the processed volume.
<i>pKernel</i>	Pointers to the kernel values.
<i>kernelVolume</i>	Size of the kernel volume.
<i>anchor</i>	Anchor 3d-cell specifying the cuboidal kernel alignment with respect to the position of the input voxel.
<i>divisor</i>	The integer value by which the computed result is divided.
<i>pBuffer</i>	Pointer to the external buffer.

### Description

This function operates with VOI. This function uses the general cuboidal kernel of size *kernelVolume* to filter a volume VOI. This function sums the products between the kernel coefficients *pKernel* and voxel values taken over the source voxel neighborhood defined by *kernelVolume* and *anchor*. The anchor 3d-cell is specified by its coordinates *anchor.x*, *anchor.y* and *anchor.z* in the coordinate system associated with the

right bottom back corner of the kernel. Note the kernel coefficients are used in inverse order. The sum is written to the destination voxel. To ensure valid operation when volume boundary voxels are processed, the application must correctly define additional border voxels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , <code>pKernel</code> or <code>pBuffer</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstVolume</code> or <code>kernelVolume</code> has a field with zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the divisor value is zero.

## FilterGetBufSize

*Calculates the size of the working buffer.*

---

### Syntax

```
IppStatus ipprFilterGetBufSize(IpprVolume dstVolume, IpprVolume kernelVolume, int nChannel, int* pSize);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters

<code>dstVolume</code>	Size of the processed volume.
<code>kernelVolume</code>	Size of the kernel volume.
<code>nChannel</code>	Number of channels or planes, possible value is one.
<code>pSize</code>	Pointer to the size of the external buffer.

### Description

This function operates with VOI. This function computes the size of the working buffer `pSize` that is required for the function `ipprFilter`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSize</code> pointer is <code>NULL</code> .
<code>ippStsNumChannelErr</code>	Indicates an error condition if <code>nChannel</code> has an illegal value.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstVolume</code> or <code>kernelVolume</code> has a field with zero or negative value.

## FilterBorder

*Filters a 3D image using a rectangular filter.*

### Syntax

```
IppStatus ipprFilterBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);
```

### Platform-aware functions

```
IppStatus ipprFilterBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL
srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

**Threading Layer (TL) functions based on the Platform Aware API**

```
IppStatus ipprFilterBorder_8u_C1V_LT(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL
srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_LT(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_LT(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_LT(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_LT(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);
```

**Threading Layer (TL) functions based on the Classic API**

```
IppStatus ipprFilterBorder_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);
```

**Include Files**

ippi.h  
ippi\_1.h  
ippi\_tl.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>pSrc</code>	Array of pointers to the planes in the source volume.
<code>srcStep</code>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<code>srcPlaneStep</code>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<code>pDst</code>	Array of pointers to the planes in the destination volume.
<code>dstStep</code>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<code>dstPlaneStep</code>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<code>dstRoiVolume</code>	Volume of the destination ROI in pixels.
<code>borderType</code>	Type of the border. Possible values are:
	<code>ipprBorderInMem</code> Border is obtained from the source image pixels in memory.
	<code>ipprBorderRepl</code> Border is replicated from the edge pixels.
	<code>ipprBorderConst</code> Border is replicated from the edge pixels.
<code>borderValue</code>	Constant value to assign to pixels of the constant border.
<code>pSpec</code>	Pointer to the filter specification structure.
<code>pBuffer</code>	Pointer to the work buffer for filtering operations.

## Description

Before using this function, you need to initialize the filter specification structure for 3D image processing using the `ipprFilterBorderInit` function.

This function operates with VOI. This function performs linear filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error condition. Any other value indicates an error condition.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcPlaneStep</code> , <code>srcStep</code> , <code>dstPlaneStep</code> , or <code>dstStep</code> has a field with negative value.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pSrc</code> , <code>pDst</code> , <code>pSpec</code> , or <code>pBuffer</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiVolume</code> has a field with zero or negative value.

## Example

To better understand usage of this function, refer to the `FilterBorder3d.c` example in the examples archive available for download from <https://software.intel.com/en-us/ipp-manual-examples>.

## See Also

`FilterBorderInit` Initializes the filter specification structure for 3D image processing.

`FilterBorderGetSize` Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

[Structures and Enumerators](#)

[Structures and Enumerators for Platform-Aware Functions](#)

## FilterBorderInit

---

*Initializes the filter specification structure for 3D image processing.*

### Syntax

```
IppStatus ipprFilterBorderInit_16s(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

IppStatus ipprFilterBorderInit_32f(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

IppStatus ipprFilterBorderInit_64f(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

### Platform-aware functions

```
IppStatus ipprFilterBorderInit_16s_L(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

IppStatus ipprFilterBorderInit_32f_L(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

IppStatus ipprFilterBorderInit_64f_L(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

### Threading Layer (TL) functions based on the Platform Aware API

```
IppStatus ipprFilterBorderInit_16s_LT(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);

IppStatus ipprFilterBorderInit_32f_LT(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);

IppStatus ipprFilterBorderInit_64f_LT(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
```

### Threading Layer (TL) functions based on the Classic API

```
IppStatus ipprFilterBorderInit_16s_T(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);

IppStatus ipprFilterBorderInit_32f_T(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);

IppStatus ipprFilterBorderInit_64f_T(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
```

## Include Files

ippi.h  
ippi\_1.h  
ippi\_t1.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>kernelVolume</i>	Size of the kernel volume.
<i>pKernel</i>	Pointers to the kernel values.
<i>dataType</i>	Data type of the source image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<i>divisor</i>	The integer value by which the computed result is divided.
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pSpec</i>	Pointer to the filter specification structure.

## Description

This function operates with VOI. This function initializes the filter specification structure *pSpec*. Before using this function, you need to compute the size of the specification structure for 3D image processing using the `ipprFilterBorderGetSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error condition. Any other value indicates an error condition.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pSpec</i> or <i>pKernel</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelVolume</i> has a field with zero or negative value.
<code>ippStsChannelErr</code>	Indicates an error condition if <i>numChannels</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <i>dataType</i> has an illegal value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <i>divisor</i> value is zero.

## See Also

[FilterBorder](#) Filters a 3D image using a rectangular filter.

[FilterBorderGetSize](#) Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

[Structures and Enumerators for Platform-Aware Functions](#)

## FilterBorderGetSize

*Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.*

### Syntax

```
IppStatus ipprFilterBorderGetSize(IpprVolume kernelVolume, IpprVolume dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

### Platform-aware function

```
IppStatus ipprFilterBorderGetSize_L(IpprVolumeL kernelVolume, IpprVolumeL dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pSpecSize,
IppSizeL* pBufferSize);
```

### Threading Layer (TL) function based on the Platform Aware API

```
IppStatus ipprFilterBorderGetSize_LT(IpprVolumeL kernelVolume, IpprVolumeL dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pSpecSize,
IppSizeL* pBufferSize);
```

### Threading Layer (TL) function based on the Classic API

```
IppStatus ipprFilterBorderGetSize_T(IpprVolume kernelVolume, IpprVolume dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

### Include Files

ippi.h  
ippi\_l.h  
ippi\_tl.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>kernelVolume</i>	Size of the kernel volume.
<i>dstRoiVolume</i>	Maximal size of the destination image ROI (in pixels).
<i>dataType</i>	Data type of the source image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<i>kernelType</i>	Data type of the filter kernel. Possible values are <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<i>numChannels</i>	Number of channels in the image. Possible value is 1.
<i>pSpecSize</i>	Pointer to the size of the filter specification structure.
<i>pBufferSize</i>	Pointer to the size of the work buffer required for filtering.

## Description

This function operates with VOI. This function computes the size of the filter specification structure *pSpec* and the size of the buffer required for 3D image filtering operations. Call this function before using the `ipprFilterBorderInit` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error condition. Any other value indicates an error condition.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <i>pSpecSize</i> or <i>pBufferSize</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiVolume</i> or <i>kernelVolume</i> has a field with zero or negative value.
<code>ippStsChannelErr</code>	Indicates an error condition if <i>numChannels</i> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if the combination of <i>kernelType</i> and <i>dataType</i> has an illegal value.

## See Also

[FilterBorder](#) Filters a 3D image using a rectangular filter.

[FilterBorderInit](#) Initializes the filter specification structure for 3D image processing.

[Structures and Enumerators for Platform-Aware Functions](#)

## FilterMedian

---

*Filters a 3D image using a median filter.*

---

### Syntax

```
IppStatus ipprFilterMedian_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);
```

## Threading Layer (TL) functions

```
IppStatus ipprFilterMedian_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);
```

## Include Files

ippi.h  
ippi\_tl.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Array of pointers to the planes in the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<i>pDst</i>	Array of pointers to the planes in the destination volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
<i>dstRoiVolume</i>	Volume of the destination ROI in pixels.
<i>borderType</i>	Type of the border. Possible values are:

	ipprBorderInMem	Border is obtained from the source image pixels in memory.
	ipprBorderRepl	Border is replicated from the edge pixels.
	ipprBorderConst	Border is replicated from the edge pixels.
<i>pBorderValue</i>		Pointer to the constant value to assign to pixels of the constant border.
<i>pSpec</i>		Pointer to the filter specification structure.
<i>pBuffer</i>		Pointer to the work buffer for filtering operations.

## Description

Before using this function, you need to initialize the filter specification structure for 3D image processing using the [ipprFilterMedianInit](#) function.

This function operates with VOI. This function performs median filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

## Return Values

ippStsNoErr	Indicates no error condition. Any other value indicates an error condition.
ippStsStepErr	Indicates an error condition if <i>srcPlaneStep</i> , <i>srcStep</i> , <i>dstPlaneStep</i> , or <i>dstStep</i> has a field with negative value.
ippStsNullPtrErr	Indicates an error condition if the <i>pSrc</i> , <i>pDst</i> , <i>pSpec</i> , <i>pBorderValue</i> , or <i>pBuffer</i> pointer is NULL.
ippStsSizeErr	Indicates an error condition if <i>dstRoiVolume</i> has a field with zero or negative value.

## See Also

[FilterMedianInit](#) Initializes the filter specification structure for 3D image processing with a median filter.

[FilterMedianGetSize](#) Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

[Structures and Enumerators](#)

## FilterMedianInit

*Initializes the filter specification structure for 3D image processing with a median filter.*

### Syntax

```
IppStatus ipprFilterMedianInit(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec* pSpec);
```

### Threading Layer (TL) function

```
IppStatus ipprFilterMedianInit_T(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec_T* pSpec);
```

### Include Files

ippi.h

ippi\_tl.h

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>maskVolume</code>	Size of the mask volume.
<code>dataType</code>	Data type of the source image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<code>numChannels</code>	Number of channels in the image. Possible value is 1.
<code>pSpec</code>	Pointer to the filter specification structure.

## Description

This function operates with VOI. This function initializes the filter specification structure `pSpec` for 3D image processing with a median filter. Before using this function, you need to compute the size of the corresponding specification structure using the [ipprFilterMedianGetSize](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error condition. Any other value indicates an error condition.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pSpec</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskVolume</code> has a field with zero or negative value.
<code>ippStsChannelErr</code>	Indicates an error condition if <code>numChannels</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> has an illegal value.

## See Also

[FilterMedianGetSize](#) Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

[FilterMedian](#) Filters a 3D image using a median filter.

## FilterMedianGetSize

---

*Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.*

---

## Syntax

```
IppStatus ipprFilterMedianGetSize(IpprVolume maskVolume, IpprVolume dstRoiVolume,  
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

## Threading Layer (TL) function

```
IppStatus ipprFilterMedianGetSize_T(IpprVolume maskVolume, IpprVolume dstRoiVolume,  
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

## Include Files

`ippi.h`

`ippi_tl.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>maskVolume</code>	Size of the mask volume.
<code>dstRoiVolume</code>	Maximal size of the destination image ROI (in pixels).
<code>dataType</code>	Data type of the source image. Possible values are <code>ipp8u</code> , <code>ipp16u</code> , <code>ipp16s</code> , <code>ipp32f</code> , and <code>ipp64f</code> .
<code>numChannels</code>	Number of channels in the image. Possible value is 1.
<code>pSpecSize</code>	Pointer to the size of the filter specification structure.
<code>pBufferSize</code>	Pointer to the size of the work buffer required for filtering.

## Description

This function operates with VOI. This function computes the size of the filter specification structure `pSpec` and the size of the buffer required for 3D image filtering operations with a median filter. Call this function before using the `ipprFilterMedianInit` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error condition. Any other value indicates an error condition.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the <code>pSpecSize</code> or <code>pBufferSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiVolume</code> or <code>maskVolume</code> has a field with zero or negative value.
<code>ippStsChannelErr</code>	Indicates an error condition if <code>numChannels</code> has an illegal value.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> has an illegal value.

## See Also

[FilterMedianInit](#) Initializes the filter specification structure for 3D image processing with a median filter.

[FILterMedian](#) Filters a 3D image using a median filter.

## ResizeGetBufSize

*Calculates the size of the external work buffer for the function `ipprResize`.*

### Syntax

```
IppStatus ipprResizeGetBufSize(IpprCuboid srcVoi, IpprCuboid dstVoi, int nChannel, int interpolation, int* pSize);
```

### Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<code>srcVoi</code>	Volume of interest in the source volume.
<code>dstVoi</code>	Volume of interest in the destination volume.
<code>nChannel</code>	Number of channels, possible value: 1.
<code>interpolation</code>	Type of interpolation, the following values are possible:  IPPI_INTER_NN- nearest neighbor interpolation, IPPI_INTER_LINEAR- trilinear interpolation, IPPI_INTER_CUBIC- tricubic interpolation, IPPI_INTER_CUBIC2P_BSPLINE- B-spline, IPPI_INTER_CUBIC2P_CATMULLROM- Catmull-Rom spline, IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).
<code>pSize</code>	Pointer to the size of the external buffer.

## Description

This function calculates the size of the external buffer required for the `ipprrResize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error when the <code>pSize</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error when width, or height, or depth of the <code>srcVoi</code> or <code>dstVoi</code> is less than, or equal to 0.
<code>ippStsNumChannelErr</code>	Indicates an error when <code>nChannel</code> is not equal to 1.
<code>ippStsInterpolationErr</code>	Indicates an error when <code>interpolation</code> has an illegal value.

## See Also

`Resize` Resizes the source volume.

## GetResizeCuboid

---

*Computes coordinates of the destination cuboid.*

## Syntax

```
IppStatus ipprGetResizeCuboid(IpprCuboid srcVoi, IpprCuboid* pDstCuboid, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation);
```

## Include Files

`ippi.h`

## Domain Dependencies

Headers: `ippcore.h, ippvm.h, ipps.h`

Libraries: `ippcore.lib, ippvm.lib, ipps.lib`

## Parameters

<code>srcVoi</code>	Volume of interest of the source volume.
<code>pDstCuboid</code>	Pointer to the destination cuboid.
<code>x-, y-, zFactor</code>	Factors by which the <code>x</code> , <code>y</code> , <code>z</code> dimensions of the source VOI are changed.
<code>x-, y-, zShift</code>	Shift values in the <code>x</code> , <code>y</code> , and <code>z</code> directions respectively.
<code>interpolation</code>	Type of interpolation, the following values are possible:  IPPI_INTER_NN- nearest neighbor interpolation, IPPI_INTER_LINEAR- trilinear interpolation, IPPI_INTER_CUBIC- tricubic interpolation, IPPI_INTER_CUBIC2P_BSPLINE- B-spline, IPPI_INTER_CUBIC2P_CATMULLROM- Catmull-Rom spline, IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).

## Description

This function operates with volume of interest (VOI).

This function computes the coordinates of the resultant cuboid which is obtained if the source volume `srcVoi` is resized with the specified parameters. The resize operation is not performed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <code>pDstCuboid</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to 0.
<code>ippStsResizeFactorErr</code>	Indicates an error when one of the <code>xFactor</code> , <code>yFactor</code> , <code>zFactor</code> values is less than, or equal to 0.
<code>ippStsInterpolationErr</code>	Indicates an error when <code>interpolation</code> has an illegal value.

## Resize

*Resizes the source volume.*

### Syntax

```
IppStatus ipprResize_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep, IpprCuboid
dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift,
double zShift, int interpolation, Ipp8u* pBuffer);
```

```
IppStatus ipprResize_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift,
double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int
srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, double
xFactor, double yFactor, double zFactor, double xShift, double yShift, double
zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int
srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi,
double xFactor, double yFactor, double zFactor, double xShift, double yShift, double
zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift,
double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift,
double yShift, double zShift, int interpolation, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Pointer to the source volume origin. An array of pointers to the source planes for non-contiguous volume.
<i>srcVolume</i>	Size of the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the planes of the source contiguous volume.
<i>srcVoi</i>	Volume of interest of the source volume.
<i>pDst</i>	Pointer to the destination volume origin. An array of pointers to the destination planes for non-contiguous volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the planes of the destination contiguous volume.
<i>dstVoi</i>	Volume of interest of the destination volume.
<i>x-, y-, zFactor</i>	Factors by which the <i>x</i> , <i>y</i> , <i>z</i> dimensions of the source VOI are changed.

*x-, y-, zShift*Shift values in the *x*, *y*, and *z* directions respectively.*interpolation*

Type of interpolation, the following values are possible:

IPPI\_INTER\_NN- nearest neighbor interpolation,  
 IPPI\_INTER\_LINEAR- trilinear interpolation,  
 IPPI\_INTER\_CUBIC- tricubic interpolation,  
 IPPI\_INTER\_CUBIC2P\_BSPLINE- B-spline,  
 IPPI\_INTER\_CUBIC2P\_CATMULLROM- Catmull-Rom spline,  
 IPPI\_INTER\_CUBIC2P\_B05C03- special two-parameters filter (1/2, 3/10).

*pBuffer*

Pointer to the external buffer.

## Description

This function operates with volume of interest (VOI).

This function resizes the source volume *srcVoi* by *xFactor* in the *x* direction, *yFactor* in the *y* direction and *zFactor* in the *z* direction. The volume size can be reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*, *zFactor*. If the value of the certain factor is greater than 1, the volume size is increased, and if it is less than 1, the volume size is reduced in the corresponding direction. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination volumeVOI.

Coordinates *x'*, *y'*, and *z'* in the resized volume are obtained from the equations:

$$x' = xFactor * x + xShift$$

$$y' = yFactor * y + yShift$$

$$z' = zFactor * z + zShift$$

where *x*, *y*, and *z* denote the coordinates of the element in the source volume. The right coordinate system (RCS) is used here.

Descriptor *PV* means working with non-contiguous volume data, where the location of the planes is passed to the function using pointers on the plane. In functions with descriptor *V*, the location of the planes is determined by the distance in bytes between the starting points of successive lines in each plane of the original volume (contiguous volume data).

In case of resize:

- In IppStatus ipprResize\_8u\_C1V function *V* means:  
`const Ipp8u* pSrc` - Pointer to the source volume origin. `srcPlaneStep` - Distance, in bytes, between the planes of the source contiguous volume.
- In IppStatus ipprResize\_8u\_C1PV function *PV* means:  
`const Ipp8u* const pSrc[]` - An array of pointers to the source planes for non-contiguous volume.

Before using this function, compute the size of the external buffer *pBuffer* using [ipprResizeGetBufSize](#).

## Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error when width, or height, or depth of the source and destination volumes is less than or equal to 0.

ippStsResizeFactorErr	Indicates an error when one of the <i>xFactor</i> , <i>yFactor</i> , <i>zFactor</i> pointers is less than, or equal to 0.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.
ippStsWrongIntersectVOI	Indicates a warning when <i>srcVoi</i> has no intersection with the source volume, operation is not performed.

## See Also

[ResizeGetBufSize](#) Calculates the size of the external work buffer for the function `ipprResize`.

## Remap

---

Performs the look-up coordinate mapping of the elements of the source volume.

### Syntax

#### Case 1: Operation on non-contiguous volume data

```
IppStatus ipprRemap_8u_C1PV(const Ipp8u* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp8u* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1PV(const Ipp16u* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp16u* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1PV(const Ipp32f* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp32f* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);
```

#### Case 2: Operation on contiguous volume data

```
IppStatus ipprRemap_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp8u* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp16u* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp32f* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);
```

### Include Files

`ippi.h`

### Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

## Parameters

<i>pSrc</i>	Array of pointers to the planes in the source volume.
<i>srcVolume</i>	Size of the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for <code>8u_C1V</code> , <code>16u_C1V</code> , and <code>32f_C1V</code> flavors).
<i>srcVoi</i>	Region of interest in the source volume.
<i>pxMap</i> , <i>pyMap</i> , <i>pzMap</i>	Arrays of pointers to the starts of the 2D buffers, containing tables of the x-, y- and z-coordinates. If the referenced coordinates correspond to a voxel outside of the source VOI, no mapping of the source pixel is done.
<i>mapStep</i>	Step, in bytes, through the buffers containing tables of the x-, y- and z-coordinates.
<i>mapPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the buffers containing tables (for <code>8u_C1V</code> , <code>16u_C1V</code> , and <code>32f_C1V</code> flavors).
<i>pDst</i>	Array of the pointers to the planes in the destination volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for <code>8u_C1V</code> , <code>16u_C1V</code> , and <code>32f_C1V</code> flavors).
<i>dstVolume</i>	Size of the destination volume.
<i>interpolation</i>	The type of interpolation, the following values are possible:  IPPI_INTER_NN- nearest neighbor interpolation, IPPI_INTER_LINEAR- trilinear interpolation, IPPI_INTER_CUBIC- tricubic interpolation, IPPI_INTER_CUBIC2P_BSPLINE- B-spline, IPPI_INTER_CUBIC2P_CATMULLROM- Catmull-Rom spline, IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).

## Description

This function operates with volume of interest (VOI).

This function transforms the source volume by remapping its voxels. Voxel remapping is performed using *pxMap*, *pyMap* and *pzMap* buffers to look-up the coordinates of the source volume voxel that is written to the target destination volume voxel. The application has to supply these look-up tables.

The remapping of the source voxels to the destination voxels is made according to the following formulas:

$$\text{dst\_voxel}[i, j, k] = \text{src\_voxel}[\text{pxMap}[i, j, k], \text{pyMap}[i, j, k], \text{pzMap}[i, j, k]]$$

where *i*, *j*, *k* are the x-, y- and z-coordinates of the target destination volume voxel *dst\_voxel*;

*pxMap*[*i*, *j*, *k*] contains the x-coordinates of the source volume voxels *src\_voxel* that are written to *dst\_voxel*.

*pyMap[i, j, k]* contains the y-coordinates of the source volume voxels *src\_voxel* that are written to *dst\_voxel*.

*pzMap[i, j, k]* contains the z-coordinates of the source volume voxels *src\_voxel* that are written to *dst\_voxel*.

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width, or height, or depth of the source and destination volumes has zero or negative value.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.
ippStsWrongIntersectVOI	Indicates a warning when <i>srcVoi</i> has no intersection with the source volume, operation is not performed.

## WarpAffineGetBufSize

---

*Calculates the size of the external buffer for the affine transform.*

### Syntax

```
IppStatus ipprWarpAffineGetBufSize(IpprVolume srcVolume, IpprCuboid srcVoi, IpprCuboid dstVoi, const double coeffs[3][4], int nChannel, int interpolation, int* pSize);
```

### Include Files

ippi.h

### Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

### Parameters

<i>srcVolume</i>	Size of the source volume.
<i>srcVoi</i>	Region of interest of the source volume.
<i>dstVoi</i>	Region of interest of the destination volume.
<i>coeffs</i>	Affine transform matrix.
<i>nChannel</i>	Number of channel or planes, possible value is 1.
<i>interpolation</i>	Type of interpolation, the following values are possible:  IPPI_INTER_NN- nearest neighbor interpolation, IPPI_INTER_LINEAR- trilinear interpolation, IPPI_INTER_CUBIC- tricubic interpolation, IPPI_INTER_CUBIC2P_BSPLINE- B-spline, IPPI_INTER_CUBIC2P_CATMULLROM- Catmull-Rom spline,

IPPI\_INTER\_CUBIC2P\_B05C03- special two-parameters filter (1/2, 3/10).

*pSize*

Pointer to the size of the external buffer.

## Description

This function calculates the size (in bytes) of the external buffer that is required for the [ipprWarpAffine](#) function. (In some cases the function returns zero size of the buffer).

## Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when <i>pSize</i> or <i>coeffs</i> is NULL.
ippStsSizeErr	Indicates an error if width, or height, or depth of the <i>srcVoi</i> or <i>dstVoi</i> is less than, or equal to zero.
ippStsNumChannelErr	Indicates an error when <i>nChannel</i> has an illegal value.
ippStsInterpolationErr	Indicates an error when <i>interpolation</i> has an illegal value.

## See Also

[WarpAffine](#) Performs the general affine transform of the source volume.

## WarpAffine

---

Performs the general affine transform of the source volume.

---

### Syntax

```
IppStatus ipprWarpAffine_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
```

## Include Files

ippi.h

## Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

## Parameters

<i>pSrc</i>	Array of pointers to the planes in the source volume.
<i>srcVolume</i>	Size, in pixels, of the source volume.
<i>srcStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
<i>srcPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
<i>srcVoi</i>	Volume of interest of the source volume.
<i>pDst</i>	Array of pointers to the planes in the destination volume.
<i>dstVoi</i>	Volume of interest of the destination volume.
<i>dstStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.
<i>dstPlaneStep</i>	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
<i>coeffs</i>	Coefficients of the affine transform.
<i>interpolation</i>	Type of interpolation, the following values are possible: IPPI_INTER_NN- nearest neighbor interpolation, IPPI_INTER_LINEAR- trilinear interpolation, IPPI_INTER_CUBIC- tricubic interpolation, IPPI_INTER_CUBIC2P_BSPLINE- B-spline, IPPI_INTER_CUBIC2P_CATMULLROM- Catmull-Rom spline, IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).
<i>pBuffer</i>	Pointer to the external buffer.

## Description

This function operates with volume of interest (VOI).

This affine warp function transforms the coordinates ( $x, y, z$ ) of the source volume voxels according to the following formulas:

$$x' = c_{00} * x + c_{01} * y + c_{02} * z + c_{03}$$

$$y' = c_{10} * x + c_{11} * y + c_{12} * z + c_{13}$$

$$z' = c_{20} * x + c_{21} * y + c_{22} * z + c_{23}$$

where  $x'$ ,  $y'$  and  $z'$  denote the voxel coordinates in the transformed volume, and  $c_{ij}$  are the affine transform coefficients stored in the array `coeffs`.

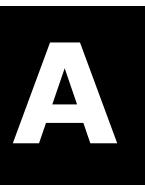
Before calling this function, compute the size of the external buffer `pBuffer` using the `ipprWarpAffineGetBufSize` function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to zero.
<code>ippStsCoeffErr</code>	Indicates an error when determinant of the transform matrix $c_{ij}$ is equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error when <code>interpolation</code> has an illegal value.
<code>ippStsWrongIntersectVOI</code>	Indicates a warning when <code>srcVoi</code> has no intersection with the source volume, operation is not performed.

## See Also

`WarpAffineGetBufSize` Calculates the size of the external buffer for the affine transform.



# Handling of Special Cases

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

Table A-1 below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with Err (except for the ippStsNoErr status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

## **Special Cases for Intel IPP Image Processing Functions**

<b>Function Base Name</b>	<b>Data Type</b>	<b>Case Description</b>	<b>Result Value</b>	<b>Status Code</b>
ippiSqrt	16s	Sqrt (x <0)	0	ippStsSqrtNegArg
	32f	Sqrt (x <0)	NAN_32F	ippStsSqrtNegArg
ippiDiv	8u	Div (0/0)	0	ippStsDivByZero
		Div (x/0)	IPP_MAX_8U	ippStsDivByZero
ippiDivC	16s	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_16S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_16S	ippStsDivByZero
ippiDiv	32f	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero
16sc	Div (0/0)	0	ippStsDivByZero	
	Div (x/0),	0	ippStsDivByZero	
32sc	Div (0/0)	0	ippStsDivByZero	
		Div (x/0), x>0	IPP_MAX_32S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_32S	ippStsDivByZero
32fc	Div (0/0)	NAN_32F		
		Div (x/0), x>0	INF_32F	
		Div (x/0), x<0	INF_NEG_32F	
ippiDivC	all	Div(x/const), const=0	-	ippStsDivByZeroErr

<b>Function Base Name</b>	<b>Data Type</b>	<b>Case Description</b>	<b>Result Value</b>	<b>Status Code</b>
<a href="#">ippiLn</a>	8u	Ln (0)	0	ippStsLnZeroArg
	16s	Ln (0)	IPP_MIN_16S	ippStsLnZeroArg
		Ln ( $x < 0$ )	IPP_MIN_16S	ippStsLnNegArg
<a href="#">ippiExp</a>	32f	Ln ( $x < 0$ )	NAN_32F	ippStsLnNegArg
		Ln ( $x < \text{IPP\_MINABS\_32F}$ )	INF_NEG_32F	ippStsLnZeroArg
		overflow	IPP_MAX_8U	ippStsNoErr
<a href="#">ippiExp</a>	16s	overflow	IPP_MAX_16S	ippStsNoErr
	32f	overflow	INF_32F	ippStsNoErr

Here  $x$  denotes an input value. For the definition of the constants used, see [Image Data Types and Ranges](#) in chapter 2.

Note that flavors of the same math function operating on different data types may produce different results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different [descriptors](#) in their names. For example, logarithm function `ippiLn` operating on `16s` data treats zero argument values in the same way for all its flavors `ippiLn_16s_C1RSfs`, `ippiLn_16s_C3RSfs`, `ippiLn_16s_C1IRSfs`, and `ippiLn_16s_C3IRSfs`.



# Interpolation in Image Geometric Transform Functions

This appendix describes the interpolation algorithms used in the geometric transformation functions of Intel IPP. For more information about each of the geometric transform functions, see [Chapter 12](#).

## Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. The library supports the following interpolation modes depending on how the type of interpolation algorithm is specified:

- **Type 1:** Application code specifies the interpolation mode by passing the *interpolation* parameter of `int` type to a processing function. The following interpolation modes are supported:
  - Nearest neighbor interpolation (`interpolation = IPPI_INTER_NN`)
  - Linear interpolation (`interpolation = IPPI_INTER_LINEAR`)
  - Cubic interpolation (`interpolation = IPPI_INTER_CUBIC`)
  - Interpolation with Lanczos window function (`interpolation = IPPI_INTER_LANZOS`)
  - Interpolation with two-parameter cubic filters with the fixed coefficients (`interpolation` can be set to the following:
    - `IPPI_INTER_CUBIC2P_BSPLINE (B=1; C=0)`
    - `IPPI_INTER_CUBIC2P_CATMULLROM (B=0; C=0.5)`
    - `IPPI_INTER_CUBIC2P_B05C03 (B=0.5; C=0.3)`
- **Type 2:** Interpolation mode is specified explicitly in a function name suffix:
  - Nearest neighbor interpolation (pass `interpolation = ippNearest` to `GetSize` functions, use the functions with the `Nearest` suffix, for example, `ResizeNearestInit` or `ResizeNearest`)
  - Linear interpolation (pass `interpolation = ippLinear` to `GetSize` functions, use the functions with the `Linear` suffix, for example, `ResizeLinearInit` or `ResizeLinear`)
  - Interpolation with two-parameter cubic filters (pass `interpolation = ippCubic` to `GetSize` functions, use the functions with the `Cubic` suffix, for example, `ResizeCubicInit` or `ResizeCubic`)
  - Supersampling (pass `interpolation = ippSuper` to `GetSize` functions, use the functions with the `Super` suffix, for example, `ResizeSuperInit` or `ResizeSuper`)
  - Interpolation with Lanczos window function (pass `interpolation = ippLanczos` to `GetSize` functions, use the functions with the `Lanczos` suffix, for example, `ResizeLanczosInit` or `ResizeLanczos`)

For certain functions of type 1, the specified interpolation modes can be combined with additional *smoothing of edges* to which the borders of the original image are transformed. To use this option, for the *interpolation* parameter, specify the edge smoothing flag and the desired interpolation mode using the bitwise OR operation. For example, in order to rotate an image with cubic interpolation and smooth the rotated image edges, pass the following value to `ippiRotate()`:

```
interpolation = IPPI_INTER_CUBIC | IPPI_SMOOTH_EDGE.
```

To enable edge smoothing for functions of type 2, pass the special flag to the `Init` function (if it exists), for example, you can pass this flag to `Init` functions for `WarpAffine` and `WarpPerspective` function groups.

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

Table B-1 lists the supported interpolation modes for the main geometric transform functions that use interpolation.

### Interpolation Modes Supported by Image Geometric Transform Functions

Function Base Name	NN	Lin	Cub2P	Cub	CR	La2	La3	SS	AA	ES
ResizeCubic			x		x *)					
ResizeLanczos						x	x			
ResizeLinear		x								
ResizeNearest	x									
ResizeSuper								x		
WarpAffineCubic			x		x *)					
WarpAffineLinear		x								
WarpAffineNearest	x									
WarpPerspectiveCubic			x		x *)					
WarpPerspectiveLinear		x								
WarpPerspectiveNearest	x									
Remap	x	x		x	x		x	n/a	n/a	x
WarpBilinear	x	x		x				n/a	n/a	x
WarpBilinearBack	x	x		x				n/a	n/a	
WarpBilinearQuad	x	x		x				n/a	n/a	x

\*) The function `ippiResizeCubic` supports the interpolation with two-parameter cubic filters, where parameters B and C can be specified explicitly.

Here **NN** - nearest neighbor interpolation, **Lin** - linear interpolation, **Cub2P** - interpolation with two-parameter cubic filters, **Cub** - cubic interpolation, **CR** - Catmull-Rom spline, **La2**, **La3** - interpolation with the Lanczos window, **SS** - super sampling interpolation, **AA** - antialiasing, **ES** - edge smoothing.

The sections that follow provide more details on each interpolation mode.

## Mathematical Notation

In this appendix the following notation is used:

$(x_D, y_D)$	pixel coordinates in the destination image (integer values);
$(x_S, y_S)$	the computed coordinates of a point in the source image that is mapped exactly to $(x_D, y_D)$ ;
$S(x, y)$	pixel value (intensity) in the source image;
$D(x, y)$	pixel value (intensity) in the destination image.

## Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point

$$(x_S, y_S) : D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S)).$$

To use the nearest neighbor interpolation, set the `interpolation` parameter to `IPPI_INTER_NN` or use the functions with the Nearest suffix (pass `interpolation=ippNearest` to `GetSize` functions).

## Linear Interpolation

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels  $(x_{S0}, y_{S0})$ ,  $(x_{S1}, y_{S0})$ ,  $(x_{S0}, y_{S1})$ ,  $(x_{S1}, y_{S1})$  that are closest to  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S), x_{S1} = x_{S0} + 1, y_{S0} = \text{int}(y_S), y_{S1} = y_{S0} + 1.$$

First, the intensity values are interpolated along the x-axis to produce two intermediate results  $I_0$  and  $I_1$  (see Figure B-1):

$$I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0}) * (x_{S1} - x_S) + S(x_{S1}, y_{S0}) * (x_S - x_{S0})$$

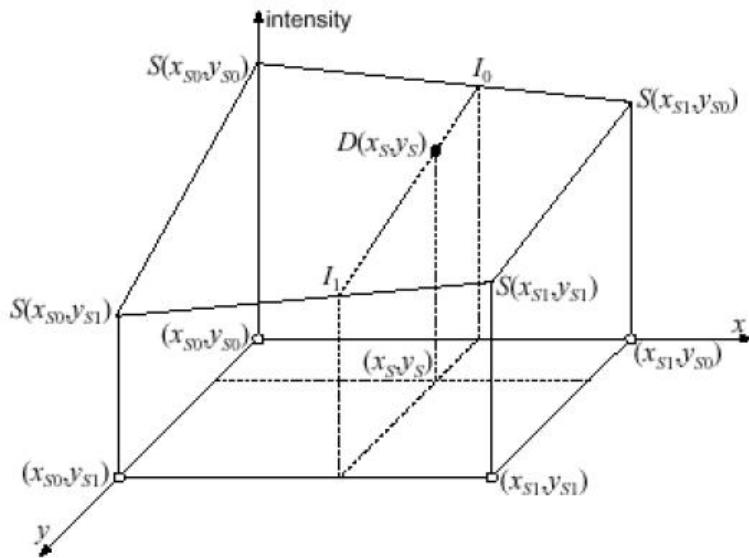
$$I_1 = S(x_S, y_{S1}) = S(x_{S0}, y_{S1}) * (x_{S1} - x_S) + S(x_{S1}, y_{S1}) * (x_S - x_{S0}).$$

Then, the sought-for intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_0$  and  $I_1$  along the y-axis:

$$D(x_D, y_D) = I_0 * (y_{S1} - y_S) + I_1 * (y_S - y_{S0}).$$

To use the linear interpolation, set the *interpolation* parameter to `IPPI_INTER_LINEAR` or use the functions with the `Linear` suffix (pass `interpolation=ippLinear` to `GetSize` functions). For images with 8-bit unsigned color channels, the `ippiWarpAffine` and `ippiResizeLinear` functions compute the coordinates  $(x_S, y_S)$  with the accuracy  $2^{-16} = 1/65536$ . For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

### Linear Interpolation



## Cubic Interpolation

The cubic interpolation algorithm (see Figure B-2) uses source image intensities at sixteen pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 1; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3;$$

$$y_{S0} = \text{int}(y_S) - 1; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3.$$

First, for each  $y_{Sk}$  the algorithm determines four cubic polynomials  $F_0(x)$ ,  $F_1(x)$ ,  $F_2(x)$ , and  $F_3(x)$ :

$$F_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k, 0 \leq k \leq 3$$

such that

$$F_k(x_{S0}) = S(x_{S0}, y_{Sk}); F_k(x_{S1}) = S(x_{S1}, y_{Sk}),$$

$$F_k(x_{S2}) = S(x_{S2}, y_{Sk}); F_k(x_{S3}) = S(x_{S3}, y_{Sk}).$$

In Figure B-2 , these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial  $F_y(y)$  such that

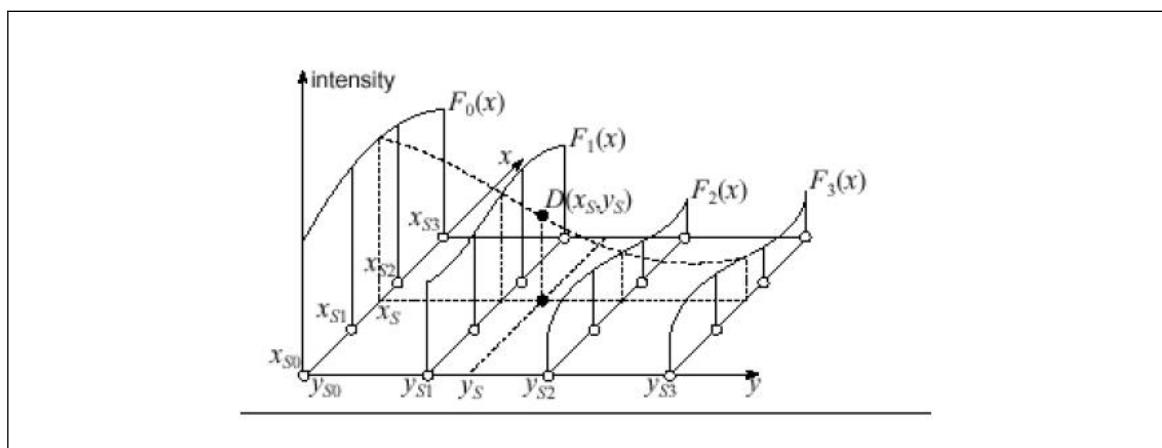
$$F_y(y_{S0}) = F_0(x_S), F_y(y_{S1}) = F_1(x_S), F_y(y_{S2}) = F_2(x_S), F_y(y_{S3}) = F_3(x_S).$$

The polynomial  $F_y(y)$  is represented by the dashed curve in Figure B-2.

Finally, the sought intensity  $D(x_D, y_D)$  is set to the value  $F_y(y_S)$ .

To use the cubic interpolation, set the *interpolation* parameter to `IPPI_INTER_CUBIC`.

### Cubic Interpolation



## Super Sampling

If the destination image is much smaller than the source image, the above interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing intensity of the destination pixels).

The super-sampling algorithm is as follows:

1. Divide the source image rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.
2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, the value of that pixel is taken with weight 1. If the rectangle and the square of the source pixel have an intersection of area  $a < 1$ , that pixel's value is taken with weight  $a$ .

Figure B-3 shows the corresponding weight value for each source pixel intersecting with the rectangle.

3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas  $S_{Src}/S_{Dst} = 1/xFactor*yFactor$ .

Here *xFactor*, and *yFactor* are the parameters of the functions that specify the factors by which the *x* and *y* dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for  $xFactor < 1$ , and  $yFactor < 1$ .

To use the supersampling interpolation, use the functions with the `Super` suffix (pass `interpolation=ippSuper` to `GetSize` functions).

## Supersampling Weights

$\Delta_2$	$\Delta_2$	$\Delta_2$	$\Delta_2 \times \Delta_3$
1	1	1	$\Delta_3$
1	1	1	$\Delta_3$
$\Delta_1$	$\Delta_1$	$\Delta_1$	$\Delta_1 \times \Delta_3$

## Lanczos Interpolation

This method is based on the 2-lobed or 3-lobed Lanczos window function as the interpolation function.

### Interpolation with the 2-lobed Lanczos Window Function

The interpolation algorithm uses source image intensities at 16 pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 1; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3;$$

$$y_{S0} = \text{int}(y_S) - 1; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3;$$

First, the intensity values are interpolated along the  $x$ -axis to produce four intermediate results  $I_0, I_1, I_2, I_3$ :

$$I_k = \sum_{i=0}^3 a_i \cdot s(x_{Si}, y_{Sk}), 0 \leq k \leq 3$$

Then the intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_k$  along the  $y$ -axis:

$$D(x_D, y_D) = \sum_{k=0}^3 b_k \cdot I_k$$

Here  $a_i$  and  $b_k$  are the coefficients defined as

$$a_i = L(x_S - x_{Si}), b_k = L(y_S - y_{Sk}),$$

where  $L(x)$  is the Lanczos windowed sinc function:

$$L(x) = \text{sinc}(x) \cdot \text{Lanczos2}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/2)}{(\pi x)/2}, & 0 \leq |x| < 2 \\ 0, & |x| \geq 2 \end{cases}$$

To use this interpolation, use the `ippiResizeLanczos` function.

## Interpolation with the 3-lobed Lanczos Window Function

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 2; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3; x_{S4} = x_{S0} + 4; x_{S5} = x_{S0} + 5;$$

$$y_{S0} = \text{int}(y_S) - 2; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3; y_{S4} = y_{S0} + 4; y_{S5} = y_{S0} + 5;$$

First, the intensity values are interpolated along the  $x$ -axis to produce six intermediate results  $I_0, I_1, \dots, I_5$ :

$$I_k = \sum_{i=0}^5 a_i \cdot s(x_{Si}, y_{Sk}), 0 \leq k \leq 5$$

Then the intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_k$  along the  $y$ -axis:

$$D(x_D, y_D) = \sum_{k=0}^5 b_k \cdot I_k$$

Here  $a_i$  and  $b_k$  are the coefficients defined as

$$a_i = L(x_S - x_{Si}), b_k = L(y_S - y_{Si}),$$

where  $L(x)$  is the Lanczos windowed sinc function:

$$L(x) = \text{sinc}(x) \cdot \text{Lanczos3}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/3)}{(\pi x)/3}, & 0 \leq |x| < 3 \\ 0, & 3 \leq |x| \end{cases}$$

To use this interpolation, set the *interpolation* parameter to `IPPI_INTER_LANZOS`, or use the functions with the `Lanczos` suffix (pass *interpolation=ippLanczos* to `GetSize` functions).

## Interpolation with Two-Parameter Cubic Filters

The two-parameter family of cubic filters have kernels of the form:

$$k(x) = \frac{1}{6} \begin{cases} (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B) & |x| < 1 \\ (-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) & 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$

where  $B$  and  $C$  are two parameters; their variations give different approximation.

These interpolation methods additionally filter the output to improve quality of an image.

To get more information about how  $B$  and  $C$  values affect the result, refer to [Mitchell88].

To use the interpolation with two-parameter cubic filters, use the functions with the `Cubic` suffix (pass *interpolation=ippCubic* to `GetSize` functions).

# Appendix C: Removed Functions for Image and Video Processing



This appendix contains tables that list the functions removed from Intel IPP 9.0. If an application created with the previous versions calls a function listed here, then the source code must be modified. The tables specify the corresponding Intel IPP 9.0 functions or workaround to replace the removed functions:

- [ippcc.h](#)
- [ippcv.h](#)
- [ippi.h](#)
- [ippj.h](#) - the whole domain is removed
- [ippvc.h](#) - the whole domain is removed

## NOTE

To get information on possible alternatives to the removed functions that do not have substitution or workaround in Intel IPP , refer to <https://software.intel.com/en-us/articles/the-alternatives-for-intel-ipp-legacy-domains-and-functions> or file a support request at [Online Service Center](#).

### [ippcc.h](#):

Removed from 9.0	Substitution or Workaround
ippiBGR555ToYCbCr411_16u8u_C3P3R	N/A
ippiBGR555ToYCbCr420_16u8u_C3P3R	N/A
ippiBGR555ToYCbCr422_16u8u_C3C2R	N/A
ippiBGR555ToYCbCr422_16u8u_C3P3R	N/A
ippiBGR555ToYCrCb420_16u8u_C3P3R	N/A
ippiBGR555ToYUV420_16u8u_C3P3R	N/A
ippiBGR565ToBGR_16u8u_C3R	N/A
ippiBGR565ToYCbCr411_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr420_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr422_16u8u_C3C2R	N/A
ippiBGR565ToYCbCr422_16u8u_C3P3R	N/A
ippiBGR565ToYCrCb420_16u8u_C3P3R	N/A
ippiBGR565ToYUV420_16u8u_C3P3R	N/A
ippiBGRToBGR565_8u16u_C3R	N/A
ippiColorTwist32f_8s_AC4IR	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_AC4R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_C3IR	8s is not supported anymore, use another data type

Removed from 9.0	Substitution or Workaround
ippiColorTwist32f_8s_C3R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_IP3R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_P3R	8s is not supported anymore, use another data type
ippiRGB565ToRGB_16u8u_C3R	N/A
ippiRGB565ToYUV420_16u8u_C3P3R	N/A
ippiRGB565ToYUV422_16u8u_C3P3R	N/A
ippiRGBCr411ToBGR555_8u16u_C3R	N/A
ippiYCbCr411ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr411ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR444_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR565Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB444_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB555_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB565_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR444Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR444_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR444_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR555Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR555_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR565Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR565Dither_8u16u_P3C3R	N/A

Removed from 9.0	Substitution or Workaround
ippiYCbCr422ToBGR565_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB444Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB444_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB444_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB555Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB555_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB555_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB565Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB565_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB565_8u16u_P3C3R	N/A
ippiYCbCrToBGR444Dither_8u16u_C3R	N/A
ippiYCbCrToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR444_8u16u_C3R	N/A
ippiYCbCrToBGR444_8u16u_P3C3R	N/A
ippiYCbCrToBGR555Dither_8u16u_C3R	N/A
ippiYCbCrToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR555_8u16u_C3R	N/A
ippiYCbCrToBGR555_8u16u_P3C3R	N/A
ippiYCbCrToBGR565Dither_8u16u_C3R	N/A
ippiYCbCrToBGR565Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR565_8u16u_C3R	N/A
ippiYCbCrToBGR565_8u16u_P3C3R	N/A
ippiYCbCrToRGB444Dither_8u16u_C3R	N/A
ippiYCbCrToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB444_8u16u_C3R	N/A
ippiYCbCrToRGB444_8u16u_P3C3R	N/A
ippiYCbCrToRGB555Dither_8u16u_C3R	N/A

Removed from 9.0	Substitution or Workaround
ippiYCbCrToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB555_8u16u_C3R	N/A
ippiYCbCrToRGB555_8u16u_P3C3R	N/A
ippiYCbCrToRGB565Dither_8u16u_C3R	N/A
ippiYCbCrToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB565_8u16u_C3R	N/A
ippiYCbCrToRGB565_8u16u_P3C3R	N/A
ippiYUV420ToBGR444Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR444_8u16u_P3C3R	N/A
ippiYUV420ToBGR555Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR555_8u16u_P3C3R	N/A
ippiYUV420ToBGR565Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR565_8u16u_P3C3R	N/A
ippiYUV420ToRGB444Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB444_8u16u_P3C3R	N/A
ippiYUV420ToRGB555Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB555_8u16u_P3C3R	N/A
ippiYUV420ToRGB565Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB565_8u16u_P3C3R	N/A

**ippcv.h:**

Removed from 9.0	Substitution or Workaround
ippiAddProduct_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddProduct_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAddSquare_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddSquare_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAddWeighted_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddWeighted_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAdd_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAdd_8s32f_C1IR	8s is not supported anymore, use another data type
ippiDilateBorderReplicate_32f_C1R	ippiDilateBorder_32f_C1R

Removed from 9.0	Substitution or Workaround
ippiDilateBorderReplicate_32f_C3R	ippiDilateBorder_32f_C3R
ippiDilateBorderReplicate_32f_C4R	ippiDilateBorder_32f_C4R
ippiDilateBorderReplicate_8u_C1R	ippiDilateBorder_8u_C1R
ippiDilateBorderReplicate_8u_C3R	ippiDilateBorder_8u_C3R
ippiDilateBorderReplicate_8u_C4R	ippiDilateBorder_8u_C4R
ippiErodeBorderReplicate_32f_C1R	ippiErodeBorder_32f_C1R
ippiErodeBorderReplicate_32f_C3R	ippiErodeBorder_32f_C3R
ippiErodeBorderReplicate_32f_C4R	ippiErodeBorder_32f_C4R
ippiErodeBorderReplicate_8u_C1R	ippiErodeBorder_8u_C1R
ippiErodeBorderReplicate_8u_C3R	ippiErodeBorder_8u_C3R
ippiErodeBorderReplicate_8u_C4R	ippiErodeBorder_8u_C4R
ippiFilterGaussBorder_32f_C1R	ippiFilterGaussianBorder_32f_C1R
ippiFilterGaussGetBufferSize_32f_C1R	ippiFilterGaussianGetSize
ippiFilterLaplacianBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterLaplacianGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterMaxBorderReplicate_32f_C1R	ippiFilterMaxBorder_32f_C1R
ippiFilterMaxBorderReplicate_32f_C3R	ippiFilterMaxBorder_32f_C3R
ippiFilterMaxBorderReplicate_32f_C4R	ippiFilterMaxBorder_32f_C4R
ippiFilterMaxBorderReplicate_8u_C1R	ippiFilterMaxBorder_8u_C1R
ippiFilterMaxBorderReplicate_8u_C3R	ippiFilterMaxBorder_8u_C3R
ippiFilterMaxBorderReplicate_8u_C4R	ippiFilterMaxBorder_8u_C4R
ippiFilterMaxGetBufferSize_32f_C1R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=1
ippiFilterMaxGetBufferSize_32f_C3R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=3
ippiFilterMaxGetBufferSize_32f_C4R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=4
ippiFilterMaxGetBufferSize_8u_C1R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=1
ippiFilterMaxGetBufferSize_8u_C3R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=3

Removed from 9.0	Substitution or Workaround
ippiFilterMaxGetBufferSize_8u_C4R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=4
ippiFilterMinBorderReplicate_32f_C1R	ippiFilterMinBorder_32f_C1R
ippiFilterMinBorderReplicate_32f_C3R	ippiFilterMinBorder_32f_C3R
ippiFilterMinBorderReplicate_32f_C4R	ippiFilterMinBorder_32f_C4R
ippiFilterMinBorderReplicate_8u_C1R	ippiFilterMinBorder_8u_C1R
ippiFilterMinBorderReplicate_8u_C3R	ippiFilterMinBorder_8u_C3R
ippiFilterMinBorderReplicate_8u_C4R	ippiFilterMinBorder_8u_C4R
ippiFilterMinGetBufferSize_32f_C1R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=1
ippiFilterMinGetBufferSize_32f_C3R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=3
ippiFilterMinGetBufferSize_32f_C4R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=4
ippiFilterMinGetBufferSize_8u_C1R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=1
ippiFilterMinGetBufferSize_8u_C3R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=3
ippiFilterMinGetBufferSize_8u_C4R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=4
ippiFilterScharrHorizBorder_32f_C1R	ippiFilterScharrHorizMaskBorder_32f_C1R from the ippIP domain
ippiFilterScharrHorizBorder_8u16s_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrHorizGetBufferSize_32f_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizGetBufferSize_8u16s_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrVertBorder_32f_C1R	ippiFilterScharrVertMaskBorder_32f_C1R from the ippIP domain
ippiFilterScharrVertBorder_8u16s_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain

Removed from 9.0	Substitution or Workaround
ippiFilterScharrVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrVertGetBufferSize_32f_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrVertGetBufferSize_8u16s_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelCrossBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelCrossGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizSecondBorder_32f_C1R	ippiFilterSobelHorizSecondBorder_32f_C1R from the ippIP domain
ippiFilterSobelHorizSecondBorder_8u16s_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelHorizSecondBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizSecondGetBufferSize_32f_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelHorizSecondGetBufferSize_8u16s_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelHorizSecondGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelNegVertBorder_32f_C1R	ippiFilterSobelNegVertBorder_32f_C1R from the ippIP domain
ippiFilterSobelNegVertBorder_8u16s_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain
ippiFilterSobelNegVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelNegVertGetBufferSize_32f_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain
ippiFilterSobelNegVertGetBufferSize_8u16s_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain

Removed from 9.0	Substitution or Workaround
ippiFilterSobelNegVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertSecondBorder_32f_C1R	ippiFilterSobelVertSecondBorder_32f_C1R from the ippIP domain
ippiFilterSobelVertSecondBorder_8u16s_C1R	ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelVertSecondBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertSecondGetBufferSize_32f_C1R	ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R	ippiFilterSobelVertSecondBorderGetBufferSize
ippiFilterSobelVertSecondGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiForegroundGaussianFree_8u_C1R	N/A
ippiForegroundGaussianFree_8u_C3R	N/A
ippiForegroundGaussianInitAlloc_8u_C1R	N/A
ippiForegroundGaussianInitAlloc_8u_C3R	N/A
ippiForegroundGaussian_8u_C1R	N/A
ippiForegroundGaussian_8u_C3R	N/A
ippiForegroundHistogramFree_8u_C1R	N/A
ippiForegroundHistogramFree_8u_C3R	N/A
ippiForegroundHistogramInitAlloc_8u_C1R	N/A
ippiForegroundHistogramInitAlloc_8u_C3R	N/A
ippiForegroundHistogramUpdate_8u_C1R	N/A
ippiForegroundHistogramUpdate_8u_C3R	N/A
ippiForegroundHistogram_8u_C1R	N/A
ippiForegroundHistogram_8u_C3R	N/A
ippiHaarClassifierFree_32f	ippFree
ippiHaarClassifierFree_32s	ippFree

Removed from 9.0	Substitution or Workaround
ippiHaarClassifierInitAlloc_32f	ippiHaarClassifierGetSize+ippMalloc +ippiHaarClassifierInit_32f
ippiHaarClassifierInitAlloc_32s	ippiHaarClassifierGetSize+ippMalloc +ippiHaarClassifierInit_32s
ippiInpaintFree_8u_C1R	ippFree
ippiInpaintFree_8u_C3R	ippFree
ippiInpaintInitAlloc_8u_C1R	ippiInpaintGetSize+ippMalloc +ippiInpaintInit_8u_C1R
ippiInpaintInitAlloc_8u_C3R	ippiInpaintGetSize+ippMalloc +ippiInpaintInit_8u_C3R
ippiMean_8s_C1MR	8s data type is not supported anymore - use another data type
ippiMean_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C1MR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C1R	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C3CR	8s data type is not supported anymore - use another data type
ippiMinMaxIdx_8s_C1MR	8s data type is not supported anymore - use another data type
ippiMinMaxIdx_8s_C1R	8s data type is not supported anymore - use another data type
ippiMinMaxIdx_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMinMaxIdx_8s_C3CR	8s data type is not supported anymore - use another data type
ippiMorphAdvFree	ippFree
ippiMorphAdvInitAlloc_32f_C1R	ippiMorphAdvGetSize_32f_C1R+ippMalloc
ippiMorphAdvInitAlloc_32f_C3R	ippiMorphAdvGetSize_32f_C3R+ippMalloc
ippiMorphAdvInitAlloc_32f_C4R	ippiMorphAdvGetSize_32f_C4R+ippMalloc

Removed from 9.0	Substitution or Workaround
ippiMorphAdvInitAlloc_8u_C1R	ippiMorphAdvGetSize_8u_C1R+ippMalloc +ippiMorphAdvInit_8u_C1R
ippiMorphAdvInitAlloc_8u_C3R	ippiMorphAdvGetSize_8u_C3R+ippMalloc +ippiMorphAdvInit_8u_C3R
ippiMorphAdvInitAlloc_8u_C4R	ippiMorphAdvGetSize_8u_C4R+ippMalloc +ippiMorphAdvInit_8u_C4R
ippiMorphGrayFree_32f_C1R	ippFree
ippiMorphGrayFree_8u_C1R	ippFree
ippiMorphGrayInitAlloc_32f_C1R	ippiMorphGrayGetSize_32f_C1R+ippMalloc +ippiMorphGrayInit_32f_C1R
ippiMorphGrayInitAlloc_8u_C1R	ippiMorphGrayGetSize_8u_C1R+ippMalloc +ippiMorphGrayInit_8u_C1R
ippiMorphReconstructGetBufferSize_16u_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_32f_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_64f_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_8u_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphologyFree	ippFree
ippiMorphologyGetSize_32f_C1R	ippiMorphologyBorderGetSize_32f_C1R
ippiMorphologyGetSize_32f_C3R	ippiMorphologyBorderGetSize_32f_C3R
ippiMorphologyGetSize_32f_C4R	ippiMorphologyBorderGetSize_32f_C4R
ippiMorphologyGetSize_8u_C1R	ippiMorphologyBorderGetSize_8u_C1R
ippiMorphologyGetSize_8u_C3R	ippiMorphologyBorderGetSize_8u_C3R
ippiMorphologyGetSize_8u_C4R	ippiMorphologyBorderGetSize_8u_C4R
ippiMorphologyInitAlloc_32f_C1R	ippiMorphologyBorderGetSize_32f_C1R +ippMalloc_8u +ippiMorphologyBorderInit_32f_C1R
ippiMorphologyInitAlloc_32f_C3R	ippiMorphologyBorderGetSize_32f_C3R +ippMalloc_8u +ippiMorphologyBorderInit_32f_C3R
ippiMorphologyInitAlloc_32f_C4R	ippiMorphologyBorderGetSize_32f_C4R +ippMalloc_8u +ippiMorphologyBorderInit_32f_C4R
ippiMorphologyInitAlloc_8u_C1R	ippiMorphologyBorderGetSize_8u_C1R +ippMalloc_8u +ippiMorphologyBorderInit_8u_C1R

Removed from 9.0	Substitution or Workaround
ippiMorphologyInitAlloc_8u_C3R	ippiMorphologyBorderGetSize_8u_C3R +ippMalloc_8u +ippiMorphologyBorderInit_8u_C3R
ippiMorphologyInitAlloc_8u_C4R	ippiMorphologyBorderGetSize_8u_C4R +ippMalloc_8u +ippiMorphologyBorderInit_8u_C4R
ippiMorphologyInit_32f_C1R	ippiMorphologyBorderInit_32f_C1R
ippiMorphologyInit_32f_C3R	ippiMorphologyBorderInit_32f_C3R
ippiMorphologyInit_32f_C4R	ippiMorphologyBorderInit_32f_C4R
ippiMorphologyInit_8u_C1R	ippiMorphologyBorderInit_8u_C1R
ippiMorphologyInit_8u_C3R	ippiMorphologyBorderInit_8u_C3R
ippiMorphologyInit_8u_C4R	ippiMorphologyBorderInit_8u_C4R
ippiNormDiff_Inf_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormDiff_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormDiff_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_L2_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_Inf_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormRel_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormRel_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormRel_L2_8s_C3CMR	8s data type is not supported anymore - use another data type

Removed from 9.0	Substitution or Workaround
ippiNorm_Inf_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNorm_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNorm_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_L2_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiOpticalFlowPyrLKFree_16u_C1R	ippFree
ippiOpticalFlowPyrLKFree_32f_C1R	ippFree
ippiOpticalFlowPyrLKFree_8u_C1R	ippFree
ippiOpticalFlowPyrLKInitAlloc_16u_C1R	ippiOpticalFlowPyrLKGetSize (with 16u dataType)+ippMalloc +ippiOpticalFlowPyrLKInit_16u_C1R
ippiOpticalFlowPyrLKInitAlloc_32f_C1R	ippiOpticalFlowPyrLKGetSize (with 32f dataType)+ippMalloc +ippiOpticalFlowPyrLKInit_32f_C1R
ippiOpticalFlowPyrLKInitAlloc_8u_C1R	ippiOpticalFlowPyrLKGetSize (with 8u dataType)+ippMalloc +ippiOpticalFlowPyrLKInit_8u_C1R
ippiPyrDownGetBufSize_Gauss5x5	ippiPyramidLayerDownGetSize_8u_C1R
ippiPyrDown_Gauss5x5_32f_C1R	ippiPyramidLayerDown_32f_C1R
ippiPyrDown_Gauss5x5_32f_C3R	ippiPyramidLayerDown_32f_C3R
ippiPyrDown_Gauss5x5_8s_C1R	8s data type is not supported anymore - use another data type
ippiPyrDown_Gauss5x5_8s_C3R	8s data type is not supported anymore - use another data type
ippiPyrDown_Gauss5x5_8u_C1R	ippiPyramidLayerDown_8u_C1R
ippiPyrDown_Gauss5x5_8u_C3R	ippiPyramidLayerDown_8u_C3R
ippiPyrUpGetBufSize_Gauss5x5	ippiPyramidLayerUpGetSize_8u_C1R
ippiPyrUp_Gauss5x5_32f_C1R	ippiPyramidLayerUp_32f_C1R
ippiPyrUp_Gauss5x5_32f_C3R	ippiPyramidLayerUp_32f_C3R

Removed from 9.0	Substitution or Workaround
ippiPyrUp_Gauss5x5_8s_C1R	8s data type is not supported anymore - use another data type
ippiPyrUp_Gauss5x5_8s_C3R	8s data type is not supported anymore - use another data type
ippiPyrUp_Gauss5x5_8u_C1R	ippiPyramidLayerUp_8u_C1R
ippiPyrUp_Gauss5x5_8u_C3R	ippiPyramidLayerUp_8u_C3R
ippiPyramidFree	ippFree
ippiPyramidInitAlloc	ippiPyramidGetSize+ippiPyramidInit
ippiPyramidLayerDownFree_16u_C1R	ippFree
ippiPyramidLayerDownFree_16u_C3R	ippFree
ippiPyramidLayerDownFree_32f_C1R	ippFree
ippiPyramidLayerDownFree_32f_C3R	ippFree
ippiPyramidLayerDownFree_8u_C1R	ippFree
ippiPyramidLayerDownFree_8u_C3R	ippFree
ippiPyramidLayerDownInitAlloc_16u_C1R	ippiPyramidLayerDownGetSize_16u_C1R +ippMalloc +ippiPyramidLayerDownInit_16u_C1R
ippiPyramidLayerDownInitAlloc_16u_C3R	ippiPyramidLayerDownGetSize_16u_C3R +ippMalloc +ippiPyramidLayerDownInit_16u_C3R
ippiPyramidLayerDownInitAlloc_32f_C1R	ippiPyramidLayerDownGetSize_32f_C1R +ippMalloc +ippiPyramidLayerDownInit_32f_C1R
ippiPyramidLayerDownInitAlloc_32f_C3R	ippiPyramidLayerDownGetSize_32f_C3R +ippMalloc +ippiPyramidLayerDownInit_32f_C3R
ippiPyramidLayerDownInitAlloc_8u_C1R	ippiPyramidLayerDownGetSize_8u_C1R +ippMalloc +ippiPyramidLayerDownInit_8u_C1R
ippiPyramidLayerDownInitAlloc_8u_C3R	ippiPyramidLayerDownGetSize_8u_C3R +ippMalloc +ippiPyramidLayerDownInit_8u_C3R
ippiPyramidLayerUpFree_16u_C1R	ippFree
ippiPyramidLayerUpFree_16u_C3R	ippFree
ippiPyramidLayerUpFree_32f_C1R	ippFree

Removed from 9.0	Substitution or Workaround
ippiPyramidLayerUpFree_32f_C3R	ippFree
ippiPyramidLayerUpFree_8u_C1R	ippFree
ippiPyramidLayerUpFree_8u_C3R	ippFree
ippiPyramidLayerUpInitAlloc_16u_C1R	ippiPyramidLayerUpGetSize_16u_C1R +ippMalloc +ippiPyramidLayerUpInit_16u_C1R
ippiPyramidLayerUpInitAlloc_16u_C3R	ippiPyramidLayerUpGetSize_16u_C3R +ippMalloc +ippiPyramidLayerUpInit_16u_C3R
ippiPyramidLayerUpInitAlloc_32f_C1R	ippiPyramidLayerUpGetSize_32f_C1R +ippMalloc +ippiPyramidLayerUpInit_32u_C1R
ippiPyramidLayerUpInitAlloc_32f_C3R	ippiPyramidLayerUpGetSize_32f_C3R +ippMalloc +ippiPyramidLayerUpInit_32f_C3R
ippiPyramidLayerUpInitAlloc_8u_C1R	ippiPyramidLayerUpGetSize_8u_C1R +ippMalloc+ippiPyramidLayerUpInit_8u_C1R
ippiPyramidLayerUpInitAlloc_8u_C3R	ippiPyramidLayerUpGetSize_8u_C3R +ippMalloc+ippiPyramidLayerUpInit_8u_C3R
ippiSRHNCalcResidual_PSF2x2_16u32f_C1R	N/A
ippiSRHNCalcResidual_PSF2x2_8u32f_C1R	N/A
ippiSRHNCalcResidual_PSF3x3_16u32f_C1R	N/A
ippiSRHNCalcResidual_PSF3x3_8u32f_C1R	N/A
ippiSRHNFree_PSF2x2	N/A
ippiSRHNFree_PSF3x3	N/A
ippiSRHNInitAlloc_PSF2x2	N/A
ippiSRHNInitAlloc_PSF3x3	N/A
ippiSRHNUpdateGradient_PSF2x2_32f_C1R	N/A
ippiSRHNUpdateGradient_PSF3x3_32f_C1R	N/A
ippiTiltedHaarClassifierInitAlloc_32f	ippiHaarClassifierGetSize+ippMalloc +ippiTiltedHaarClassifierInit_32f
ippiTiltedHaarClassifierInitAlloc_32s	ippiHaarClassifierGetSize+ippMalloc +ippiTiltedHaarClassifierInit_32s
ippiFastArctan_32f	ippsAtan_32f_A11

[ippi.h](#)

Removed from 9.0	Substitution or Workaround
ippiAddC_16sc_AC4IRSfs	N/A
ippiAddC_16sc_AC4RSfs	N/A
ippiAddC_16sc_C1IRSfs	N/A
ippiAddC_16sc_C1RSfs	N/A
ippiAddC_16sc_C3IRSfs	N/A
ippiAddC_16sc_C3RSfs	N/A
ippiAddC_32fc_AC4IR	N/A
ippiAddC_32fc_AC4R	N/A
ippiAddC_32fc_C1IR	N/A
ippiAddC_32fc_C1R	N/A
ippiAddC_32fc_C3IR	N/A
ippiAddC_32fc_C3R	N/A
ippiAddC_32sc_AC4IRSfs	N/A
ippiAddC_32sc_AC4RSfs	N/A
ippiAddC_32sc_C1IRSfs	N/A
ippiAddC_32sc_C1RSfs	N/A
ippiAddC_32sc_C3IRSfs	N/A
ippiAddC_32sc_C3RSfs	N/A
ippiAddRandGauss_Direct_16s_AC4IR	ippiAddRandGauss_16s_AC4IR
ippiAddRandGauss_Direct_16s_C1IR	ippiAddRandGauss_16s_C1IR
ippiAddRandGauss_Direct_16s_C3IR	ippiAddRandGauss_16s_C3IR
ippiAddRandGauss_Direct_16s_C4IR	ippiAddRandGauss_16s_C4IR
ippiAddRandGauss_Direct_16u_AC4IR	ippiAddRandGauss_16u_AC4IR
ippiAddRandGauss_Direct_16u_C1IR	ippiAddRandGauss_16u_C1IR
ippiAddRandGauss_Direct_16u_C3IR	ippiAddRandGauss_16u_C3IR
ippiAddRandGauss_Direct_16u_C4IR	ippiAddRandGauss_16u_C4IR
ippiAddRandGauss_Direct_32f_AC4IR	ippiAddRandGauss_32f_AC4IR
ippiAddRandGauss_Direct_32f_C1IR	ippiAddRandGauss_32f_C1IR
ippiAddRandGauss_Direct_32f_C3IR	ippiAddRandGauss_32f_C3IR
ippiAddRandGauss_Direct_32f_C4IR	ippiAddRandGauss_32f_C4IR

Removed from 9.0	Substitution or Workaround
ippiAddRandGauss_Direct_8u_AC4IR	ippiAddRandGauss_8u_AC4IR
ippiAddRandGauss_Direct_8u_C1IR	ippiAddRandGauss_8u_C1IR
ippiAddRandGauss_Direct_8u_C3IR	ippiAddRandGauss_8u_C3IR
ippiAddRandGauss_Direct_8u_C4IR	ippiAddRandGauss_8u_C4IR
ippiAddRandUniform_Direct_16s_AC4IR	ippiAddRandUniform_16s_AC4IR
ippiAddRandUniform_Direct_16s_C1IR	ippiAddRandUniform_16s_C1IR
ippiAddRandUniform_Direct_16s_C3IR	ippiAddRandUniform_16s_C3IR
ippiAddRandUniform_Direct_16s_C4IR	ippiAddRandUniform_16s_C4IR
ippiAddRandUniform_Direct_16u_AC4IR	ippiAddRandUniform_16u_AC4IR
ippiAddRandUniform_Direct_16u_C1IR	ippiAddRandUniform_16u_C1IR
ippiAddRandUniform_Direct_16u_C3IR	ippiAddRandUniform_16u_C3IR
ippiAddRandUniform_Direct_16u_C4IR	ippiAddRandUniform_16u_C4IR
ippiAddRandUniform_Direct_32f_AC4IR	ippiAddRandUniform_32f_AC4IR
ippiAddRandUniform_Direct_32f_C1IR	ippiAddRandUniform_32f_C1IR
ippiAddRandUniform_Direct_32f_C3IR	ippiAddRandUniform_32f_C3IR
ippiAddRandUniform_Direct_32f_C4IR	ippiAddRandUniform_32f_C4IR
ippiAddRandUniform_Direct_8u_AC4IR	ippiAddRandUniform_8u_AC4IR
ippiAddRandUniform_Direct_8u_C1IR	ippiAddRandUniform_8u_C1IR
ippiAddRandUniform_Direct_8u_C3IR	ippiAddRandUniform_8u_C3IR
ippiAddRandUniform_Direct_8u_C4IR	ippiAddRandUniform_8u_C4IR
ippiAddRotateShift	N/A
ippiAdd_16sc_AC4IRSfs	N/A
ippiAdd_16sc_AC4RSfs	N/A
ippiAdd_16sc_C1IRSfs	N/A
ippiAdd_16sc_C1RSfs	N/A
ippiAdd_16sc_C3IRSfs	N/A
ippiAdd_16sc_C3RSfs	N/A
ippiAdd_32fc_AC4IR	N/A
ippiAdd_32fc_AC4R	N/A
ippiAdd_32fc_C1IR	N/A

Removed from 9.0	Substitution or Workaround
ippiAdd_32fc_C1R	N/A
ippiAdd_32fc_C3IR	N/A
ippiAdd_32fc_C3R	N/A
ippiAdd_32sc_AC4IRSfs	N/A
ippiAdd_32sc_AC4RSfs	N/A
ippiAdd_32sc_C1IRSfs	N/A
ippiAdd_32sc_C1RSfs	N/A
ippiAdd_32sc_C3IRSfs	N/A
ippiAdd_32sc_C3RSfs	N/A
ippiAlphaCompC_8s_C1R	ippiConvert_8s16s+ippiAlphaCompC_16s
ippiAlphaComp_8s_AC1R	ippiConvert_8s16s+ippiAlphaComp_16s
ippiAlphaComp_8s_AC4R	ippiConvert_8s16s+ippiAlphaComp_16s
ippiComplement_32s_C1IR	N/A
ippiConvFull_16s_AC4R	ippiConv_16s_C4R (with algType=ippiROIFull)
ippiConvFull_16s_C1R	ippiConv_16s_C1R (with algType=ippiROIFull)
ippiConvFull_16s_C3R	ippiConv_16s_C3R (with algType=ippiROIFull)
ippiConvFull_32f_AC4R	ippiConv_32f_C4R (with algType=ippiROIFull)
ippiConvFull_32f_C1R	ippiConv_32f_C1R (with algType=ippiROIFull)
ippiConvFull_32f_C3R	ippiConv_32f_C3R (with algType=ippiROIFull)
ippiConvFull_8u_AC4R	ippiConv_8u_C4R (with algType=ippiROIFull)
ippiConvFull_8u_C1R	ippiConv_8u_C1R (with algType=ippiROIFull)
ippiConvFull_8u_C3R	ippiConv_8u_C3R (with algType=ippiROIFull)
ippiConvValid_16s_AC4R	ippiConv16s_C4R (with algType=ippiROIValid)
ippiConvValid_16s_C1R	ippiConv16s_C1R (with algType=ippiROIValid)
ippiConvValid_16s_C3R	ippiConv16s_C3R (with algType=ippiROIValid)
ippiConvValid_32f_AC4R	ippiConv32f_C4R (with algType=ippiROIValid)
ippiConvValid_32f_C1R	ippiConv32f_C1R (with algType=ippiROIValid)
ippiConvValid_32f_C3R	ippiConv32f_C3R (with algType=ippiROIValid)
ippiConvValid_8u_AC4R	ippiConv8u_C4R (with algType=ippiROIValid)
ippiConvValid_8u_C1R	ippiConv8u_C1R (with algType=ippiROIValid)

Removed from 9.0	Substitution or Workaround
ippiConvValid_8u_C3R	ippiConv8u_C3R (with <code>algType=ippiROIValid</code> )
ippiConvert_1u8u_C1R	ippiBinToGray_1u8u_C1R
ippiCplxExtendToPack_16sc16s_C1R	ippiConvert_16s32f +ippiCplxExtendToPack_32f
ippiCplxExtendToPack_16sc16s_C3R	ippiConvert_16s32f +ippiCplxExtendToPack_32f
ippiCplxExtendToPack_32sc32s_C1R	ippiConvert_32s32f +ippiCplxExtendToPack_32f
ippiCplxExtendToPack_32sc32s_C3R	ippiConvert_32s32f +ippiCplxExtendToPack_32f
ippiCrossCorrFull_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>
ippiCrossCorrFull_NormLevel_64f_C1R	N/A
ippiCrossCorrFull_NormLevel_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R + <code>algType=ippiROIFull ippiNormCoefficient</code>

Removed from 9.0	Substitution or Workaround
ippiCrossCorrFull_NormLevel_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNormCoefficient
ippiCrossCorrFull_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrFull_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrFull_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiCrossCorrFull_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiCrossCorrSame_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient

Removed from 9.0	Substitution or Workaround
ippiCrossCorrSame_NormLevel_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNormCoefficient
ippiCrossCorrSame_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrSame_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNorm
ippiCrossCorrSame_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISame ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrSame_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNormNone
ippiCrossCorrValid_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormNone
ippiCrossCorrValid_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormNone
ippiCrossCorrValid_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNormNone
ippiCrossCorrValid_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_64f_C1R	N/A
ippiCrossCorrValid_NormLevel_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient

Removed from 9.0	Substitution or Workaround
ippiCrossCorrValid_NormLevel_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNormCoefficient
ippiCrossCorrValid_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrValid_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8s32f_C1R	ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm

Removed from 9.0	Substitution or Workaround
ippiCrossCorrValid_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiCrossCorrValid_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiDCTFwdFree_32f	ippiFree
ippiDCTFwdGetBufSize_32f	N/A
ippiDCTFwdInitAlloc_32f	ippiDCTGetSize+ippiMalloc+ippiDCTInit
ippiDCTInvFree_32f	ippiFree
ippiDCTInvGetBufSize_32f	N/A
ippiDCTInvInitAlloc_32f	ippiDCTGetSize+ippiMalloc+ippiDCTInit
ippiDFTFree_C_32fc	ippiFree
ippiDFTFree_R_32f	ippiFree
ippiDFTFree_R_32s	ippiFree
ippiDFTFwd_RToPack_8u32s_AC4RSfs	ippiConvert_8u32f+ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C1RSfs	ippiConvert_8u32f+ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C3RSfs	ippiConvert_8u32f+ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C4RSfs	ippiConvert_8u32f+ippiDFTFwd_RToPack_32f
ippiDFTGetBufSize_C_32fc	N/A
ippiDFTGetBufSize_R_32f	N/A
ippiDFTGetBufSize_R_32s	N/A
ippiDFTInitAlloc_C_32fc	ippiDFTGetSize+ippiMalloc+ippiDFTInit
ippiDFTInitAlloc_R_32f	ippiDFTGetSize+ippiMalloc+ippiDFTInit
ippiDFTInitAlloc_R_32s	ippiConvert_32s32f(or to 64f) +ippiDFTGetSize+ippiMalloc+ippiDFTInit
ippiDFTInv_PackToR_32s8u_AC4RSfs	ippiDFTFwd_PackToR_32f+ippiConvert_32f8u
ippiDFTInv_PackToR_32s8u_C1RSfs	ippiDFTFwd_PackToR_32f+ippiConvert_32f8u
ippiDFTInv_PackToR_32s8u_C3RSfs	ippiDFTFwd_PackToR_32f+ippiConvert_32f8u
ippiDFTInv_PackToR_32s8u_C4RSfs	ippiDFTFwd_PackToR_32f+ippiConvert_32f8u

Removed from 9.0	Substitution or Workaround
ippiDeconvFFTFree_32f_C1R	ippFree
ippiDeconvFFTFree_32f_C3R	ippFree
ippiDeconvFFTInitAlloc_32f_C1R	ippiDeconvFFTGetSize_32f+ippMalloc_8u +ippiDeconvFFTInit_32f_C1R
ippiDeconvFFTInitAlloc_32f_C3R	ippiDeconvFFTGetSize_32f+ippMalloc_8u +ippiDeconvFFTInit_32f_C3R
ippiDeconvLRFree_32f_C1R	ippFree
ippiDeconvLRFree_32f_C3R	ippFree
ippiDeconvLRInitAlloc_32f_C1R	ippiDeconvLRGetSize_32f+ippMalloc_8u +ippiDeconvLRInit_32f_C1R
ippiDeconvLRInitAlloc_32f_C3R	ippiDeconvLRGetSize_32f+ippMalloc_8u +ippiDeconvLRInit_32f_C3R
ippiDilate3x3_16u_AC4IR	Use not-in-place flavor
ippiDilate3x3_16u_AC4R	ippiDilateBorder_16u_C4R with 3x3 mask
ippiDilate3x3_16u_C1IR	Use not-in-place flavor
ippiDilate3x3_16u_C1R	ippiDilateBorder_16u_C1R with 3x3 mask
ippiDilate3x3_16u_C3IR	Use not-in-place flavor
ippiDilate3x3_16u_C3R	ippiDilateBorder_16u_C3R with 3x3 mask
ippiDilate3x3_16u_C4IR	Use not-in-place flavor
ippiDilate3x3_16u_C4R	ippiDilateBorder_16u_C4R with 3x3 mask
ippiDilate3x3_32f_AC4IR	Use not-in-place flavor
ippiDilate3x3_32f_AC4R	ippiDilateBorder_32f_C4R with 3x3 mask
ippiDilate3x3_32f_C1IR	Use not-in-place flavor
ippiDilate3x3_32f_C1R	ippiDilateBorder_32f_C1R with 3x3 mask
ippiDilate3x3_32f_C3IR	Use not-in-place flavor
ippiDilate3x3_32f_C3R	ippiDilateBorder_32f_C3R with 3x3 mask
ippiDilate3x3_32f_C4IR	Use not-in-place flavor
ippiDilate3x3_32f_C4R	ippiDilateBorder_32f_C4R with 3x3 mask
ippiDilate3x3_8u_AC4IR	Use not-in-place flavor
ippiDilate3x3_8u_AC4R	ippiDilateBorder_8u_C4R with 3x3 mask
ippiDilate3x3_8u_C1IR	Use not-in-place flavor
ippiDilate3x3_8u_C1R	ippiDilateBorder_8u_C1R with 3x3 mask

Removed from 9.0	Substitution or Workaround
ippiDilate3x3_8u_C3IR	Use not-in-place flavor
ippiDilate3x3_8u_C3R	ippiDilateBorder_8u_C3R with 3x3 mask
ippiDilate3x3_8u_C4IR	Use not-in-place flavor
ippiDilate3x3_8u_C4R	ippiDilateBorder_8u_C4R with 3x3 mask
ippiDilate_16u_AC4IR	Use not-in-place flavor
ippiDilate_16u_AC4R	ippiDilateBorder_16u_C4R
ippiDilate_16u_C1IR	Use not-in-place flavor
ippiDilate_16u_C1R	ippiDilateBorder_16u_C1R
ippiDilate_16u_C3IR	Use not-in-place flavor
ippiDilate_16u_C3R	ippiDilateBorder_16u_C3R
ippiDilate_16u_C4R	ippiDilateBorder_16u_C4R
ippiDilate_32f_AC4IR	Use not-in-place flavor
ippiDilate_32f_AC4R	ippiDilateBorder_32f_C4R
ippiDilate_32f_C1IR	Use not-in-place flavor
ippiDilate_32f_C1R	ippiDilateBorder_32f_C1R
ippiDilate_32f_C3IR	Use not-in-place flavor
ippiDilate_32f_C3R	ippiDilateBorder_32f_C3R
ippiDilate_32f_C4R	ippiDilateBorder_32f_C4R
ippiDilate_8u_AC4IR	Use not-in-place flavor
ippiDilate_8u_AC4R	ippiDilateBorder_8u_C4R
ippiDilate_8u_C1IR	Use not-in-place flavor
ippiDilate_8u_C1R	ippiDilateBorder_8u_C1R
ippiDilate_8u_C3IR	Use not-in-place flavor
ippiDilate_8u_C3R	ippiDilateBorder_8u_C3R
ippiDilate_8u_C4R	ippiDilateBorder_8u_C4R
ippiDivC_16sc_AC4IRSfs	N/A
ippiDivC_16sc_AC4RSfs	N/A
ippiDivC_16sc_C1IRSfs	N/A
ippiDivC_16sc_C1RSfs	N/A
ippiDivC_16sc_C3IRSfs	N/A

Removed from 9.0	Substitution or Workaround
ippiDivC_16sc_C3RSfs	N/A
ippiDivC_32fc_AC4IR	N/A
ippiDivC_32fc_AC4R	N/A
ippiDivC_32fc_C1IR	N/A
ippiDivC_32fc_C1R	N/A
ippiDivC_32fc_C3IR	N/A
ippiDivC_32fc_C3R	N/A
ippiDivC_32sc_AC4IRSfs	N/A
ippiDivC_32sc_AC4RSfs	N/A
ippiDivC_32sc_C1IRSfs	N/A
ippiDivC_32sc_C1RSfs	N/A
ippiDivC_32sc_C3IRSfs	N/A
ippiDivC_32sc_C3RSfs	N/A
ippiDiv_16sc_AC4IRSfs	N/A
ippiDiv_16sc_AC4RSfs	N/A
ippiDiv_16sc_C1IRSfs	N/A
ippiDiv_16sc_C1RSfs	N/A
ippiDiv_16sc_C3IRSfs	N/A
ippiDiv_16sc_C3RSfs	N/A
ippiDiv_32fc_AC4IR	N/A
ippiDiv_32fc_AC4R	N/A
ippiDiv_32fc_C1IR	N/A
ippiDiv_32fc_C1R	N/A
ippiDiv_32fc_C3IR	N/A
ippiDiv_32fc_C3R	N/A
ippiDiv_32sc_AC4IRSfs	N/A
ippiDiv_32sc_AC4RSfs	N/A
ippiDiv_32sc_C1IRSfs	N/A
ippiDiv_32sc_C1RSfs	N/A
ippiDiv_32sc_C3IRSfs	N/A
ippiDiv_32sc_C3RSfs	N/A

Removed from 9.0	Substitution or Workaround
ippiDotProd_8s64f_AC4R	ippiConvert_8s16s+ippiDotProd_16s64f
ippiDotProd_8s64f_C1R	ippiConvert_8s16s+ippiDotProd_16s64f
ippiDotProd_8s64f_C3R	ippiConvert_8s16s+ippiDotProd_16s64f
ippiDotProd_8s64f_C4R	ippiConvert_8s16s+ippiDotProd_16s64f
ippiErode3x3_16u_AC4IR	Use not-in-place flavor
ippiErode3x3_16u_AC4R	ippiErodeBorder_16u_C4R with 3x3 mask
ippiErode3x3_16u_C1IR	Use not-in-place flavor
ippiErode3x3_16u_C1R	ippiErodeBorder_16u_C1R with 3x3 mask
ippiErode3x3_16u_C3IR	Use not-in-place flavor
ippiErode3x3_16u_C3R	ippiErodeBorder_16u_C3R with 3x3 mask
ippiErode3x3_16u_C4IR	Use not-in-place flavor
ippiErode3x3_16u_C4R	ippiErodeBorder_16u_C4R with 3x3 mask
ippiErode3x3_32f_AC4IR	Use not-in-place flavor
ippiErode3x3_32f_AC4R	ippiErodeBorder_32f_C4R with 3x3 mask
ippiErode3x3_32f_C1IR	Use not-in-place flavor
ippiErode3x3_32f_C1R	ippiErodeBorder_32f_C1R with 3x3 mask
ippiErode3x3_32f_C3IR	Use not-in-place flavor
ippiErode3x3_32f_C3R	ippiErodeBorder_32f_C3R with 3x3 mask
ippiErode3x3_32f_C4IR	Use not-in-place flavor
ippiErode3x3_32f_C4R	ippiErodeBorder_32f_C4R with 3x3 mask
ippiErode3x3_8u_AC4IR	Use not-in-place flavor
ippiErode3x3_8u_AC4R	ippiErodeBorder_8u_C4R with 3x3 mask
ippiErode3x3_8u_C1IR	Use not-in-place flavor
ippiErode3x3_8u_C1R	ippiErodeBorder_8u_C1R with 3x3 mask
ippiErode3x3_8u_C3IR	Use not-in-place flavor
ippiErode3x3_8u_C3R	ippiErodeBorder_8u_C3R with 3x3 mask
ippiErode3x3_8u_C4IR	Use not-in-place flavor
ippiErode3x3_8u_C4R	ippiErodeBorder_8u_C4R with 3x3 mask
ippiErode_16u_AC4IR	Use not-in-place flavor
ippiErode_16u_AC4R	ippiErodeBorder_16u_C4R

Removed from 9.0	Substitution or Workaround
ippiErode_16u_C1IR	Use not-in-place flavor
ippiErode_16u_C1R	ippiErodeBorder_16u_C1R
ippiErode_16u_C3IR	Use not-in-place flavor
ippiErode_16u_C3R	ippiErodeBorder_16u_C3R
ippiErode_16u_C4R	ippiErodeBorder_16u_C4R
ippiErode_32f_AC4IR	Use not-in-place flavor
ippiErode_32f_AC4R	ippiErodeBorder_32f_C4R
ippiErode_32f_C1IR	Use not-in-place flavor
ippiErode_32f_C1R	ippiErodeBorder_32f_C1R
ippiErode_32f_C3IR	Use not-in-place flavor
ippiErode_32f_C3R	ippiErodeBorder_32f_C3R
ippiErode_32f_C4R	ippiErodeBorder_32f_C4R
ippiErode_8u_AC4IR	Use not-in-place flavor
ippiErode_8u_AC4R	ippiErodeBorder_8u_C4R
ippiErode_8u_C1IR	Use not-in-place flavor
ippiErode_8u_C1R	ippiErodeBorder_8u_C1R
ippiErode_8u_C3IR	Use not-in-place flavor
ippiErode_8u_C3R	ippiErodeBorder_8u_C3R
ippiErode_8u_C4R	ippiErodeBorder_8u_C4R
ippiIFFTFree_C_32fc	ippiFree
ippiIFFTFree_R_32f	ippiFree
ippiIFFTFree_R_32s	ippiFree
ippiIFFTFwd_RToPack_8u32s_AC4RSfs	ippiConvert_8u32f+ippiIFFTFwd_RToPack_32f
ippiIFFTFwd_RToPack_8u32s_C1RSfs	ippiConvert_8u32f+ippiIFFTFwd_RToPack_32f
ippiIFFTFwd_RToPack_8u32s_C3RSfs	ippiConvert_8u32f+ippiIFFTFwd_RToPack_32f
ippiIFFTFwd_RToPack_8u32s_C4RSfs	ippiConvert_8u32f+ippiIFFTFwd_RToPack_32f
ippiIFFTGetSize_C_32fc	N/A
ippiIFFTGetSize_R_32f	N/A
ippiIFFTGetSize_R_32s	N/A
ippiIFTInitAlloc_C_32fc	ippiFFTGetSize+ippiMalloc+ippiIFTInit

Removed from 9.0	Substitution or Workaround
ippiFFTInitAlloc_R_32f	ippiFFTGetSize+ippiMalloc+ippiFFTInit
ippiFFTInitAlloc_R_32s	ippiConvert_32s32f(or to 64f) +ippiFFTGetSize+ippiMalloc+ippiFFTInit
ippiFFTInv_PackToR_32s8u_AC4RSfs	ippiFFTFwd_PackToR_32f+ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C1RSfs	ippiFFTFwd_PackToR_32f+ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C3RSfs	ippiFFTFwd_PackToR_32f+ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C4RSfs	ippiFFTFwd_PackToR_32f+ippiConvert_32f8u
ippiFilter32f_16s_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R
<hr/> ippiFilter32f_16s_C1R	<hr/> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p> <hr/>
ippiFilter32f_16s_C3R	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p> <hr/>
ippiFilter32f_16s_C4R	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p> <hr/>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_16u_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_16u_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C1R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_16u_C3R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_16u_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C4R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_32s_C1R	ippiFilterBorderInit_32f +ippiConvert_32s32f +ippiFilterBorder_32f_C1R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_32s_C3R	ippiFilterBorderInit_32f +ippiConvert_32s32f +ippiFilterBorder_32f_C3R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_32s_C4R	ippiFilterBorderInit_32f +ippiConvert_32s32f +ippiFilterBorder_32f_C4R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s16s_C1R	ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C1R

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s16s_C3R	<pre>ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C3R</pre>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s16s_C4R	<pre>ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C4R</pre>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s_C1R	<pre>ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C1R</pre>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s_C3R	<pre>ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C3R</pre>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8s_C4R	ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C4R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8u16s_C1R	ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C1R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8u16s_C3R	ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C3R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter32f_8u16s_C4R	ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C4R

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_8u_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_8u_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C1R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_8u_C3R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter32f_8u_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code>

Removed from 9.0	Substitution or Workaround
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
<code>ippiFilterBilateralGetSize_8u_C1R</code>	<code>ippiFilterBilateralBorderGetBufferSize</code>
<code>ippiFilterBilateralInit_8u_C1R</code>	<code>ippiFilterBilateralBorderInit</code>
<code>ippiFilterBilateral_8u_C1R</code>	<code>ippiFilterBilateralBorder_8u_C1R</code>
<code>ippiFilterBox_16s_AC4IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_AC4R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C1IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C1R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C3IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C3R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C4IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16s_C4R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_AC4IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_AC4R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C1IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C1R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C3IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C3R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C4IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_16u_C4R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_32f_AC4IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_32f_AC4R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_32f_C1IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_32f_C1R</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>
<code>ippiFilterBox_32f_C3IR</code>	<b>Use</b> <code>ippiFilterBoxBorder</code>

Removed from 9.0	Substitution or Workaround
ippiFilterBox_32f_C3R	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_32f_C4IR	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_32f_C4R	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_AC4IR	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_AC4R	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C1IR	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C1R	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C3IR	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C3R	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C4IR	Use <code>ippiFilterBoxBorder</code>
ippiFilterBox_8u_C4R	Use <code>ippiFilterBoxBorder</code>
ippiFilterColumn_64f_C1R	Use <code>ippiFilter_64f_C1R</code>
ippiFilterColumn32f_16s_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R (with filter.width=1)
ippiFilterColumn32f_16s_C1R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C1R (with filter.width=1)
ippiFilterColumn32f_16s_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C3R (with filter.width=1)

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_16s_C4R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_16u_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_16u_C4R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_16u_C1R	ippiFilterBorderInit_32f +ippiFilterBorder_16u_C1R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_16u_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_16u_C3R (with filter.width=1)

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_16u_C4R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_16u_C4R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_8u_AC4R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_8u_C4R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_8u_C1R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_8u_C1R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn32f_8u_C3R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_8u_C3R (with <code>filter.width=1</code>)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterColumn32f_8u_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code> (with <code>filter.width=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterColumn_16s_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C4R</code> (with <code>filter.width=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterColumn_16s_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C1R</code> (with <code>filter.width=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterColumn_16s_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C3R</code> (with <code>filter.width=1</code> )

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_16s_C4R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R (with filter.width=1)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_16u_AC4R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16u_C4R (with filter.width=1)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_16u_C1R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16u_C1R (with filter.width=1)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_16u_C3R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16u_C3R (with filter.width=1)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_16u_C4R	ippiFilterBorderInit_16s +ippiFilterBorder_16u_C4R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_32f_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_32f_C1R	ippiFilterBorderInit_32f +ippiFilterBorder_32f_C1R (with filter.width=1)
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_32f_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_32f_C3R (with filter.width=1)

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_32f_C4R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_8u_AC4R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_8u_C4R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_8u_C1R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_8u_C1R (with <code>filter.width=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterColumn_8u_C3R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_8u_C3R (with <code>filter.width=1</code>)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterColumn_8u_C4R</code>	<code>ippiFilterBorderInit_16s</code> <code>+ippiFilterBorder_8u_C4R (with filter.width=1)</code> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterGauss_16s_AC4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_16s_C1R</code>
<code>ippiFilterGauss_16s_C1R</code>	<code>ippiFilterGaussianBorder_16s_C1R</code>
<code>ippiFilterGauss_16s_C3R</code>	<code>ippiFilterGaussianBorder_16s_C3R</code>
<code>ippiFilterGauss_16s_C4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_16s_C1R</code>
<code>ippiFilterGauss_16u_AC4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_16u_C1R</code>
<code>ippiFilterGauss_16u_C1R</code>	<code>ippiFilterGaussianBorder_16u_C1R</code>
<code>ippiFilterGauss_16u_C3R</code>	<code>ippiFilterGaussianBorder_16u_C3R</code>
<code>ippiFilterGauss_16u_C4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_16u_C1R</code>
<code>ippiFilterGauss_32f_AC4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_32f_C1R</code>
<code>ippiFilterGauss_32f_C1R</code>	<code>ippiFilterGaussianBorder_32f_C1R</code>
<code>ippiFilterGauss_32f_C3R</code>	<code>ippiFilterGaussianBorder_32f_C3R</code>
<code>ippiFilterGauss_32f_C4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_32f_C1R</code>
<code>ippiFilterGauss_8u_AC4R</code>	<code>ippiCopy_C4C1R</code> <code>+ippiFilterGaussianBorder_8u_C1R</code>
<code>ippiFilterGauss_8u_C1R</code>	<code>ippiFilterGaussianBorder_8u_C1R</code>

Removed from 9.0	Substitution or Workaround
ippiFilterGauss_8u_C3R	ippiFilterGaussianBorder_8u_C3R
ippiFilterGauss_8u_C4R	ippiCopy_C4C1R +ippiFilterGaussianBorder_8u_C1R
ippiFilterHipass_16s_AC4R	ippiFilterHipassBorder_16s_AC4R
ippiFilterHipass_16s_C1R	ippiFilterHipassBorder_16s_C1R
ippiFilterHipass_16s_C3R	ippiFilterHipassBorder_16s_C3R
ippiFilterHipass_16s_C4R	ippiFilterHipassBorder_16s_C4R
ippiFilterHipass_32f_AC4R	ippiFilterHipassBorder_32f_AC4R
ippiFilterHipass_32f_C1R	ippiFilterHipassBorder_32f_C1R
ippiFilterHipass_32f_C3R	ippiFilterHipassBorder_32f_C3R
ippiFilterHipass_32f_C4R	ippiFilterHipassBorder_32f_C4R
ippiFilterHipass_8u_AC4R	ippiFilterHipassBorder_8u_AC4R
ippiFilterHipass_8u_C1R	ippiFilterHipassBorder_8u_C1R
ippiFilterHipass_8u_C3R	ippiFilterHipassBorder_8u_C3R
ippiFilterHipass_8u_C4R	ippiFilterHipassBorder_8u_C4R
ippiFilterLaplace_16s_AC4R	ippiFilterLaplaceBorder_16s_AC4R
ippiFilterLaplace_16s_C1R	ippiFilterLaplaceBorder_16s_C1R
ippiFilterLaplace_16s_C3R	ippiFilterLaplaceBorder_16s_C3R
ippiFilterLaplace_16s_C4R	ippiFilterLaplaceBorder_16s_C4R
ippiFilterLaplace_32f_AC4R	ippiFilterLaplaceBorder_32f_AC4R
ippiFilterLaplace_32f_C1R	ippiFilterLaplaceBorder_32f_C1R
ippiFilterLaplace_32f_C3R	ippiFilterLaplaceBorder_32f_C3R
ippiFilterLaplace_32f_C4R	ippiFilterLaplaceBorder_32f_C4R
ippiFilterLaplace_8s16s_C1R	ippiConvert_8s16s +ippiFilterLaplaceBorder_16s
ippiFilterLaplace_8u16s_C1R	ippiConvert_8u16s +ippiFilterLaplaceBorder_16s
ippiFilterLaplace_8u_AC4R	ippiFilterLaplaceBorder_8u_AC4R
ippiFilterLaplace_8u_C1R	ippiFilterLaplaceBorder_8u_C1R
ippiFilterLaplace_8u_C3R	ippiFilterLaplaceBorder_8u_C3R
ippiFilterLaplace_8u_C4R	ippiFilterLaplaceBorder_8u_C4R

Removed from 9.0	Substitution or Workaround
ippiFilterLowpass_16s_AC4R	ippiFilterBoxBorder_16s_AC4R
ippiFilterLowpass_16s_C1R	ippiFilterBoxBorder_16s_C1R
ippiFilterLowpass_16s_C3R	ippiFilterBoxBorder_16s_C3R
ippiFilterLowpass_16u_AC4R	ippiFilterBoxBorder_16u_AC4R
ippiFilterLowpass_16u_C1R	ippiFilterBoxBorder_16u_C1R
ippiFilterLowpass_16u_C3R	ippiFilterBoxBorder_16u_C3R
ippiFilterLowpass_32f_AC4R	ippiFilterBoxBorder_32f_AC4R
ippiFilterLowpass_32f_C1R	ippiFilterBoxBorder_32f_C1R
ippiFilterLowpass_32f_C3R	ippiFilterBoxBorder_32f_C3R
ippiFilterLowpass_8u_AC4R	ippiFilterBoxBorder_8u_AC4R
ippiFilterLowpass_8u_C1R	ippiFilterBoxBorder_8u_C1R
ippiFilterLowpass_8u_C3R	ippiFilterBoxBorder_8u_C3R
ippiFilterMax_16s_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C1R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C3R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C4R	Use ippiFilterMaxBorder
ippiFilterMax_16u_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C1R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C3R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C4R	Use ippiFilterMaxBorder
ippiFilterMax_32f_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_32f_C1R	Use ippiFilterMaxBorder
ippiFilterMax_32f_C3R	Use ippiFilterMaxBorder
ippiFilterMax_32f_C4R	Use ippiFilterMaxBorder
ippiFilterMax_8u_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C1R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C3R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C4R	Use ippiFilterMaxBorder
ippiFilterMedianHoriz_16s_AC4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)

Removed from 9.0	Substitution or Workaround
ippiFilterMedianHoriz_16s_C1R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16s_C3R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16s_C4R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16u_AC4R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16u_C1R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16u_C3R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_16u_C4R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_8u_AC4R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_8u_C1R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_8u_C3R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianHoriz_8u_C4R	Use <code>ippiFilterMedianBorder</code> with horiz mask ( <code>mask.height=1</code> )
ippiFilterMedianVert_16s_AC4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16s_C1R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16s_C3R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16s_C4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16u_AC4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16u_C1R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16u_C3R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_16u_C4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )

Removed from 9.0	Substitution or Workaround
ippiFilterMedianVert_8u_AC4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_8u_C1R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_8u_C3R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedianVert_8u_C4R	Use <code>ippiFilterMedianBorder</code> with vertical mask ( <code>mask.width=1</code> )
ippiFilterMedian_16s_AC4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16s_C1R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16s_C3R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16s_C4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16u_AC4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16u_C1R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16u_C3R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_16u_C4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_32f_C1R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_8u_AC4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_8u_C1R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_8u_C3R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMedian_8u_C4R	Use <code>ippiFilterMedianBorder</code>
ippiFilterMin_16s_AC4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16s_C1R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16s_C3R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16s_C4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16u_AC4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16u_C1R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16u_C3R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_16u_C4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_32f_AC4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_32f_C1R	Use <code>ippiFilterMinBorder</code>

Removed from 9.0	Substitution or Workaround
ippiFilterMin_32f_C3R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_32f_C4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_8u_AC4R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_8u_C1R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_8u_C3R	Use <code>ippiFilterMinBorder</code>
ippiFilterMin_8u_C4R	Use <code>ippiFilterMinBorder</code>
ippiFilterPrewittHoriz_16s_AC4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_16s_C1R</code>
ippiFilterPrewittHoriz_16s_C1R	<code>ippiFilterPrewittHorizBorder_16s_C1R</code>
ippiFilterPrewittHoriz_16s_C3R	Use <code>ippiCopy_16s_C3C1R</code> + <code>ippiFilterPrewittHorizBorder_16s_C1R</code>
ippiFilterPrewittHoriz_16s_C4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_16s_C1R</code>
ippiFilterPrewittHoriz_32f_AC4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_32f_C1R</code>
ippiFilterPrewittHoriz_32f_C1R	<code>ippiFilterPrewittHorizBorder_32f_C1R</code>
ippiFilterPrewittHoriz_32f_C3R	Use <code>ippiCopy_32f_C3C1R</code> + <code>ippiFilterPrewittHorizBorder_32f_C1R</code>
ippiFilterPrewittHoriz_32f_C4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_32f_C1R</code>
ippiFilterPrewittHoriz_8u_AC4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_8u16s_C1R</code>
ippiFilterPrewittHoriz_8u_C1R	<code>ippiFilterPrewittHorizBorder_8u16s_C1R</code>
ippiFilterPrewittHoriz_8u_C3R	Use <code>ippiCopy_8u_C3C1R</code> + <code>ippiFilterPrewittHorizBorder_8u16s_C1R</code>
ippiFilterPrewittHoriz_8u_C4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterPrewittHorizBorder_8u16s_C1R</code>
ippiFilterPrewittVert_16s_AC4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterPrewittVertBorder_16s_C1R</code>
ippiFilterPrewittVert_16s_C1R	<code>ippiFilterPrewittVertBorder_16s_C1R</code>
ippiFilterPrewittVert_16s_C3R	Use <code>ippiCopy_16s_C3C1R</code> + <code>ippiFilterPrewittVertBorder_16s_C1R</code>
ippiFilterPrewittVert_16s_C4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterPrewittVertBorder_16s_C1R</code>

Removed from 9.0	Substitution or Workaround
ippiFilterPrewittVert_32f_AC4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterPrewittVertBorder_32f_C1R</code>
ippiFilterPrewittVert_32f_C1R	ippiFilterPrewittVertBorder_32f_C1R
ippiFilterPrewittVert_32f_C3R	Use <code>ippiCopy_32f_C3C1R</code> + <code>ippiFilterPrewittVertBorder_32f_C1R</code>
ippiFilterPrewittVert_32f_C4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterPrewittVertBorder_32f_C1R</code>
ippiFilterPrewittVert_8u_AC4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterPrewittVertBorder_8u16s_C1R</code>
ippiFilterPrewittVert_8u_C1R	ippiFilterPrewittVertBorder_8u16s_C1R
ippiFilterPrewittVert_8u_C3R	Use <code>ippiCopy_8u_C3C1R</code> + <code>ippiFilterPrewittVertBorder_8u16s_C1R</code>
ippiFilterPrewittVert_8u_C4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterPrewittVertBorder_8u16s_C1R</code>
ippiFilterRobertsDown_16s_AC4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterRobertsDownBorder_16s_C1R</code>
ippiFilterRobertsDown_16s_C1R	ippiFilterRobertsDownBorder_16s_C1R
ippiFilterRobertsDown_16s_C3R	Use <code>ippiCopy_16s_C3C1R</code> + <code>ippiFilterRobertsDownBorder_16s_C1R</code>
ippiFilterRobertsDown_32f_AC4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterRobertsDownBorder_32f_C1R</code>
ippiFilterRobertsDown_32f_C1R	ippiFilterRobertsDownBorder_32f_C1R
ippiFilterRobertsDown_32f_C3R	Use <code>ippiCopy_32f_C3C1R</code> + <code>ippiFilterRobertsDownBorder_32f_C1R</code>
ippiFilterRobertsDown_8u_AC4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterRobertsDownBorder_8u16s_C1R</code>
ippiFilterRobertsDown_8u_C1R	ippiFilterRobertsDownBorder_8u16s_C1R
ippiFilterRobertsDown_8u_C3R	Use <code>ippiCopy_8u_C3C1R</code> + <code>ippiFilterRobertsDownBorder_8u16s_C1R</code>
ippiFilterRobertsUp_16s_AC4R	Use <code>ippiCopy_16s_C4C1R</code> + <code>ippiFilterRobertsUpBorder_16s_C1R</code>
ippiFilterRobertsUp_16s_C1R	ippiFilterRobertsUpBorder_16s_C1R
ippiFilterRobertsUp_16s_C3R	Use <code>ippiCopy_16s_C3C1R</code> + <code>ippiFilterRobertsUpBorder_16s_C1R</code>
ippiFilterRobertsUp_32f_AC4R	Use <code>ippiCopy_32f_C4C1R</code> + <code>ippiFilterRobertsUpBorder_32f_C1R</code>

Removed from 9.0	Substitution or Workaround
ippiFilterRobertsUp_32f_C1R	ippiFilterRobertsUpBorder_32f_C1R
ippiFilterRobertsUp_32f_C3R	Use <code>ippiCopy_32f_C3C1R</code> + <code>ippiFilterRobertsUpBorder_32f_C1R</code>
ippiFilterRobertsUp_8u_AC4R	Use <code>ippiCopy_8u_C4C1R</code> + <code>ippiFilterRobertsUpBorder_8u16s_C1R</code>
ippiFilterRobertsUp_8u_C1R	ippiFilterRobertsUpBorder_8u16s_C1R
ippiFilterRobertsUp_8u_C3R	Use <code>ippiCopy_8u_C3C1R</code> + <code>ippiFilterRobertsUpBorder_8u16s_C1R</code>
ippiFilterRoundGetBufSize16s_8u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_C3R	ippiFilterBorderGetSize

Removed from 9.0	Substitution or Workaround
ippiFilterRoundGetBufSize32s_16u_C4R	ippiFilterBorderGetSize
ippiFilterRow_64f_C1R	ippiFilter_64f_C1R
ippiFilterRow32f_16s_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R (with filter.height=1)
<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .	
ippiFilterRow32f_16s_C1R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C1R (with filter.height=1)
<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .	
ippiFilterRow32f_16s_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C3R (with filter.height=1)
<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .	
ippiFilterRow32f_16s_C4R	ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R (with filter.height=1)

Removed from 9.0	Substitution or Workaround
ippiFilterRow32f_16u_AC4R	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow32f_16u_C1R	<p><code>ippiFilterBorderInit_32f</code> +<code>ippiFilterBorder_16u_C1R</code> (with <code>filter.height=1</code>)</p> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow32f_16u_C3R	<p><code>ippiFilterBorderInit_32f</code> +<code>ippiFilterBorder_16u_C3R</code> (with <code>filter.height=1</code>)</p> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow32f_16u_C4R	<p><code>ippiFilterBorderInit_32f</code> +<code>ippiFilterBorder_16u_C4R</code> (with <code>filter.height=1</code>)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow32f_8u_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow32f_8u_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C1R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow32f_8u_C3R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C3R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow32f_8u_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code> (with <code>filter.height=1</code> )

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_16s_AC4R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R (with <code>filter.height=1</code>)</p> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_16s_C1R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C1R (with <code>filter.height=1</code>)</p> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_16s_C3R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C3R (with <code>filter.height=1</code>)</p> <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_16s_C4R	<p>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R (with <code>filter.height=1</code>)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_16u_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_16u_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C1R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_16u_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C3R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_16u_C4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code> (with <code>filter.height=1</code> )

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_32f_AC4R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with <code>filter.height=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_32f_C1R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C1R (with <code>filter.height=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_32f_C3R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C3R (with <code>filter.height=1</code>)</p>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilterRow_32f_C4R	<p>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with <code>filter.height=1</code>)</p>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_8u_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C4R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_8u_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C1R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_8u_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C3R</code> (with <code>filter.height=1</code> )
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilterRow_8u_C4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C4R</code> (with <code>filter.height=1</code> )

Removed from 9.0	Substitution or Workaround
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
<code>ippiFilterScharrHoriz_32f_C1R</code>	<code>ippiFilterScharrHorizMaskBorder_32f_C1R</code>
<code>ippiFilterScharrHoriz_8s16s_C1R</code>	<code>ippiConvert_8s16s</code> + <code>ippiFilterScharrHorizMaskBorder_16s_C1R</code>
<code>ippiFilterScharrHoriz_8u16s_C1R</code>	<code>ippiFilterScharrHorizMaskBorder_8u16s_C1R</code>
<code>ippiFilterScharrVert_32f_C1R</code>	<code>ippiFilterScharrVertMaskBorder_32f_C1R</code>
<code>ippiFilterScharrVert_8s16s_C1R</code>	<code>ippiConvert_8s16s</code> + <code>ippiFilterScharrVertMaskBorder_16s_C1R</code>
<code>ippiFilterScharrVert_8u16s_C1R</code>	<code>ippiFilterScharrVertMaskBorder_8u16s_C1R</code>
<code>ippiFilterSharpen_16s_AC4R</code>	<code>ippiFilterSharpenBorder_16s_AC4R</code>
<code>ippiFilterSharpen_16s_C1R</code>	<code>ippiFilterSharpenBorder_16s_C1R</code>
<code>ippiFilterSharpen_16s_C3R</code>	<code>ippiFilterSharpenBorder_16s_C3R</code>
<code>ippiFilterSharpen_16s_C4R</code>	<code>ippiFilterSharpenBorder_16s_C4R</code>
<code>ippiFilterSharpen_32f_AC4R</code>	<code>ippiFilterSharpenBorder_32f_AC4R</code>
<code>ippiFilterSharpen_32f_C1R</code>	<code>ippiFilterSharpenBorder_32f_C1R</code>
<code>ippiFilterSharpen_32f_C3R</code>	<code>ippiFilterSharpenBorder_32f_C3R</code>
<code>ippiFilterSharpen_32f_C4R</code>	<code>ippiFilterSharpenBorder_32f_C4R</code>
<code>ippiFilterSharpen_8u_AC4R</code>	<code>ippiFilterSharpenBorder_8u_AC4R</code>
<code>ippiFilterSharpen_8u_C1R</code>	<code>ippiFilterSharpenBorder_8u_C1R</code>
<code>ippiFilterSharpen_8u_C3R</code>	<code>ippiFilterSharpenBorder_8u_C3R</code>
<code>ippiFilterSharpen_8u_C4R</code>	<code>ippiFilterSharpenBorder_8u_C4R</code>
<code>ippiFilterSobelCross_32f_C1R</code>	<code>ippiFilterSobelCrossBorder_32f_C1R</code>
<code>ippiFilterSobelCross_8s16s_C1R</code>	<code>ippiConvert_8s32f</code> + <code>ippiFilterSobelCrossBorder_32f_C1R</code>
<code>ippiFilterSobelCross_8u16s_C1R</code>	<code>ippiFilterSobelCrossBorder_8u16s_C1R</code>
<code>ippiFilterSobelHorizGetBufferSize_32f_C1R</code>	<code>ippiFilterSobelHorizBorderGetBufferSize</code>
<code>ippiFilterSobelHorizGetBufferSize_8u16s_C1R</code>	<code>ippiFilterSobelHorizBorderGetBufferSize</code>

Removed from 9.0	Substitution or Workaround
ippiFilterSobelHorizMask_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHorizSecond_32f_C1R	ippiFilterSobelHorizSecondBorder_32f_C1R
ippiFilterSobelHorizSecond_8s16s_C1R	ippiConvert_8s32f +ippiFilterSobelHorizSecondBorder_32f_C1R
ippiFilterSobelHorizSecond_8u16s_C1R	ippiFilterSobelHorizSecondBorder_8u16s_C1R
ippiFilterSobelHoriz_16s_AC4R	UseippiCopy_16s_C4C1R +ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C1R	ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C3R	UseippiCopy_16s_C3C1R +ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C4R	UseippiCopy_16s_C4C1R +ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_32f_AC4R	UseippiCopy_32f_C4C1R +ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C3R	UseippiCopy_32f_C3C1R +ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C4R	UseippiCopy_32f_C4C1R +ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_8s16s_C1R	ippiConvert_8s16s +ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_8u16s_C1R	ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_AC4R	UseippiCopy_8u_C4C1R +ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C1R	ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C3R	UseippiCopy_8u_C3C1R +ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C4R	UseippiCopy_8u_C4C1R +ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelVertGetBufferSize_32f_C1R	ippiFilterSobelVertBorderGetBufferSize
ippiFilterSobelVertGetBufferSize_8u16s_C1R	ippiFilterSobelVertBorderGetBufferSize
ippiFilterSobelVertMask_32f_C1R	ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVertSecond_32f_C1R	ippiFilterSobelVertSecondBorder_32f_C1R

Removed from 9.0	Substitution or Workaround
ippiFilterSobelVertSecond_8s16s_C1R	ippiConvert_8s32f +ippiFilterSobelVertSecondBorder_32f_C1R
ippiFilterSobelVertSecond_8u16s_C1R	ippiFilterSobelVertSecondBorder_8u16s_C1R
ippiFilterSobelVert_16s_AC4R	<b>Use</b> ippiCopy_16s_C4C1R +ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C1R	ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C3R	<b>Use</b> ippiCopy_16s_C3C1R +ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C4R	<b>Use</b> ippiCopy_16s_C4C1R +ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_32f_AC4R	<b>Use</b> ippiCopy_32f_C4C1R +ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C1R	ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C3R	<b>Use</b> ippiCopy_32f_C3C1R +ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C4R	<b>Use</b> ippiCopy_32f_C4C1R +ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_8s16s_C1R	ippiConvert_8s16s +ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_8u16s_C1R	ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_AC4R	<b>Use</b> ippiCopy_8u_C4C1R +ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C1R	ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C3R	<b>Use</b> ippiCopy_8u_C3C1R +ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C4R	<b>Use</b> ippiCopy_8u_C4C1R +ippiFilterSobelVertBorder_8u16s_C1R
ippiFilter_16s_AC4R	ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

Removed from 9.0	Substitution or Workaround
ippiFilter_16s_C1R	ippiFilterBorderInit_16s +ippiFilterBorder_16s_C1R
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
ippiFilter_16s_C3R	ippiFilterBorderInit_16s +ippiFilterBorder_16s_C3R
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
ippiFilter_16s_C4R	ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
ippiFilter_16u_AC4R	ippiFilterBorderInit_16s +ippiFilterBorder_16u_C4R
	<b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code> .
ippiFilter_16u_C1R	ippiFilterBorderInit_16s +ippiFilterBorder_16u_C1R

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_16u_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_16u_C4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_32f_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_32f_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_32f_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_32f_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_32f_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_32f_C3R <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_32f_C4R	ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_8u_AC4R	ippiFilterBorderInit_16s +ippiFilterBorder_8u_C4R <p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_8u_C1R	ippiFilterBorderInit_16s +ippiFilterBorder_8u_C1R

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_8u_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_8u_C4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round16s_8u_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round16s_8u_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round16s_8u_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_8u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round16s_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16s_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round16s_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16s_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_16s_C3R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16s_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_16s_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16s_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_16u_AC4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_16u_C1R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_16u_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_Round32f_16u_C3R	ippiFilterBorderInit_32f +ippiFilterBorder_16u_C3R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_Round32f_16u_C4R	ippiFilterBorderInit_32f +ippiFilterBorder_16u_C4R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_Round32f_8u_AC4R	ippiFilterBorderInit_32f +ippiFilterBorder_8u_C4R
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
ippiFilter_Round32f_8u_C1R	ippiFilterBorderInit_32f +ippiFilterBorder_8u_C1R

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_8u_C3R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32f_8u_C4R</code>	<code>ippiFilterBorderInit_32f</code> + <code>ippiFilterBorder_8u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16s_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16s_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16s_C3R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16s_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16u_AC4R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16u_C1R</code>	<code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C1R</code>

Removed from 9.0	Substitution or Workaround
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16u_C3R</code>	<code>ippiFilterBorderInit_16s</code> <code>+ippiFilterBorder_16u_C3R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiFilter_Round32s_16u_C4R</code>	<code>ippiFilterBorderInit_16s</code> <code>+ippiFilterBorder_16u_C4R</code>
	<p><b>NOTE</b> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</p>
<code>ippiGetCentralMoment_64s</code>	Use <code>Moments_64f</code>
<code>ippiGetHuMoments_64s</code>	Use <code>Moments_64f</code>
<code>ippiGetNormalizedCentralMoment_64s</code>	Use <code>Moments_64f</code>
<code>ippiGetNormalizedSpatialMoment_64s</code>	Use <code>Moments_64f</code>
<code>ippiGetRotateBound</code>	N/A
<code>ippiGetRotateQuad</code>	N/A
<code>ippiGetShearBound</code>	N/A
<code>ippiGetShearQuad</code>	N/A
<code>ippiGetSpatialMoment_64s</code>	Use <code>Moments_64f</code>
<code>ippiHistogramEven_16s_AC4R</code>	<code>ippiHistogram_16s_C4R</code>
<code>ippiHistogramEven_16s_C1R</code>	<code>ippiHistogram_16s_C1R</code>
<code>ippiHistogramEven_16s_C3R</code>	<code>ippiHistogram_16s_C3R</code>

Removed from 9.0	Substitution or Workaround
ippiHistogramEven_16s_C4R	ippiHistogram_16s_C4R
ippiHistogramEven_16u_AC4R	ippiHistogram_16u_C4R
ippiHistogramEven_16u_C1R	ippiHistogram_16u_C1R
ippiHistogramEven_16u_C3R	ippiHistogram_16u_C3R
ippiHistogramEven_16u_C4R	ippiHistogram_16u_C4R
ippiHistogramEven_8u_AC4R	ippiHistogram_8u_C4R
ippiHistogramEven_8u_C1R	ippiHistogram_8u_C1R
ippiHistogramEven_8u_C3R	ippiHistogram_8u_C3R
ippiHistogramEven_8u_C4R	ippiHistogram_8u_C4R
ippiHistogramRange_16s_AC4R	ippiHistogram_16s_C4R
ippiHistogramRange_16s_C1R	ippiHistogram_16s_C1R
ippiHistogramRange_16s_C3R	ippiHistogram_16s_C3R
ippiHistogramRange_16s_C4R	ippiHistogram_16s_C4R
ippiHistogramRange_16u_AC4R	ippiHistogram_16u_C4R
ippiHistogramRange_16u_C1R	ippiHistogram_16u_C1R
ippiHistogramRange_16u_C3R	ippiHistogram_16u_C3R
ippiHistogramRange_16u_C4R	ippiHistogram_16u_C4R
ippiHistogramRange_32f_AC4R	ippiHistogram_32f_C4R
ippiHistogramRange_32f_C1R	ippiHistogram_32f_C1R
ippiHistogramRange_32f_C3R	ippiHistogram_32f_C3R
ippiHistogramRange_32f_C4R	ippiHistogram_32f_C4R
ippiHistogramRange_8u_AC4R	ippiHistogram_8u_C4R
ippiHistogramRange_8u_C1R	ippiHistogram_8u_C1R
ippiHistogramRange_8u_C3R	ippiHistogram_8u_C3R
ippiHistogramRange_8u_C4R	ippiHistogram_8u_C4R
ippiImageJaehne_32s_AC4R	ippiImageJaehne_32f_C1R +ippiConvert_32f32s
ippiImageJaehne_32s_C1R	ippiImageJaehne_32f_C1R +ippiConvert_32f32s
ippiImageJaehne_32s_C3R	ippiImageJaehne_32f_C3R +ippiConvert_32f32s

Removed from 9.0	Substitution or Workaround
ippiImageJaehne_32s_C4R	ippiImageJaehne_32f_C4R +ippiConvert_32f32s
ippiImageJaehne_8s_AC4R	ippiImageJaehne_16s_C4R+ippiConvert_16s8s
ippiImageJaehne_8s_C1R	ippiImageJaehne_16s_C1R+ippiConvert_16s8s
ippiImageJaehne_8s_C3R	ippiImageJaehne_16s_C3R+ippiConvert_16s8s
ippiImageJaehne_8s_C4R	ippiImageJaehne_16s_C4R+ippiConvert_16s8s
ippiImageRamp_32s_AC4R	ippiImageRamp_32f_C1R+ippiConvert_32f32s
ippiImageRamp_32s_C1R	ippiImageJaehne_32f_C1R +ippiConvert_32f32s
ippiImageRamp_32s_C3R	ippiImageJaehne_32f_C3R +ippiConvert_32f32s
ippiImageRamp_32s_C4R	ippiImageJaehne_32f_C4R +ippiConvert_32f32s
ippiImageRamp_8s_AC4R	ippiImageRamp_16s_C4R+ippiConvert_16s8s
ippiImageRamp_8s_C1R	ippiImageRamp_16s_C1R+ippiConvert_16s8s
ippiImageRamp_8s_C3R	ippiImageRamp_16s_C3R+ippiConvert_16s8s
ippiImageRamp_8s_C4R	ippiImageJaehne_16s_C4R+ippiConvert_16s8s
ippiLBPIImage3x3_32f8u_C1R	ippiLBPIImageMode3x3_32f8u_C1R
ippiLBPIImage3x3_8u_C1R	ippiLBPIImageMode3x3_8u_C1R
ippiLBPIImage5x5_32f16u_C1R	ippiLBPIImageMode5x5_32f16u_C1R
ippiLBPIImage5x5_32f8u_C1R	ippiLBPIImageMode5x5_32f8u_C1R
ippiLBPIImage5x5_8u16u_C1R	ippiLBPIImageMode5x5_8u16u_C1R
ippiLBPIImage5x5_8u_C1R	ippiLBPIImageMode5x5_8u_C1R
ippiLUTPaletteSwap_16u_C3A0C4R	N/A
ippiLUTPaletteSwap_8u_C3A0C4R	N/A
ippiLUTPalette_16u_C3A0C4R	N/A
ippiLUTPalette_8u_C3A0C4R	N/A
ippiLUT_Cubic	UseippiLUT
ippiLUT_Linear	UseippiLUT
ippiMagnitudePack_16s_C1RSfs	ippiConvert_16s32f+ippiMagnitudePack_32f
ippiMagnitudePack_16s_C3RSfs	ippiConvert_16s32f+ippiMagnitudePack_32f

Removed from 9.0	Substitution or Workaround
ippiMagnitudePack_32s_C1RSfs	ippiConvert_32s32f+ippiMagnitudePack_32f
ippiMagnitudePack_32s_C3RSfs	ippiConvert_32s32f+ippiMagnitudePack_32f
ippiMagnitude_16sc16s_C1RSfs	ippiConvert_16s32f+ippiMagnitude_32f
ippiMagnitude_16sc16s_C3RSfs	ippiConvert_16s32f+ippiMagnitude_32f
ippiMagnitude_16uc16u_C1RSfs	ippiConvert_16u32f+ippiMagnitude_32f
ippiMagnitude_16uc16u_C3RSfs	ippiConvert_16u32f+ippiMagnitude_32f
ippiMagnitude_32sc32s_C1RSfs	ippiConvert_32s32f+ippiMagnitude_32f
ippiMagnitude_32sc32s_C3RSfs	ippiConvert_32s32f+ippiMagnitude_32f
ippiMean_16s_AC4R	ippiMean_16s_C4R
ippiMean_16u_AC4R	ippiMean_16u_C4R
ippiMean_32f_AC4R	ippiMean_32f_C4R
ippiMean_8u_AC4R	ippiMean_8u_C4R
ippiMomentFree_64f	ippiFree
ippiMomentFree_64s	Use Moments_64f
ippiMomentGetStateSize_64s	Use Moments_64f
ippiMomentInitAlloc_64f	ippiMomentsGetSize+ippiMalloc +ippiMomentsInit
ippiMomentInitAlloc_64s	Use Moments_64f
ippiMomentInit_64s	Use Moments_64f
ippiMoments64s_16u_AC4R	Use Moments_64f
ippiMoments64s_16u_C1R	Use Moments_64f
ippiMoments64s_16u_C3R	Use Moments_64f
ippiMoments64s_8u_AC4R	Use Moments_64f
ippiMoments64s_8u_C1R	Use Moments_64f
ippiMoments64s_8u_C3R	Use Moments_64f
ippiMulC_16sc_AC4IRSfs	N/A
ippiMulC_16sc_AC4RSfs	N/A
ippiMulC_16sc_C1IRSfs	N/A
ippiMulC_16sc_C1RSfs	N/A
ippiMulC_16sc_C3IRSfs	N/A
ippiMulC_16sc_C3RSfs	N/A

Removed from 9.0	Substitution or Workaround
ippiMulC_32fc_AC4IR	N/A
ippiMulC_32fc_AC4R	N/A
ippiMulC_32fc_C1IR	N/A
ippiMulC_32fc_C1R	N/A
ippiMulC_32fc_C3IR	N/A
ippiMulC_32fc_C3R	N/A
ippiMulC_32sc_AC4IRSfs	N/A
ippiMulC_32sc_AC4RSfs	N/A
ippiMulC_32sc_C1IRSfs	N/A
ippiMulC_32sc_C1RSfs	N/A
ippiMulC_32sc_C3IRSfs	N/A
ippiMulC_32sc_C3RSfs	N/A
ippiMulPack_16s_AC4IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_AC4RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C1IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C1RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C3IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C3RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C4IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_16s_C4RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_AC4IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_AC4RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C1IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C1RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C3IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C3RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C4IRSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMulPack_32s_C4RSfs	ippiConvert_16s32f+ippiMulPack_32f
ippiMul_16sc_AC4IRSfs	N/A
ippiMul_16sc_AC4RSfs	N/A

Removed from 9.0	Substitution or Workaround
ippiMul_16sc_C1IRSfs	N/A
ippiMul_16sc_C1RSfs	N/A
ippiMul_16sc_C3IRSfs	N/A
ippiMul_16sc_C3RSfs	N/A
ippiMul_32fc_AC4IR	N/A
ippiMul_32fc_AC4R	N/A
ippiMul_32fc_C1IR	N/A
ippiMul_32fc_C1R	N/A
ippiMul_32fc_C3IR	N/A
ippiMul_32fc_C3R	N/A
ippiMul_32sc_AC4IRSfs	N/A
ippiMul_32sc_AC4RSfs	N/A
ippiMul_32sc_C1IRSfs	N/A
ippiMul_32sc_C1RSfs	N/A
ippiMul_32sc_C3IRSfs	N/A
ippiMul_32sc_C3RSfs	N/A
ippiNormDiff_Inf_16s_AC4R	ippiNormDiff_Inf_16s_C4R
ippiNormDiff_Inf_16u_AC4R	ippiNormDiff_Inf_16u_C4R
ippiNormDiff_Inf_32f_AC4R	ippiNormDiff_Inf_32f_C4R
ippiNormDiff_Inf_8u_AC4R	ippiNormDiff_Inf_8u_C4R
ippiNormDiff_L1_16s_AC4R	ippiNormDiff_L1_16s_C4R
ippiNormDiff_L1_16u_AC4R	ippiNormDiff_L1_16u_C4R
ippiNormDiff_L1_32f_AC4R	ippiNormDiff_L1_32f_C4R
ippiNormDiff_L1_8u_AC4R	ippiNormDiff_L1_8u_C4R
ippiNormDiff_L2_16s_AC4R	ippiNormDiff_L2_16s_C4R
ippiNormDiff_L2_16u_AC4R	ippiNormDiff_L2_16u_C4R
ippiNormDiff_L2_32f_AC4R	ippiNormDiff_L2_32f_C4R
ippiNormDiff_L2_8u_AC4R	ippiNormDiff_L2_8u_C4R
ippiNormRel_Inf_16s_AC4R	ippiNormRel_Inf_16s_C4R
ippiNormRel_Inf_16u_AC4R	ippiNormRel_Inf_16u_C4R

Removed from 9.0	Substitution or Workaround
ippiNormRel_Inf_32f_AC4R	ippiNormRel_Inf_32f_C4R
ippiNormRel_Inf_8u_AC4R	ippiNormRel_Inf_8u_C4R
ippiNormRel_L1_16s_AC4R	ippiNormRel_L1_16s_C4R
ippiNormRel_L1_16u_AC4R	ippiNormRel_L1_16u_C4R
ippiNormRel_L1_32f_AC4R	ippiNormRel_L1_32f_C4R
ippiNormRel_L1_8u_AC4R	ippiNormRel_L1_8u_C4R
ippiNormRel_L2_16s_AC4R	ippiNormRel_L2_16s_C4R
ippiNormRel_L2_16u_AC4R	ippiNormRel_L2_16u_C4R
ippiNormRel_L2_32f_AC4R	ippiNormRel_L2_32f_C4R
ippiNormRel_L2_8u_AC4R	ippiNormRel_L2_8u_C4R
ippiNorm_Inf_16s_AC4R	ippiNorm_Inf_16s_C4R
ippiNorm_Inf_16u_AC4R	ippiNorm_Inf_16u_C4R
ippiNorm_Inf_32f_AC4R	ippiNorm_Inf_32f_C4R
ippiNorm_Inf_32s_C1R	ippiConvert_32s32f+ippiNorm_Inf_32f
ippiNorm_Inf_8u_AC4R	ippiNorm_Inf_8u_C4R
ippiNorm_L1_16s_AC4R	ippiNorm_L1_16s_C4R
ippiNorm_L1_16u_AC4R	ippiNorm_L1_16u_C4R
ippiNorm_L1_32f_AC4R	ippiNorm_L1_32f_C4R
ippiNorm_L1_8u_AC4R	ippiNorm_L1_8u_C4R
ippiNorm_L2_16s_AC4R	ippiNorm_L2_16s_C4R
ippiNorm_L2_16u_AC4R	ippiNorm_L2_16u_C4R
ippiNorm_L2_32f_AC4R	ippiNorm_L2_32f_C4R
ippiNorm_L2_8u_AC4R	ippiNorm_L2_8u_C4R
ippiPackToCplxExtend_32s32sc_C1R	ippiConvert_32s32f +ippiPackToCplxExtend_32f
ippiPhasePack_16s_C1RSfs	ippiConvert_16s32f+ippiPhasePack_32f
ippiPhasePack_16s_C3RSfs	ippiConvert_16s32f+ippiPhasePack_32f
ippiPhasePack_32s_C1RSfs	ippiConvert_32s32f+ippiPhasePack_32f
ippiPhasePack_32s_C3RSfs	ippiConvert_32s32f+ippiPhasePack_32f
ippiPhase_16sc16s_C1RSfs	ippiConvert_16s32f+ippiPhase_32f

Removed from 9.0	Substitution or Workaround
ippiPhase_16sc16s_C3RSfs	ippiConvert_16s32f+ippiPhase_32f
ippiPhase_16uc16u_C1RSfs	ippiConvert_16u32f+ippiPhase_32f
ippiPhase_16uc16u_C3RSfs	ippiConvert_16u32f+ippiPhase_32f
ippiPhase_32sc32s_C1RSfs	ippiConvert_32s32f+ippiPhase_32f
ippiPhase_32sc32s_C3RSfs	ippiConvert_32s32f+ippiPhase_32f
ippiPolarToCart_16sc_C1R	ippiConvert_16s32f +ippiPolarToCart_32fc_C1R
ippiPolarToCart_16sc_C3R	ippiConvert_16s32f +ippiPolarToCart_32fc_C3R
ippiPolarToCart_32f32fc_P2C1R	Use ippsCplxToReal_32fc +ippsPolarToCart_32fc with loop by row
ippiPolarToCart_32sc_C1R	ippiConvert_32s32f +ippiPolarToCart_32fc_C1R
ippiPolarToCart_32sc_C3R	ippiConvert_32s32f +ippiPolarToCart_32fc_C3R
ippiRShiftC_8s_AC4IR	Convert to other data type
ippiRShiftC_8s_AC4R	Convert to other data type
ippiRShiftC_8s_C1IR	Convert to other data type
ippiRShiftC_8s_C1R	Convert to other data type
ippiRShiftC_8s_C3IR	Convert to other data type
ippiRShiftC_8s_C3R	Convert to other data type
ippiRShiftC_8s_C4IR	Convert to other data type
ippiRShiftC_8s_C4R	Convert to other data type
ippiRemap_16s_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16s_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16u_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16u_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_32f_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_32f_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_64f_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_64f_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_8u_P3R	Use ippiRemap_XX_C1R for each plane

Removed from 9.0	Substitution or Workaround
ippiRemap_8u_P4R	Use <code>ippiRemap_XX_C1R</code> for each plane
ippiResampleRowGetBorderWidth_32f	N/A
ippiResampleRowGetSize_32f	N/A
ippiResampleRowInit_32f	N/A
ippiResampleRowReplicateBorder_32f_C1R	N/A
ippiResampleRow_32f_C1	N/A
ippiResizeGetBufSize	N/A
ippiResizeGetBufSize_64f	N/A
ippiResizeSqrPixel_16s_AC4R	N/A
ippiResizeSqrPixel_16s_C1R	N/A
ippiResizeSqrPixel_16s_C3R	N/A
ippiResizeSqrPixel_16s_C4R	N/A
ippiResizeSqrPixel_16s_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_16s_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_16u_AC4R	N/A
ippiResizeSqrPixel_16u_C1R	N/A
ippiResizeSqrPixel_16u_C3R	N/A
ippiResizeSqrPixel_16u_C4R	N/A
ippiResizeSqrPixel_16u_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_16u_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_32f_AC4R	N/A
ippiResizeSqrPixel_32f_C1R	N/A
ippiResizeSqrPixel_32f_C3R	N/A
ippiResizeSqrPixel_32f_C4R	N/A
ippiResizeSqrPixel_32f_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_32f_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_64f_AC4R	N/A
ippiResizeSqrPixel_64f_C1R	N/A
ippiResizeSqrPixel_64f_C3R	N/A
ippiResizeSqrPixel_64f_C4R	N/A
ippiResizeSqrPixel_64f_P3R	Use C1R flavor for each plane

Removed from 9.0	Substitution or Workaround
ippiResizeSqrPixel_64f_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_8u_AC4R	N/A
ippiResizeSqrPixel_8u_C1R	N/A
ippiResizeSqrPixel_8u_C3R	N/A
ippiResizeSqrPixel_8u_C4R	N/A
ippiResizeSqrPixel_8u_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_8u_P4R	Use C1R flavor for each plane
ippiResizeYUV420GetBufSize	N/A
ippiResizeYUV420_8u_P2R	N/A
ippiResizeYUV422_8u_C2R	N/A
ippiRotateCenter_16u_AC4R	See the <a href="#">RotateCenter.c</a> example.
ippiRotateCenter_16u_C1R	
ippiRotateCenter_16u_C3R	
ippiRotateCenter_16u_C4R	
ippiRotateCenter_16u_P3R	
ippiRotateCenter_16u_P4R	
ippiRotateCenter_32f_AC4R	
ippiRotateCenter_32f_C1R	
ippiRotateCenter_32f_C3R	
ippiRotateCenter_32f_C4R	
ippiRotateCenter_32f_P3R	
ippiRotateCenter_32f_P4R	
ippiRotateCenter_64f_AC4R	
ippiRotateCenter_64f_C1R	
ippiRotateCenter_64f_C3R	
ippiRotateCenter_64f_C4R	
ippiRotateCenter_64f_P3R	
ippiRotateCenter_64f_P4R	
ippiRotateCenter_8u_AC4R	
ippiRotateCenter_8u_C1R	
ippiRotateCenter_8u_C3R	
ippiRotateCenter_8u_C4R	
ippiRotateCenter_8u_P3R	
ippiRotateCenter_8u_P4R	
ippiRotate_16u_AC4R	See the <a href="#">Rotate.c</a> example.
ippiRotate_16u_C1R	
ippiRotate_16u_C3R	
ippiRotate_16u_C4R	

Removed from 9.0	Substitution or Workaround
ippiRotate_16u_P3R	
ippiRotate_16u_P4R	
ippiRotate_32f_AC4R	
ippiRotate_32f_C1R	
ippiRotate_32f_C3R	
ippiRotate_32f_C4R	
ippiRotate_32f_P3R	
ippiRotate_32f_P4R	
ippiRotate_64f_AC4R	
ippiRotate_64f_C1R	
ippiRotate_64f_C3R	
ippiRotate_64f_C4R	
ippiRotate_64f_P3R	
ippiRotate_64f_P4R	
ippiRotate_8u_AC4R	
ippiRotate_8u_C1R	
ippiRotate_8u_C3R	
ippiRotate_8u_C4R	
ippiRotate_8u_P3R	
ippiRotate_8u_P4R	
ippiSSIM_32f_C1R	N/A
ippiShear_16u_AC4R	N/A
ippiShear_16u_C1R	N/A
ippiShear_16u_C3R	N/A
ippiShear_16u_C4R	N/A
ippiShear_16u_P3R	Use C1R flavor for each plane
ippiShear_16u_P4R	Use C1R flavor for each plane
ippiShear_32f_AC4R	N/A
ippiShear_32f_C1R	N/A
ippiShear_32f_C3R	N/A
ippiShear_32f_C4R	N/A
ippiShear_32f_P3R	Use C1R flavor for each plane
ippiShear_32f_P4R	Use C1R flavor for each plane
ippiShear_64f_AC4R	N/A
ippiShear_64f_C1R	N/A
ippiShear_64f_C3R	N/A

Removed from 9.0	Substitution or Workaround
ippiShear_64f_C4R	N/A
ippiShear_64f_P3R	Use C1R flavor for each plane
ippiShear_64f_P4R	Use C1R flavor for each plane
ippiShear_8u_AC4R	N/A
ippiShear_8u_C1R	N/A
ippiShear_8u_C3R	N/A
ippiShear_8u_C4R	N/A
ippiShear_8u_P3R	Use C1R flavor for each plane
ippiShear_8u_P4R	Use C1R flavor for each plane
ippiSqrDistanceFull_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceFull_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8s32f_C1R	ippiConvert_8s32f +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm

Removed from 9.0	Substitution or Workaround
ippiSqrDistanceFull_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceFull_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIFull ippiNorm
ippiSqrDistanceFull_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiSqrDistanceSame_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROISame ippiNorm

Removed from 9.0	Substitution or Workaround
ippiSqrDistanceSame_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C1R	ippiConvert_8s32f +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROISame ippiNorm

Removed from 9.0	Substitution or Workaround
ippiSqrDistanceSame_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROISame ippiNorm
ippiSqrDistanceSame_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROISame ippiNorm
ippiSqrDistanceValid_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C3R	ippiCopy_16u_C3C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C4R	ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_32f_AC4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_32f_C3R	ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_32f_C4R	ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8s32f_AC4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8s32f_C1R	ippiConvert_8s32f +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8s32f_C3R	ippiConvert_8s32f+ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8s32f_C4R	ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid ippiNorm

Removed from 9.0	Substitution or Workaround
ippiSqrDistanceValid_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C3R	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C4R	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u32f_C1R +algType=ippiROISame ippiNorm
ippiSqrDistanceValid_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiSqrDistanceValid_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R +ippiSqrDistanceNorm_8u_C1RSfs +algType=ippiROIValid ippiNorm
ippiSubC_16sc_AC4IRSfs	N/A
ippiSubC_16sc_AC4RSfs	N/A
ippiSubC_16sc_C1IRSfs	N/A
ippiSubC_16sc_C1RSfs	N/A
ippiSubC_16sc_C3IRSfs	N/A
ippiSubC_16sc_C3RSfs	N/A
ippiSubC_32fc_AC4IR	N/A
ippiSubC_32fc_AC4R	N/A
ippiSubC_32fc_C1IR	N/A
ippiSubC_32fc_C1R	N/A
ippiSubC_32fc_C3IR	N/A
ippiSubC_32fc_C3R	N/A
ippiSubC_32sc_AC4IRSfs	N/A
ippiSubC_32sc_AC4RSfs	N/A

Removed from 9.0	Substitution or Workaround
ippiSubC_32sc_C1IRSfs	N/A
ippiSubC_32sc_C1RSfs	N/A
ippiSubC_32sc_C3IRSfs	N/A
ippiSubC_32sc_C3RSfs	N/A
ippiSub_16sc_AC4IRSfs	N/A
ippiSub_16sc_AC4RSfs	N/A
ippiSub_16sc_C1IRSfs	N/A
ippiSub_16sc_C1RSfs	N/A
ippiSub_16sc_C3IRSfs	N/A
ippiSub_16sc_C3RSfs	N/A
ippiSub_32fc_AC4IR	N/A
ippiSub_32fc_AC4R	N/A
ippiSub_32fc_C1IR	N/A
ippiSub_32fc_C1R	N/A
ippiSub_32fc_C3IR	N/A
ippiSub_32fc_C3R	N/A
ippiSub_32sc_AC4IRSfs	N/A
ippiSub_32sc_AC4RSfs	N/A
ippiSub_32sc_C1IRSfs	N/A
ippiSub_32sc_C1RSfs	N/A
ippiSub_32sc_C3IRSfs	N/A
ippiSub_32sc_C3RSfs	N/A
ippiSum_16s_AC4R	ippiSum_16s_C4R
ippiSum_16u_AC4R	ippiSum_16u_C4R
ippiSum_32f_AC4R	ippiSum_32f_C4R
ippiSum_8u_AC4R	ippiSum_8u_C4R
ippiSuperSamplingGetBufSize	N/A
ippiSuperSampling_16s_AC4R	N/A
ippiSuperSampling_16s_C1R	N/A
ippiSuperSampling_16s_C3R	N/A
ippiSuperSampling_16s_C4R	N/A

Removed from 9.0	Substitution or Workaround
ippiSuperSampling_16s_P3R	Use C1R flavor for each plane
ippiSuperSampling_16s_P4R	Use C1R flavor for each plane
ippiSuperSampling_16u_AC4R	N/A
ippiSuperSampling_16u_C1R	N/A
ippiSuperSampling_16u_C3R	N/A
ippiSuperSampling_16u_C4R	N/A
ippiSuperSampling_16u_P3R	Use C1R flavor for each plane
ippiSuperSampling_16u_P4R	Use C1R flavor for each plane
ippiSuperSampling_32f_AC4R	N/A
ippiSuperSampling_32f_C1R	N/A
ippiSuperSampling_32f_C3R	N/A
ippiSuperSampling_32f_C4R	N/A
ippiSuperSampling_32f_P3R	Use C1R flavor for each plane
ippiSuperSampling_32f_P4R	Use C1R flavor for each plane
ippiSuperSampling_8u_AC4R	N/A
ippiSuperSampling_8u_C1R	N/A
ippiSuperSampling_8u_C3R	N/A
ippiSuperSampling_8u_C4R	N/A
ippiSuperSampling_8u_P3R	Use C1R flavor for each plane
ippiSuperSampling_8u_P4R	Use C1R flavor for each plane
ippiWTFwdFree_32f_C1R	ippiFree
ippiWTFwdFree_32f_C3R	ippiFree
ippiWTFwdGetBufSize_C1R	N/A
ippiWTFwdGetBufSize_C3R	N/A
ippiWTFwdInitAlloc_32f_C1R	ippiWTFwdGetSize+ippiMalloc +ippiWTFwdInit
ippiWTFwdInitAlloc_32f_C3R	ippiWTFwdGetSize+ippiMalloc +ippiWTFwdInit
ippiWTInvFree_32f_C1R	ippiFree
ippiWTInvFree_32f_C3R	ippiFree
ippiWTInvGetBufSize_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiWTInvGetBufSize_C3R	N/A
ippiWTInvInitAlloc_32f_C1R	ippiWTInvGetSize+ippiMalloc +ippiWTInvInit
ippiWTInvInitAlloc_32f_C3R	ippiWTInvGetSize+ippiMalloc +ippiWTInvInit
ippiWarpAffineBack_16u_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_16u_C1R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_16u_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_16u_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_16u_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_16u_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_32f_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpAffineBack_32f_C1R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_32f_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_32f_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_32f_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_32f_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_8u_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_8u_C1R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpAffineBack_8u_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_8u_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_8u_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineBack_8u_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_16u_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_16u_C1R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_16u_C3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_16u_C4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/">https://software.intel.com/content/www/us/en/</a>

Removed from 9.0	Substitution or Workaround
ippiWarpAffineQuad_16u_P3R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_16u_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_32f_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_32f_C1R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_32f_C3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_32f_C4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_32f_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpAffineQuad_32f_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_C1R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_C3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_C4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffineQuad_8u_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_16u_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/">https://software.intel.com/content/www/us/en/</a>

Removed from 9.0	Substitution or Workaround
ippiWarpAffine_16u_C1R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_16u_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_16u_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_16u_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_16u_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_32f_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_32f_C1R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpAffine_32f_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_32f_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_32f_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_32f_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_64f_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_64f_C1R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_64f_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_64f_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/">https://software.intel.com/content/www/us/en/</a>

Removed from 9.0	Substitution or Workaround
ippiWarpAffine_64f_P3R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_64f_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_8u_AC4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_8u_C1R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_8u_C3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_8u_C4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpAffine_8u_P3R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpAffine_8u_P4R	Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearBack_16u_AC4R	N/A
ippiWarpBilinearBack_16u_P3R	Use C1R flavor for each plane
ippiWarpBilinearBack_16u_P4R	Use C1R flavor for each plane
ippiWarpBilinearBack_32f_AC4R	N/A
ippiWarpBilinearBack_32f_P3R	Use C1R flavor for each plane
ippiWarpBilinearBack_32f_P4R	Use C1R flavor for each plane
ippiWarpBilinearBack_8u_AC4R	N/A
ippiWarpBilinearBack_8u_P3R	Use C1R flavor for each plane N/A
ippiWarpBilinearBack_8u_P4R	Use C1R flavor for each plane
ippiWarpBilinearQuad_16u_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_16u_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_16u_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_32f_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_32f_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpBilinearQuad_32f_P4R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_8u_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_8u_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinearQuad_8u_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpBilinear_16u_AC4R	N/A
ippiWarpBilinear_16u_P3R	Use C1R flavor for each plane
ippiWarpBilinear_16u_P4R	Use C1R flavor for each plane
ippiWarpBilinear_32f_AC4R	N/A
ippiWarpBilinear_32f_P3R	Use C1R flavor for each plane
ippiWarpBilinear_32f_P4R	Use C1R flavor for each plane
ippiWarpBilinear_8u_AC4R	N/A
ippiWarpBilinear_8u_P3R	Use C1R flavor for each plane
ippiWarpBilinear_8u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_16u_AC4R	N/A
ippiWarpPerspectiveBack_16u_C1R	N/A
ippiWarpPerspectiveBack_16u_C3R	N/A
ippiWarpPerspectiveBack_16u_C4R	N/A

Removed from 9.0	Substitution or Workaround
ippiWarpPerspectiveBack_16u_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_16u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_32f_AC4R	N/A
ippiWarpPerspectiveBack_32f_C1R	N/A
ippiWarpPerspectiveBack_32f_C3R	N/A
ippiWarpPerspectiveBack_32f_C4R	N/A
ippiWarpPerspectiveBack_32f_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_32f_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_8u_AC4R	N/A
ippiWarpPerspectiveBack_8u_C1R	N/A
ippiWarpPerspectiveBack_8u_C3R	N/A
ippiWarpPerspectiveBack_8u_C4R	N/A
ippiWarpPerspectiveBack_8u_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_8u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveQuad_16u_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_16u_C1R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_16u_C3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_16u_C4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpPerspectiveQuad_16u_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_16u_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_AC4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_C1R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_C3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_C4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_P3R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspectiveQuad_32f_P4R	Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpPerspectiveQuad_8u_AC4R	<p><a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspectiveQuad_8u_C1R	<p>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspectiveQuad_8u_C3R	<p>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspectiveQuad_8u_C4R	<p>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspectiveQuad_8u_P3R	<p>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspectiveQuad_8u_P4R	<p>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>
ippiWarpPerspective_16u_AC4R	<p>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a>.</p>

Removed from 9.0	Substitution or Workaround
ippiWarpPerspective_16u_C1R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_16u_C3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_16u_C4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_16u_P3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_16u_P4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_32f_AC4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_32f_C1R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_32f_C3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/">https://software.intel.com/</a>

Removed from 9.0	Substitution or Workaround
ippiWarpPerspective_32f_C4R	<a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_32f_P3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_32f_P4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_8u_AC4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_8u_C1R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_8u_C3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_8u_C4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

Removed from 9.0	Substitution or Workaround
ippiWarpPerspective_8u_P3R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .
ippiWarpPerspective_8u_P4R	Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html">https://software.intel.com/content/www/us/en/develop/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them.html</a> .

[ippj.h](#):

Removed from 9.0	Substitution or Workaround
ippiAdd128_JPEG_16s8u_C1R	N/A
ippiBGR555ToYCbCr411LS_MCU_16u16s_C3P3R	N/A
ippiBGR555ToYCbCr422LS_MCU_16u16s_C3P3R	N/A
ippiBGR555ToYCbCr444LS_MCU_16u16s_C3P3R	N/A
ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr411LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr422LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr444LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr411_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr411_JPEG_8u_C4P3R	N/A
ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr422_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr422_JPEG_8u_C4P3R	N/A
ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr_JPEG_8u_C4P3R	N/A
ippiBGRToY_JPEG_8u_C3C1R	N/A
ippiCMYKToYCK411LS_MCU_8u16s_C4P4R	N/A

Removed from 9.0	Substitution or Workaround
ippiCMYKToYCK422LS_MCU_8u16s_C4P4R	N/A
ippiCMYKToYCK444LS_MCU_8u16s_C4P4R	N/A
ippiCMYKToYCK_JPEG_8u_C4P4R	N/A
ippiCMYKToYCK_JPEG_8u_P4R	N/A
ippiDCTQuantFwd8x8LS_JPEG_16u16s_C1R	N/A
ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R	N/A
ippiDCTQuantFwd8x8_JPEG_16s_C1	N/A
ippiDCTQuantFwd8x8_JPEG_16s_C1I	N/A
ippiDCTQuantInv8x8LS_1x1_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_2x2_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_4x4_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_JPEG_16s16u_C1R	N/A
ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8To2x2LS_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8To4x4LS_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8_JPEG_16s_C1	N/A
ippiDCTQuantInv8x8_JPEG_16s_C1I	N/A
ippiDecodeCBProgrAttach_JPEG2K_32s_C1R	N/A
ippiDecodeCBProgrFree_JPEG2K	N/A
ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K	N/A
ippiDecodeCBProgrGetPassCounter_JPEG2K	N/A
ippiDecodeCBProgrGetSize_JPEG2K	N/A
ippiDecodeCBProgrInitAlloc_JPEG2K	N/A
ippiDecodeCBProgrInit_JPEG2K	N/A
ippiDecodeCBProgrSetPassCounter_JPEG2K	N/A
ippiDecodeCBProgrStep_JPEG2K	N/A
ippiDecodeCodeBlock_JPEG2K_1u32s_C1R	N/A
ippiDecodeGetBufSize_JPEG2K	N/A
ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1	N/A

Removed from 9.0	Substitution or Workaround
ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_JPEG_1u16s_C1	N/A
ippiDecodeHuffmanOne_JPEG_1u16s_C1	N/A
ippiDecodeHuffmanRow_JPEG_1u16s_C1P4	N/A
ippiDecodeHuffmanSpecFree_JPEG_8u	N/A
ippiDecodeHuffmanSpecGetBufSize_JPEG_8u	N/A
ippiDecodeHuffmanSpecInitAlloc_JPEG_8u	N/A
ippiDecodeHuffmanSpecInit_JPEG_8u	N/A
ippiDecodeHuffmanStateFree_JPEG_8u	N/A
ippiDecodeHuffmanStateGetBufSize_JPEG_8u	N/A
ippiDecodeHuffmanStateInitAlloc_JPEG_8u	N/A
ippiDecodeHuffmanStateInit_JPEG_8u	N/A
ippiDiffPredFirstRow_JPEG_16s_C1	N/A
ippiDiffPredRow_JPEG_16s_C1	N/A
ippiEncodeFree_JPEG2K	N/A
ippiEncodeGetDist_JPEG2K	N/A
ippiEncodeGetRate_JPEG2K	N/A
ippiEncodeGetTermPassLen_JPEG2K	N/A
ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_JPEG_16s1u_C1	N/A
ippiEncodeHuffmanOne_JPEG_16s1u_C1	N/A
ippiEncodeHuffmanRawTableInit_JPEG_8u	N/A
ippiEncodeHuffmanRow_JPEG_16s1u_P4C1	N/A
ippiEncodeHuffmanSpecFree_JPEG_8u	N/A
ippiEncodeHuffmanSpecGetBufSize_JPEG_8u	N/A
ippiEncodeHuffmanSpecInitAlloc_JPEG_8u	N/A

Removed from 9.0	Substitution or Workaround
ippiEncodeHuffmanSpecInit_JPEG_8u	N/A
ippiEncodeHuffmanStateFree_JPEG_8u	N/A
ippiEncodeHuffmanStateGetBufSize_JPEG_8u	N/A
ippiEncodeHuffmanStateInitAlloc_JPEG_8u	N/A
ippiEncodeHuffmanStateInit_JPEG_8u	N/A
ippiEncodeInitAlloc_JPEG2K	N/A
ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R	N/A
ippiEncodeStoreBits_JPEG2K_1u	N/A
ippiFLCDecode4x4_JPEGXR_1u16s_C1IR	N/A
ippiFLCDecode4x4_JPEGXR_1u32s_C1IR	N/A
ippiFLCEncode4x4_JPEGXR_16s1u_C1R	N/A
ippiFLCEncode4x4_JPEGXR_32s1u_C1R	N/A
ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_JPEG_16s_C1	N/A
ippiGetHuffmanStatisticsOne_JPEG_16s_C1	N/A
ippiICTFwd_JPEG2K_16s_P3IR	N/A
ippiICTFwd_JPEG2K_32f_C3P3R	N/A
ippiICTFwd_JPEG2K_32f_P3IR	N/A
ippiICTFwd_JPEG2K_32s_P3IR	N/A
ippiICTInv_JPEG2K_16s_P3IR	N/A
ippiICTInv_JPEG2K_32f_P3C3R	N/A
ippiICTInv_JPEG2K_32f_P3IR	N/A
ippiICTInv_JPEG2K_32s_P3IR	N/A
ippiJoin422LS_MCU_16s8u_P3C2R	N/A
ippiJoinRow_TIFF_8u16u_C1	N/A
ippiPCTFwd16x16_JPEGXR_16s_C1IR	N/A
ippiPCTFwd16x16_JPEGXR_32f_C1IR	N/A
ippiPCTFwd16x16_JPEGXR_32s_C1IR	N/A
ippiPCTFwd8x16_JPEGXR_16s_C1IR	N/A

Removed from 9.0	Substitution or Workaround
ippiPCTFwd8x16_JPEGXR_32f_C1IR	N/A
ippiPCTFwd8x16_JPEGXR_32s_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_16s_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_32f_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_32s_C1IR	N/A
ippiPCTFwd_JPEGXR_16s_C1IR	N/A
ippiPCTFwd_JPEGXR_32f_C1IR	N/A
ippiPCTFwd_JPEGXR_32s_C1IR	N/A
ippiPCTInv16x16_JPEGXR_16s_C1IR	N/A
ippiPCTInv16x16_JPEGXR_32f_C1IR	N/A
ippiPCTInv16x16_JPEGXR_32s_C1IR	N/A
ippiPCTInv8x16_JPEGXR_16s_C1IR	N/A
ippiPCTInv8x16_JPEGXR_32f_C1IR	N/A
ippiPCTInv8x16_JPEGXR_32s_C1IR	N/A
ippiPCTInv8x8_JPEGXR_16s_C1IR	N/A
ippiPCTInv8x8_JPEGXR_32f_C1IR	N/A
ippiPCTInv8x8_JPEGXR_32s_C1IR	N/A
ippiPCTInv_JPEGXR_16s_C1IR	N/A
ippiPCTInv_JPEGXR_32f_C1IR	N/A
ippiPCTInv_JPEGXR_32s_C1IR	N/A
ippiPackBitsRow_TIFF_8u_C1	N/A
ippiQuantFwd8x8_JPEG_16s_C1I	N/A
ippiQuantFwdRawTableInit_JPEG_8u	N/A
ippiQuantFwdTableInit_JPEG_8u16u	N/A
ippiQuantInv8x8_JPEG_16s_C1I	N/A
ippiQuantInvTableInit_JPEG_8u16u	N/A
ippiRCTFwd_JPEG2K_16s_C3P3R	N/A
ippiRCTFwd_JPEG2K_16s_P3IR	N/A
ippiRCTFwd_JPEG2K_32s_C3P3R	N/A
ippiRCTFwd_JPEG2K_32s_P3IR	N/A
ippiRCTInv_JPEG2K_16s_P3C3R	N/A

Removed from 9.0	Substitution or Workaround
ippiRCTInv_JPEG2K_16s_P3IR	N/A
ippiRCTInv_JPEG2K_32s_P3C3R	N/A
ippiRCTInv_JPEG2K_32s_P3IR	N/A
ippiRGB555ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiRGB565ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiRGBToYCbCr411LS MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr411_JPEG_8u_C3P3R	N/A
ippiRGBToYCbCr411_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr422LS MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr422_JPEG_8u_C3P3R	N/A
ippiRGBToYCbCr422_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr444LS MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr_JPEG_8u_C3P3R	N/A
ippiRGBToYCbCr_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr_JPEG_8u_P3R	N/A
ippiRGBToY_JPEG_8u_C3C1R	N/A
ippiRGBToY_JPEG_8u_P3C1R	N/A
ippiReconstructPredFirstRow_JPEG_16s_C1	N/A
ippiReconstructPredRow_JPEG_16s_C1	N/A
ippiSampleDown411LS MCU_8u16s_C3P3R	N/A
ippiSampleDown422LS MCU_8u16s_C3P3R	N/A
ippiSampleDown444LS MCU_8u16s_C3P3R	N/A
ippiSampleDownH2V1_JPEG_8u_C1R	N/A
ippiSampleDownH2V2_JPEG_8u_C1R	N/A
ippiSampleDownRowH2V1_Box_JPEG_8u_C1	N/A
ippiSampleDownRowH2V2_Box_JPEG_8u_C1	N/A
ippiSampleUp411LS MCU_16s8u_P3C3R	N/A
ippiSampleUp422LS MCU_16s8u_P3C3R	N/A
ippiSampleUp444LS MCU_16s8u_P3C3R	N/A
ippiSampleUpH2V1_JPEG_8u_C1R	N/A
ippiSampleUpH2V2_JPEG_8u_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1	N/A
ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1	N/A
ippiSplit422LS MCU_8u16s_C2P3R	N/A
ippiSplitRow_TIFF_16u8u_C1	N/A
ippiSub128_JPEG_8u16s_C1R	N/A
ippiUnpackBitsRow_TIFF_8u_C1	N/A
ippiVLCAadapt_JPEGXR	N/A
ippiVLDecode4x4_JPEGXR_1u16s_C1R	N/A
ippiVLDecode4x4_JPEGXR_1u32s_C1R	N/A
ippiVLDecodeDC420_JPEGXR_1u32s	N/A
ippiVLDecodeDC422_JPEGXR_1u32s	N/A
ippiVLDecodeDC444_JPEGXR_1u32s	N/A
ippiVLCEncode4x4Flex_JPEGXR_16s1u_C1R	N/A
ippiVLCEncode4x4Flex_JPEGXR_32s1u_C1R	N/A
ippiVLCEncode4x4_JPEGXR_16s1u_C1R	N/A
ippiVLCEncode4x4_JPEGXR_32s1u_C1R	N/A
ippiVLCEncodeDC420_JPEGXR_32s1u	N/A
ippiVLCEncodeDC422_JPEGXR_32s1u	N/A
ippiVLCEncodeDC444_JPEGXR_32s1u	N/A
ippiVLGetStateSize_JPEGXR	N/A
ippiVLInit_JPEGXR	N/A
ippiVLScan4x4DC_JPEGXR_32s	N/A
ippiVLScanReset_JPEGXR	N/A
ippiVLScanSet_JPEGXR	N/A
ippiWTFwdColLift_B53_JPEG2K_16s_C1	N/A
ippiWTFwdColLift_B53_JPEG2K_32s_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_16s_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_32f_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_32s_C1	N/A
ippiWTFwdCol_B53_JPEG2K_16s_C1R	N/A
ippiWTFwdCol_B53_JPEG2K_32s_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiWTFwdCol_D97_JPEG2K_32f_C1R	N/A
ippiWTFwdRow_B53_JPEG2K_16s_C1R	N/A
ippiWTFwdRow_B53_JPEG2K_32s_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_16s_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_32f_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_32s_C1R	N/A
ippiWTFwd_B53_JPEG2K_16s_C1IR	N/A
ippiWTFwd_B53_JPEG2K_16s_C1R	N/A
ippiWTFwd_B53_JPEG2K_32s_C1IR	N/A
ippiWTFwd_B53_JPEG2K_32s_C1R	N/A
ippiWTFwd_D97_JPEG2K_16s_C1IR	N/A
ippiWTFwd_D97_JPEG2K_16s_C1R	N/A
ippiWTFwd_D97_JPEG2K_32s_C1IR	N/A
ippiWTFwd_D97_JPEG2K_32s_C1R	N/A
ippiWTGetSize_B53_JPEG2K_16s_C1IR	N/A
ippiWTGetSize_B53_JPEG2K_16s_C1R	N/A
ippiWTGetSize_B53_JPEG2K_32s_C1IR	N/A
ippiWTGetSize_B53_JPEG2K_32s_C1R	N/A
ippiWTGetSize_D97_JPEG2K_16s_C1IR	N/A
ippiWTGetSize_D97_JPEG2K_16s_C1R	N/A
ippiWTGetSize_D97_JPEG2K_32s_C1IR	N/A
ippiWTGetSize_D97_JPEG2K_32s_C1R	N/A
ippiWTInvCollift_B53_JPEG2K_16s_C1	N/A
ippiWTInvCollift_B53_JPEG2K_32s_C1	N/A
ippiWTInvCollift_D97_JPEG2K_16s_C1	N/A
ippiWTInvCollift_D97_JPEG2K_32f_C1	N/A
ippiWTInvCollift_D97_JPEG2K_32s_C1	N/A
ippiWTInvCol_B53_JPEG2K_16s_C1R	N/A
ippiWTInvCol_B53_JPEG2K_32s_C1R	N/A
ippiWTInvCol_D97_JPEG2K_32f_C1R	N/A
ippiWTInvRow_B53_JPEG2K_16s_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiWTInvRow_B53_JPEG2K_32s_C1R	N/A
ippiWTInvRow_D97_JPEG2K_16s_C1R	N/A
ippiWTInvRow_D97_JPEG2K_32f_C1R	N/A
ippiWTInvRow_D97_JPEG2K_32s_C1R	N/A
ippiWTInv_B53_JPEG2K_16s_C1IR	N/A
ippiWTInv_B53_JPEG2K_16s_C1R	N/A
ippiWTInv_B53_JPEG2K_32s_C1IR	N/A
ippiWTInv_B53_JPEG2K_32s_C1R	N/A
ippiWTInv_D97_JPEG2K_16s_C1IR	N/A
ippiWTInv_D97_JPEG2K_16s_C1R	N/A
ippiWTInv_D97_JPEG2K_32s_C1IR	N/A
ippiWTInv_D97_JPEG2K_32s_C1R	N/A
ippiYCK411ToCMYKLS MCU_16s8u_P4C4R	N/A
ippiYCK422ToCMYKLS MCU_16s8u_P4C4R	N/A
ippiYCK444ToCMYKLS MCU_16s8u_P4C4R	N/A
ippiYCKToCMYK_JPEG_8u_P4C4R	N/A
ippiYCKToCMYK_JPEG_8u_P4R	N/A
ippiYCbCr411ToBGR555LS MCU_16s16u_P3C3R	N/A
ippiYCbCr411ToBGR565LS MCU_16s16u_P3C3R	N/A
ippiYCbCr411ToBGRLS MCU_16s8u_P3C3R	N/A
ippiYCbCr411ToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCr411ToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCr411ToRGBLS MCU_16s8u_P3C3R	N/A
ippiYCbCr411ToRGB_JPEG_8u_P3C3R	N/A
ippiYCbCr411ToRGB_JPEG_8u_P3C4R	N/A
ippiYCbCr422ToBGR555LS MCU_16s16u_P3C3R	N/A
ippiYCbCr422ToBGR565LS MCU_16s16u_P3C3R	N/A
ippiYCbCr422ToBGRLS MCU_16s8u_P3C3R	N/A
ippiYCbCr422ToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCr422ToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCr422ToRGBLS MCU_16s8u_P3C3R	N/A

Removed from 9.0	Substitution or Workaround
ippiYCbCr422ToRGB_JPEG_8u_C2C3R	N/A
ippiYCbCr422ToRGB_JPEG_8u_P3C3R	N/A
ippiYCbCr422ToRGB_JPEG_8u_P3C4R	N/A
ippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr444ToBGRLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr444ToRGBOBS_MCU_16s8u_P3C3R	N/A
ippiYCbCrToBGR555_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToBGR565_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCrToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCrToRGBOBS_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToRGBOBS_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToRGBOBS_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToRGBOBS_JPEG_8u_P3C3R	N/A
ippiYCbCrToRGBOBS_JPEG_8u_P3C4R	N/A
ippiYCbCrToRGBOBS_JPEG_8u_P3R	N/A
ippjGetLibVersion	N/A

**ippvc.h**

Removed from 9.0	Substitution or Workaround
ippiAdd8x8HP_16s8u_C1RS	N/A
ippiAdd8x8_16s8u_C1IRS	N/A
ippiAddBackPredPB_H263_8u_C1R	N/A
ippiAddC8x8_16s8u_C1IR	N/A
ippiAverage16x16_8u_C1IR	N/A
ippiAverage16x16_8u_C1R	N/A
ippiAverage8x8_8u_C1IR	N/A
ippiAverage8x8_8u_C1R	N/A
ippiBiDirWeightBlockImplicit_H264_8u_P2P1R	N/A
ippiBiDirWeightBlockImplicit_H264_8u_P3P1R	N/A
ippiBiDirWeightBlock_H264_8u_C2R	N/A

Removed from 9.0	Substitution or Workaround
ippiBiDirWeightBlock_H264_8u_P2P1R	N/A
ippiBiDirWeightBlock_H264_8u_P3P1R	N/A
ippiBidirWeightImplicit_H264_16u_P2P1R	N/A
ippiBidirWeight_H264_16u_P2P1R	N/A
ippiBidir_H264_16u_P2P1R	N/A
ippiCABACEncodeBinBypass_H264	N/A
ippiCABACEncodeBin_H264	N/A
ippiCABACEncodeResidualBlock_H264_16s	N/A
ippiCABACEncodeResidualBlock_H264_32s	N/A
ippiCABACFree_H264	N/A
ippiCABACGetContexts_H264	N/A
ippiCABACGetSize_H264	N/A
ippiCABACGetStreamSize_H264	N/A
ippiCABACInitAlloc_H264	N/A
ippiCABACInit_H264	N/A
ippiCABACSetStream_H264	N/A
ippiCABACTerminateSlice_H264	N/A
ippiCalcGlobalMV_MPEG4	N/A
ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R	N/A
ippiCbYCr422ToYCbCr420_Rotate_8u_P3R	N/A
ippiChangeSpriteBrightness_MPEG4_8u_C1R	N/A
ippiCopy16x16HP_8u_C1R	N/A
ippiCopy16x16QP_MPEG4_8u_C1R	N/A
ippiCopy16x16_8u_C1R	N/A
ippiCopy16x8HP_8u_C1R	N/A
ippiCopy16x8QP_MPEG4_8u_C1R	N/A
ippiCopy8x4HP_8u_C1R	N/A
ippiCopy8x8HP_8u_C1R	N/A
ippiCopy8x8QP_MPEG4_8u_C1R	N/A
ippiCopy8x8_8u_C1R	N/A
ippiCountZeros8x8_16s_C1	N/A

Removed from 9.0	Substitution or Workaround
ippiCreateRLEncodeTable	N/A
ippiDCT2x4x8Frw_16s_C1I	N/A
ippiDCT2x4x8Inv_16s_C1I	N/A
ippiDCT8x4x2To4x4Inv_DV_16s_C1I	N/A
ippiDCT8x8Fwd_8u16s_C2P2	N/A
ippiDCT8x8InvOrSet_16s8u_P2C2	N/A
ippiDCT8x8Inv_AANTransposed_16s8u_C1R	N/A
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R	N/A
ippiDCT8x8Inv_AANTransposed_16s_C1R	N/A
ippiDCT8x8Inv_AANTransposed_16s_P2C2R	N/A
ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s	N/A
ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s	N/A
ippiDecodeCAVLCCoeffsIdxs_H264_1u16s	N/A
ippiDecodeCAVLCCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCCoeffs_H264_1u32s	N/A
ippiDecodeChromaBlock_AVs_1u16s	N/A
ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s	N/A
ippiDecodeCoeffsInter_H261_1u16s	N/A
ippiDecodeCoeffsInter_H263_1u16s	N/A
ippiDecodeCoeffsInter_MPEG4_1u16s	N/A
ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s	N/A
ippiDecodeCoeffsIntra_H261_1u16s	N/A
ippiDecodeCoeffsIntra_H263_1u16s	N/A
ippiDecodeCoeffsIntra_MPEG4_1u16s	N/A
ippiDecodeDCIntra_H263_1u16s	N/A
ippiDecodeDCIntra_MPEG4_1u16s	N/A
ippiDecodeExpGolombOne_H264_1u16s	N/A
ippiDecodeExpGolombOne_H264_1u32s	N/A
ippiDecodeHuffmanOne_1u32s	N/A

Removed from 9.0	Substitution or Workaround
ippiDecodeHuffmanPair_1u16s	N/A
ippiDecodeLumaBlockInter_AVs_1u16s	N/A
ippiDecodeLumaBlockIntra_AVs_1u16s	N/A
ippiDeinterlaceBlendFree_8u_C1	N/A
ippiDeinterlaceBlendFree_8u_C2	N/A
ippiDeinterlaceBlendGetSize_8u_C1	N/A
ippiDeinterlaceBlendGetSize_8u_C2	N/A
ippiDeinterlaceBlendInitAlloc_8u_C1	N/A
ippiDeinterlaceBlendInitAlloc_8u_C2	N/A
ippiDeinterlaceBlendInit_8u_C1	N/A
ippiDeinterlaceBlendInit_8u_C2	N/A
ippiDeinterlaceBlend_8u_C1	N/A
ippiDeinterlaceBlend_8u_C2	N/A
ippiDeinterlaceEdgeDetect_8u_C1R	N/A
ippiDeinterlaceFilterTriangle_8u_C1R	N/A
ippiDeinterlaceFilterTriangle_8u_C2R	N/A
ippiDeinterlaceMedianThreshold_8u_C1R	N/A
ippiDeinterlaceMotionAdaptive_8u_C1	N/A
ippiDequantTransformResidualAndAdd_H264_16s_C1I	N/A
ippiDequantTransformResidual_H264_16s_C1I	N/A
ippiDequantTransformResidual_SISP_H264_16s_C1I	N/A
ippiDisassembleChroma420Intra_AVs_16s8u_C1R	N/A
ippiDisassembleLumaIntra_AVs_16s8u_C1R	N/A
ippiDownsampleFour16x16_H263_16s_C1R	N/A
ippiDownsampleFour_H263_8u_C1R	N/A
ippiEdgesDetect16x16_16u_C1R	N/A
ippiEdgesDetect16x16_8u_C1R	N/A
ippiEncodeChromaDcCoeffsCAVLC_H264_16s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x2_H264_32s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x4_H264_16s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x4_H264_32s	N/A

Removed from 9.0	Substitution or Workaround
ippiEncodeCoeffsCAVLC_H264_16s	N/A
ippiEncodeCoeffsCAVLC_H264_32s	N/A
ippiEncodeCoeffsInter_H261_16s1u	N/A
ippiEncodeCoeffsInter_H263_16s1u	N/A
ippiEncodeCoeffsInter_MPEG4_16s1u	N/A
ippiEncodeCoeffsIntra_H261_16s1u	N/A
ippiEncodeCoeffsIntra_H263_16s1u	N/A
ippiEncodeCoeffsIntra_MPEG4_16s1u	N/A
ippiEncodeDCIntra_H263_16s1u	N/A
ippiEncodeDCIntra_MPEG4_16s1u	N/A
ippiExpandPlane_H264_8u_C1R	N/A
ippiFilter8x8_H261_8u_C1R	N/A
ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR	N/A
ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR	N/A
ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR	N/A
ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChromaHorEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR	N/A
ippiFilterDeblockingChromaVerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C2I	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C2TR	N/A

Removed from 9.0	Substitution or Workaround
ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_VC1_8u_C2IR	N/A
ippiFilterDeblockingChroma_VerEdge_AVSS_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C2IR	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C3IR	N/A
ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_VC1_8u_C2IR	N/A
ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR	N/A
ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingLuma_HorEdge_AVSS_8u_C1IR	N/A
ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_AVSS_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR	N/A
ippiFilterDenoiseAdaptiveFree_8u_C1	N/A
ippiFilterDenoiseAdaptiveInitAlloc_8u_C1	N/A
ippiFilterDenoiseAdaptive_8u_C1R	N/A
ippiFilterDenoiseCASTInit	N/A
ippiFilterDenoiseCASTYUV422_8u_C2R	N/A
ippiFilterDenoiseCAST_8u_C1R	N/A
ippiFilterDenoiseMosquitoFree_8u_C1	N/A
ippiFilterDenoiseMosquitoInitAlloc_8u_C1	N/A
ippiFilterDenoiseMosquito_8u_C1R	N/A
ippiFilterDenoiseSmooth_8u_C1R	N/A
ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R	N/A
ippiFilterDeringingThreshold_MPEG4_8u_P3R	N/A

Removed from 9.0	Substitution or Workaround
ippiFrameFieldSAD16x16_16s32s_C1R	N/A
ippiFrameFieldSAD16x16_8u32s_C1R	N/A
ippiFreeHuffmanTable_DV_32u	N/A
ippiGenScaleLevel4x4_H264_8u16s_C1	N/A
ippiGenScaleLevel8x8_H264_8u16s_D2	N/A
ippiGetDiff16x16B_8u16s_C1	N/A
ippiGetDiff16x16_8u16s_C1	N/A
ippiGetDiff16x8B_8u16s_C1	N/A
ippiGetDiff16x8_8u16s_C1	N/A
ippiGetDiff8x16B_8u16s_C1	N/A
ippiGetDiff8x16_8u16s_C1	N/A
ippiGetDiff8x4B_8u16s_C1	N/A
ippiGetDiff8x4B_8u16s_C2P2	N/A
ippiGetDiff8x4_8u16s_C1	N/A
ippiGetDiff8x4_8u16s_C2P2	N/A
ippiGetDiff8x8B_8u16s_C1	N/A
ippiGetDiff8x8B_8u16s_C2P2	N/A
ippiGetDiff8x8_8u16s_C1	N/A
ippiGetDiff8x8_8u16s_C2P2	N/A
ippiHadamard8x8Sum_VC1_16s	N/A
ippiHadamard8x8Sum_VC1_8u16s	N/A
ippiHuffmanDecodeSegmentOnePass_DV_8u16s	N/A
ippiHuffmanDecodeSegment_DV100_8u16s	N/A
ippiHuffmanDecodeSegment_DV_8u16s	N/A
ippiHuffmanRunLevelTableInitAlloc_32s	N/A
ippiHuffmanTableFree_32s	N/A
ippiHuffmanTableInitAlloc_32s	N/A
ippiInitAllocHuffmanTable_DV_32u	N/A
ippiInterpolateAverage16x16_8u_C1IR	N/A
ippiInterpolateAverage16x8_8u_C1IR	N/A
ippiInterpolateAverage8x4_8u_C1IR	N/A

Removed from 9.0	Substitution or Workaround
ippiInterpolateAverage8x8_8u_C1R	N/A
ippiInterpolateBlock_H264_8u_P2P1R	N/A
ippiInterpolateBlock_H264_8u_P3P1R	N/A
ippiInterpolateChromaBlock_H264_16u_P2R	N/A
ippiInterpolateChromaBlock_H264_8u_C2C2R	N/A
ippiInterpolateChromaBlock_H264_8u_C2P2R	N/A
ippiInterpolateChromaBlock_H264_8u_P2R	N/A
ippiInterpolateChromaBottom_H264_16u_C1R	N/A
ippiInterpolateChromaBottom_H264_8u_C1R	N/A
ippiInterpolateChromaTop_H264_16u_C1R	N/A
ippiInterpolateChromaTop_H264_8u_C1R	N/A
ippiInterpolateChroma_H264_16u_C1R	N/A
ippiInterpolateChroma_H264_8u_C1R	N/A
ippiInterpolateICBicubicBlock_VC1_8u_C1R	N/A
ippiInterpolateICBilinearBlock_VC1_8u_C1R	N/A
ippiInterpolateICBilinearBlock_VC1_8u_C2R	N/A
ippiInterpolateLumaBlock_AVG_8u_P1R	N/A
ippiInterpolateLumaBlock_H264_16u_P1R	N/A
ippiInterpolateLumaBlock_H264_8u_P1R	N/A
ippiInterpolateLumaBottom_H264_16u_C1R	N/A
ippiInterpolateLumaBottom_H264_8u_C1R	N/A
ippiInterpolateLumaTop_H264_16u_C1R	N/A
ippiInterpolateLumaTop_H264_8u_C1R	N/A
ippiInterpolateLuma_H264_16u_C1R	N/A
ippiInterpolateLuma_H264_8u_C1R	N/A
ippiInterpolateQPbicubic_VC1_8u_C1R	N/A
ippiInterpolateQPBilinear_VC1_8u_C1R	N/A
ippiInterpolateQPBilinear_VC1_8u_C2R	N/A
ippiMC16x16B_8u_C1	N/A
ippiMC16x16_8u_C1	N/A

Removed from 9.0	Substitution or Workaround
ippiMC16x4B_8u_C1	N/A
ippiMC16x4_8u_C1	N/A
ippiMC16x8BUV_8u_C1	N/A
ippiMC16x8B_8u_C1	N/A
ippiMC16x8UV_8u_C1	N/A
ippiMC16x8_8u_C1	N/A
ippiMC2x2B_8u_C1	N/A
ippiMC2x2_8u_C1	N/A
ippiMC2x4B_8u_C1	N/A
ippiMC2x4_8u_C1	N/A
ippiMC4x2B_8u_C1	N/A
ippiMC4x2_8u_C1	N/A
ippiMC4x4B_8u_C1	N/A
ippiMC4x4_8u_C1	N/A
ippiMC4x8B_8u_C1	N/A
ippiMC4x8_8u_C1	N/A
ippiMC8x16B_8u_C1	N/A
ippiMC8x16_8u_C1	N/A
ippiMC8x4B_16s8u_P2C2R	N/A
ippiMC8x4B_8u_C1	N/A
ippiMC8x4_16s8u_P2C2R	N/A
ippiMC8x4_8u_C1	N/A
ippiMC8x8B_16s8u_P2C2R	N/A
ippiMC8x8B_8u_C1	N/A
ippiMC8x8_16s8u_P2C2R	N/A
ippiMC8x8_8u_C1	N/A
ippiMeanAbsDev16x16_8u32s_C1R	N/A
ippiMeanAbsDev8x8_8u32s_C1R	N/A
ippiOBMC16x16HP_MPEG4_8u_C1R	N/A
ippiOBMC8x8HP_MPEG4_8u_C1R	N/A
ippiOBMC8x8QP_MPEG4_8u_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiPredictIntraChroma8x8_H264_8u_C1IR	N/A
ippiPredictIntra_16x16_H264_8u_C1IR	N/A
ippiPredictIntra_4x4_H264_8u_C1IR	N/A
ippiPutIntraBlock	N/A
ippiQuantInterGetSize_MPEG4	N/A
ippiQuantInterInit_MPEG4	N/A
ippiQuantInterNonuniform_VC1_16s_C1IR	N/A
ippiQuantInterUniform_VC1_16s_C1IR	N/A
ippiQuantInter_H263_16s_C1I	N/A
ippiQuantInter_MPEG4_16s_C1I	N/A
ippiQuantIntraGetSize_MPEG4	N/A
ippiQuantIntraInit_MPEG4	N/A
ippiQuantIntraNonuniform_VC1_16s_C1IR	N/A
ippiQuantIntraUniform_VC1_16s_C1IR	N/A
ippiQuantIntra_H263_16s_C1I	N/A
ippiQuantIntra_MPEG2_16s_C1I	N/A
ippiQuantIntra_MPEG4_16s_C1I	N/A
ippiQuantInvInterGetSize_MPEG4	N/A
ippiQuantInvInterInit_MPEG4	N/A
ippiQuantInvInterNonuniform_VC1_16s_C1IR	N/A
ippiQuantInvInterNonuniform_VC1_16s_C1R	N/A
ippiQuantInvInterUniform_VC1_16s_C1IR	N/A
ippiQuantInvInterUniform_VC1_16s_C1R	N/A
ippiQuantInvInter_H263_16s_C1I	N/A
ippiQuantInvInter_MPEG4_16s_C1I	N/A
ippiQuantInvIntraGetSize_MPEG4	N/A
ippiQuantInvIntraInit_MPEG4	N/A
ippiQuantInvIntraNonuniform_VC1_16s_C1IR	N/A
ippiQuantInvIntraNonuniform_VC1_16s_C1R	N/A
ippiQuantInvIntraUniform_VC1_16s_C1IR	N/A
ippiQuantInvIntraUniform_VC1_16s_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiQuantInvIntra_H263_16s_C1I	N/A
ippiQuantInvIntra_MPEG2_16s_C1I	N/A
ippiQuantInvIntra_MPEG4_16s_C1I	N/A
ippiQuantInvLuma8x8_H264_32s_C1I	N/A
ippiQuantInv_DV_16s_C1I	N/A
ippiQuantInv_MPEG2_16s_C1I	N/A
ippiQuantLuma8x8Inv_H264_16s_C1I	N/A
ippiQuantLuma8x8_H264_16s_C1	N/A
ippiQuantLuma8x8_H264_32s_C1	N/A
ippiQuantWeightBlockInv_DV100_16s_C1I	N/A
ippiQuantWeightBlockInv_DV_16s_C1I	N/A
ippiQuant_MPEG2_16s_C1I	N/A
ippiQuantizeResidual4x4Fwd_H264_16s32s_C1	N/A
ippiQuantizeResidual4x4Fwd_H264_16s_C1	N/A
ippiQuantizeResidual4x4Fwd_H264_32s_C1	N/A
ippiRangeMapping_VC1_8u_C1R	N/A
ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaInter4x4MB_H264_16s8u_C2R	N/A
ippiReconstructChromaInter4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaInter4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaInterMB_H264_16s8u_C2R	N/A
ippiReconstructChromaInterMB_H264_16s8u_P2R	N/A
ippiReconstructChromaInter_AVs_16s8u_C1R	N/A
ippiReconstructChromaIntra4x4MB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R	N/A

Removed from 9.0	Substitution or Workaround
ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaIntraHalfs4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalfsMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraMB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntraMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntra_AVs_16s8u_C1R	N/A
ippiReconstructCoeffsInter_H261_1u16s	N/A
ippiReconstructCoeffsInter_H263_1u16s	N/A
ippiReconstructCoeffsInter_MPEG4_1u16s	N/A
ippiReconstructCoeffsIntra_H261_1u16s	N/A
ippiReconstructCoeffsIntra_H263_1u16s	N/A
ippiReconstructDCTBlockIntra_MPEG1_32s	N/A
ippiReconstructDCTBlockIntra_MPEG2_32s	N/A
ippiReconstructDCTBlock_MPEG1_32s	N/A
ippiReconstructDCTBlock_MPEG2_32s	N/A
ippiReconstructLumaInter4x4MB_H264_16s8u_C1R	N/A
ippiReconstructLumaInter4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaInter8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaInter8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaInterMB_H264_16s8u_C1R	N/A
ippiReconstructLumaInter_AVs_16s8u_C1R	N/A
ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra16x16_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiReconstructLumaIntraHalf4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntraHalf8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntraMB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra_AVs_16s8u_C1R	N/A
ippiResample_H263_8u_P3R	N/A
ippiResizeCCRotate_8u_C2R	N/A
ippiSAD16x16Blocks4x4_16u32u_C1R	N/A
ippiSAD16x16Blocks4x4_8u16u	N/A
ippiSAD16x16Blocks8x8_16u32u_C1R	N/A
ippiSAD16x16Blocks8x8_8u16u	N/A
ippiSAD16x16_16u32s_C1R	N/A
ippiSAD16x16_8u32s	N/A
ippiSAD16x16xNI_8u16u_C1R	N/A
ippiSAD16x16xN_8u16u_C1R	N/A
ippiSAD16x8_8u32s_C1R	N/A
ippiSAD2x2xN_8u16u_C1R	N/A
ippiSAD4x4_16u32s_C1R	N/A
ippiSAD4x4_8u32s	N/A
ippiSAD4x4xNI_8u16u_C1R	N/A
ippiSAD4x4xN_8u16u_C1R	N/A
ippiSAD4x8_8u32s_C1R	N/A
ippiSAD8x16_8u32s_C1R	N/A
ippiSAD8x4_8u32s_C1R	N/A
ippiSAD8x8_16u32s_C1R	N/A
ippiSAD8x8_8u32s_C1R	N/A
ippiSAD8x8_8u32s_C2R	N/A
ippiSAD8x8xNI_8u16u_C1R	N/A
ippiSAD8x8xN_8u16u_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiSAT8x8D_16u32s_C1R	N/A
ippiSAT8x8D_8u32s_C1R	N/A
ippiSATD16x16_8u32s_C1R	N/A
ippiSATD16x8_8u32s_C1R	N/A
ippiSATD4x4_8u32s_C1R	N/A
ippiSATD4x8_8u32s_C1R	N/A
ippiSATD8x16_8u32s_C1R	N/A
ippiSATD8x4_8u32s_C1R	N/A
ippiSATD8x8_8u32s_C1R	N/A
ippiSSD4x4_8u32s_C1R	N/A
ippiSSD8x8_8u32s_C1R	N/A
ippiScanFwd_16s_C1	N/A
ippiScanInv_16s_C1	N/A
ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R	N/A
ippiSmoothingChroma_HorEdge_VC1_16s8u_P2C2R	N/A
ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R	N/A
ippiSmoothingChroma_VerEdge_VC1_16s8u_P2C2R	N/A
ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R	N/A
ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R	N/A
ippiSpatialInterpolation_H263_8u_C1R	N/A
ippiSqrDiff16x16B_8u32s	N/A
ippiSqrDiff16x16_8u32s	N/A
ippiSub16x16_8u16s_C1R	N/A
ippiSub4x4_16u16s_C1R	N/A
ippiSub4x4_8u16s_C1R	N/A
ippiSub8x8_16u16s_C1R	N/A
ippiSub8x8_8u16s_C1R	N/A
ippiSubSAD8x8_8u16s_C1R	N/A
ippiSumsDiff16x16Blocks4x4_16u32s_C1R	N/A
ippiSumsDiff16x16Blocks4x4_8u16s_C1	N/A
ippiSumsDiff8x8Blocks4x4_16u32s_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiSumsDiff8x8Blocks4x4_8u16s_C1	N/A
ippiSumsDiff8x8Blocks4x4_8u16s_C2P2	N/A
ippiTransform4x4Fwd_VC1_16s_C1IR	N/A
ippiTransform4x4Fwd_VC1_16s_C1R	N/A
ippiTransform4x4Inv_VC1_16s_C1IR	N/A
ippiTransform4x4Inv_VC1_16s_C1R	N/A
ippiTransform4x8Fwd_VC1_16s_C1IR	N/A
ippiTransform4x8Fwd_VC1_16s_C1R	N/A
ippiTransform4x8Inv_VC1_16s_C1IR	N/A
ippiTransform4x8Inv_VC1_16s_C1R	N/A
ippiTransform8x4Fwd_VC1_16s_C1IR	N/A
ippiTransform8x4Fwd_VC1_16s_C1R	N/A
ippiTransform8x4Inv_VC1_16s_C1IR	N/A
ippiTransform8x4Inv_VC1_16s_C1R	N/A
ippiTransform8x8Fwd_VC1_16s_C1IR	N/A
ippiTransform8x8Fwd_VC1_16s_C1R	N/A
ippiTransform8x8Inv_VC1_16s_C1IR	N/A
ippiTransform8x8Inv_VC1_16s_C1R	N/A
ippiTransformDequantChromaDC_H264_16s_C1I	N/A
ippiTransformDequantChromaDC_SISP_H264_16s_C1I	N/A
ippiTransformDequantLumaDC_H264_16s_C1I	N/A
ippiTransformFwdLuma8x8_H264_16s32s_C1	N/A
ippiTransformFwdLuma8x8_H264_16s_C1	N/A
ippiTransformInvAddPredLuma8x8_H264_32s16u_C1P	N/A
ippiTransformLuma8x8Fwd_H264_16s_C1I	N/A
ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R	N/A
ippiTransformPrediction_H264_8u16s_C1	N/A
ippiTransformQuant8x8Fwd_AV8_16s_C1	N/A
ippiTransformQuantChromaDC_H264_16s_C1I	N/A
ippiTransformQuantFwd4x4_H264_16s32s_C1	N/A
ippiTransformQuantFwd4x4_H264_16s_C1	N/A

Removed from 9.0	Substitution or Workaround
ippiTransformQuantFwdChromaDC2x2_H264_16s_C1T	N/A
ippiTransformQuantFwdChromaDC2x2_H264_32s_C1T	N/A
ippiTransformQuantFwdChromaDC2x4_H264_16s_C1T	N/A
ippiTransformQuantFwdChromaDC2x4_H264_32s_C1T	N/A
ippiTransformQuantFwdLumaDC4x4_H264_16s_C1I	N/A
ippiTransformQuantFwdLumaDC4x4_H264_32s_C1I	N/A
ippiTransformQuantInvAddPred4x4_H264_16s_C1IR	N/A
ippiTransformQuantInvAddPred4x4_H264_32s_C1IR	N/A
ippiTransformQuantInvChromaDC2x2_H264_16s_C1T	N/A
ippiTransformQuantInvChromaDC2x2_H264_32s_C1T	N/A
ippiTransformQuantInvChromaDC2x4_H264_16s_C1T	N/A
ippiTransformQuantInvChromaDC2x4_H264_32s_C1T	N/A
ippiTransformQuantInvLumaDC4x4_H264_16s_C1I	N/A
ippiTransformQuantInvLumaDC4x4_H264_32s_C1I	N/A
ippiTransformQuantLumaDC_H264_16s_C1I	N/A
ippiTransformQuantResidual_H264_16s_C1I	N/A
ippiTransformResidual4x4Fwd_H264_16s_C1	N/A
ippiTransformResidual4x4Fwd_H264_32s_C1	N/A
ippiTransformResidual4x4Inv_H264_16s_C1	N/A
ippiTransformResidual4x4Inv_H264_32s_C1	N/A
ippiUniDirWeightBlock_H264_8u_C1IR	N/A
ippiUniDirWeightBlock_H264_8u_C1R	N/A
ippiUniDirWeightBlock_H264_8u_C2R	N/A
ippiUnidirWeight_H264_16u_IP2P1R	N/A
ippiUpsampleFour8x8_H263_16s_C1R	N/A
ippiUpsampleFour_H263_8u_C1R	N/A
ippiVarMean8x8_16s32s_C1R	N/A
ippiVarMean8x8_8u32s_C1R	N/A
ippiVarMeanDiff16x16_8u32s_C1R	N/A
ippiVarMeanDiff16x8_8u32s_C1R	N/A
ippiVarSum8x8_16s32s_C1R	N/A

Removed from 9.0	Substitution or Workaround
ippiVarSum8x8_8u32s_C1R	N/A
ippiVariance16x16_8u32s	N/A
ippiWarpChroma_MPEG4_8u_P2R	N/A
ippiWarpGetSize_MPEG4	N/A
ippiWarpInit_MPEG4	N/A
ippiWarpLuma_MPEG4_8u_C1R	N/A
ippiWeightPrediction_AVG_8u_C1R	N/A
ippiWeightedAverage_H264_8u_C1IR	N/A
ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_8x8x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_8x4x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippvcGetLibVersion	N/A



# Bibliography for Image Processing

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rog85], [Rog90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.

- [Aka96] A.Akansu, M.Smith (editors). *Subband and Wavelet transform. Design and Applications*, Kluwer Academic Publishers, 1996.
- [AP922] *A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Intel® Streaming SIMD Extensions and MMX™ Instructions*, Application Note AP922, Intel Corp. Order number 742474, 1999.
- [APMF] *Fast Algorithms for Median Filtering*, Application Note, Intel Corp. Document number 79835, 2001.
- [AVS] GB/T 200090.2-2006. China Standard. Information Technology. *Coding of Audio-Visual Objects - Part 2: Visual* (02/2006).
- [Bert01] M.Bertalmio, A.L.Bertozzi, G.Sapiro. *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting*. Proc. ICCV 2001, pp.1335-1362, 2001.
- [Bor86] G.Borgefors. *Distance Transformations in Digital Images*. Computer Vision, Graphics, and Image Processing 34, 1986.
- [Bou99] J-Y.Bouguet. *Pyramidal Implementation of the Lucas-Kanade Feature Tracker*. OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.
- [Canny86] J. Canny. *A Computational Approach to Edge Detection*, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.
- [Davis97] J.Davis and Bobick. *The Representation and Recognition of Action Using Temporal Templates*. MIT Media Lab Technical Report 402, 1997.
- [Davis99] J.Davis and G.Bradski. *Real-Time Motion Template Gradients Using Intel(R) Computer Vision Library*. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.
- [Dalal05] N. Dalal and B. Triggs. *Histograms of Oriented Gradients for Human Detection*. INRIA, 2005.
- [DICOM] Digital Imaging and Communications in Medicine (DICOM), published by National Electrical Manufacturers Association, 2003.
- [Feig92] E. Feig and S. Winograd. *Fast Algorithms for the Discrete Cosine Transform*, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.
- [Felz04] P. Felzenszwalb, D. Hattenlocher. *Distance Transforms of Sampled Functions*. Cornell Computing and Information Science Technical Report TR2004-1963, September 2004.
- [Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics — Principles and Practice*, Second Edition. Addison Wesley, 1990.
- [Gon93] R.C. Gonzalez and R.E.Wood. *Digital Image Processing*. Prentice Hall, 1993.
- [Hir05] K. Hirakawa, T.W. Parks, *Adaptive Homogeneity-Directed Demosaicing Algorithm*, IEEE Trans. Image Processing, March, 2005.

- [IEC61834] IEC 61834. International Electrochemical Commission. *Specifications of Consumer-Use Digital VCRs using 6.3 mm magnetic tape (the "Blue Book" DV specification)*.
- [IEEE] IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform, IEEE #1180 (1997).
- [IPL] Intel Image Processing Library Reference Manual. Intel Corp. Order number 663791, 1999.
- [ISO11172] ISOCD 11172. Information Technology. Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s (1993).
- [ISO13818] ISO/IEC 13818. Information Technology. Coding of Moving Pictures and Associated Audio Information, (11/94).
- [ISO14496] International Standard ISO/IEC 14496-2. Information Technology. Coding of Audio-Visual Objects - Part 2: Visual.
- [ISO14496A] ISO/IEC 14496-2:1999/Amd.1:2000(E) . Information Technology. Coding of Audio-Visual Objects. Part2:Visual. Amendment 1: Visual Extensions (01/00).
- [ISO10918] International Standard ISO/IEC 10918-1, *Digital Compression and Coding of Continuous Tone Still Images*, Appendix A - Requirements and guidelines.
- [ISO15444] International Standard ISO/IEC 15444-1, *JPEG 2000 Image coding system*, part 1: Core coding system.
- [ISO29199] International Standard ISO/IEC 29199-2, *JPEG XR Image coding system*, part 2: Image coding specification. (2009-08-15)
- [ITUH261] ITU-T Recommendation H.261. *Line transmission of non-telephone signals. Video codec for audiovisual services at p x 64 kbits* (03/93).
- [ITUH263] ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication* (02/98).
- [ITUH264] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2005)(03/05).
- [ITUH264\_07] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2008)(11/07).
- [ITU709] ITU-R Recommendation BT.709, *Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange* [formerly CCIR Rec.709] ITU, Geneva, Switzerland, 1990.
- [Jae95] Jaehne, Bernd. *Digital Image Processing*, 3rd Edition, Springer-Verlag, Berlin, 1995.
- [Jae97] Jaehne, Bernd. *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, New York, 1997.
- [Jack01] Jack, Keith. *Video Demystified: a Handbook for the Digital Engineer*, LLH Technology Publishing, 3rd Edition, 2001.
- [JVTG050] JVT-G050. *ITU-T Recommendation and Final Draft International Standard of Joint Video Specification* (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC) (03/03).

- [Kadew01] P.KadewTraKuPong, R.Bowden. *An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection*. Proc. 2nd European Workshop on Advanced Video-Based Surveillance Systems, 2001.
- [Lein02] R.Leinhart, J.Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP, vol.1, pp. 900-903, Sep. 2002.
- [Li03] L.Li, W.Huang, I.Gu, Q.Tian. *Foreground Object Detection from Videos Containing Complex Background*. Proc.ACM Multimedia Conference, Berkley, 2003.
- [Lim90] Jae S.Lim. *Two-Dimensional Signal and Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1990.
- [Lotufo00] R.Lotufo, A.Falcao. *The Ordered Queue and the Optimality of the Watershed Algorithm*. Mathematical Morphology and its Applications to Image and Signal Processing, vol.18, pp.341-350. Kluwer Academic Publishers. Palo Alto, USA, June 2000.
- [Lowe04] D.G.Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol.60, No.2, pp. 91-110, 2004.
- [Malvar03] H.S.Malvar, G.J.Sullivan. *Transform, Scaling & Color Space Impact of Professional Extensions*, ISO/IEC JTC/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031, Geneva, May 2003.
- [Matas00] J.Matas, C.Galambos, J.V.Kittler. *Robust Detection of Lines Using the Progressive Probabilistic Hough Transform*, CVIU 78 1, pp. 119-137, 2000.
- [Malvar03-1] H.S.Malvar, G.J.Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No.JVT-1014r3, July 2003.
- [Meyer92] F.Meyer. *Color Image Segmentation*, 4th International Conference on Image Processing and its Applications, p.4, Maastricht, April 1993.
- [Meyer94] F.Meyer. *Topographic Distance and Watershed Lines*. Signal Processing, No.38, pp.113-125, 1994.
- [Mitchell88] Don P. Mitchell, Arun N. Netravali. *Reconstruction Filters in Computer Graphics*. Computer Graphics, Volume 22, Number 4, AT&T Bell Laboratories, Murray Hill, New Jersey, August 1988.
- [Myler93] H.Myler, A.Weeks. *Computer Imaging Recipes in C*, Prentice Hall, 1993.
- [Otsu79] N. Otsu. *A Threshold Selection Method From Gray Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, vol.9, No.1, January 1979, pp. 62-66.
- [Puettner05] R.C.Puettner, T.R.Gosnell, and Amos Yahil. *Digital Image Reconstruction: Deblurring and Denoising*, Annual Review of Astronomy and Astrophysics, 2005.
- [Randy97] Randy Crane. *A Simplified Approach to Image Processing*, Prentice Hall PTR, 1997.
- [Rao90]] K.R. Rao and P. Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications*. Academic Press, Inc, London, 1990.
- [Ric72] W.Richardson. *Bayesian-Based Iterative Method of Image Reconstruction*. Journal of the Optical Society of America, vol.62, No.1, January 1972.
- [Ritter96] G.Ritter, J.Wilson. *Computer Vision. Algorithms in Image Algebra*. CRC Press, 1996.
- [Rog85] David Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

- [Rog90] David Rogers and J.Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1990.
- [S3TC] *S3 Texture Compression*. <http://en.wikipedia.org/wiki/S3TC>
- [Sak98] T. Sakamoto, C. Nakanishi, and T. Hase, *Software pixel interpolation for digital still cameras suitable for a 32-bit MCU*, IEEE Trans. Consumer Electronics, vol. 44, No. 4, November 1998.
- [Serra82] J.Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.
- [Schu92] Dale A. Schumacher. *General Filtered Image Rescaling*, Graphic Gems III, Academic Press, 1992.
- [Schu94] Dale A. Schumacher. *A comparison of digital halftoning techniques*, Graphic Gems III, Academic Press, 1994, pp. 57-71.
- [Sha98] Tom Shanley. *Pentium Pro and Pentium II System Architecture*. Addison-Wesley, 1998.
- [SMPTE314M] SMPTE 314M-2005 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s*. The Society of Motion Picture and Television Engineers (09/05).
- [SMPTE370M] SMPTE 370M-2002 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p*. The Society of Motion Picture and Television Engineers (07/02).
- [SMPTE370M-06] SMPTE 370M-2006 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p, 720/50p*. The Society of Motion Picture and Television Engineers (04/06).
- [SMPTE421M] SMPTE 421M. Final Draft SMPTE Standard - *VC-1 Compressed Video Bitstream Format and Decoding Process* (01/06).
- [Telea04] A.Telea. *An Image Inprinting Technique Based on the Fast Marching Method*. Journal of Graphic Tools, vol.9, No.1, ACM Press, 2004.
- [Tho91] Spencer W. Thomas and Rod G. Bogart. *Color dithering*, Graphic Gems II, Academic Press, 1991, pp. 72-77.
- [Ulichney93] R.Ulichney. *Digital halftoning*. MIT press, 1993.
- [Vincent91] L.Vincent, P.Soille. *Watershed in Digital Spaces: An Efficient Algorithm Based on Immersion Simulation*. IEEE Transactions of Pattern Analysis and Machine Intelligence, vol.3, No.6, June 1991, pp.583-598.
- [Vincent93] L.Vincent. *Morphological Gray Scale Reconstruction in Image Analysis: Applications and Efficient Algorithms*. IEEE Transactions on Image Processing, vol.2, No.2, April 1993.
- [Viola01] P.Viola, M.J.Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.
- [Wang02] Z.Wang, A.C.Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol.9, No.3, March 2002, pp.81-84.
- [Wolberg96] G.Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1996.
- [Zivkovic04] Z.Zivkovic. *Improved Adaptive Gaussian Mixture Model for Background Subtraction*. International Conference Pattern Recognition, UK, August, 2004.



# Glossary

absolute colors	Colors specified by each pixel's coordinates in a color space. Intel Integrated Performance Primitives for image processing use images with absolute colors.
alpha channel	A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.
arithmetic operation	An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.
color-twist matrix	A matrix used to multiply the pixel components in one color space for determining the components in another color space.
DCT	Acronym for the discrete cosine transform. See <a href="#">Discrete Cosine Transforms</a> in Chapter 10 of this document.
dilation	A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.
dyadic operation	An operation that has two input images. It can have other input parameters as well.
element-wise operation	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.
erosion	A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.
four-channel model	A color model that uses four color channels; for example, the RGBA color model.
gray scale image	An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.
in-place operation	An operation whose output image is one of the input images.
linear filtering	In this document, 2D convolution operations.
linear image transforms	In this document, the discrete cosine transform (DCT).
MMX™ technology	An enhancement to the Intel® architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
monadic operation	An operation that has a single input image. It can have other input parameters as well.
morphological operation	An erosion, dilation, or their combinations.
not-in-place operation	An operation whose output is an image other than the input image(s). See in-place operation.
pixel depth	The number of bits determining each channel intensity for a single pixel in the image.

pixel-oriented ordering	Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBRGB.... .
planar-oriented ordering	Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate “plane” for each channel; for example, RRRRRGGGGGBBBB....
region of interest	A rectangular image region on which an operation acts (or processing occurs).
RGB	Red-green-blue. A three-channel color model that uses red, green, and blue color channels.
RGBA	Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.
ROI	See identity matrix.
row-major order	The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array $a[3][4]$ , the element $a[1][0]$ immediately follows $a[0][3]$ .
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than $7FFFh$ saturates to $7FFFh$ . When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than $8000h$ saturates to $8000h$ .
Intel® Streaming SIMD Extensions	The enhancement to the Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
three-channel model	A color model that uses three color channels; for example, the RGB color model.