/R=*roiSpec* Specifies a region of interest (ROI). The ROI is defined by a wave of type unsigned byte (/b/u). The ROI wave must have the same number of rows and columns as the image wave. The ROI itself is defined by the entries/pixels whose values are 0. Pixels outside the ROI can take any nonzero value. The ROI does not have to be contiguous and can take any arbitrary shape. See **ImageGenerateROIMask** for more information on creating ROI waves.

In general, the *roiSpec* has the form {*roiWaveName*, *roiFlag*}, where *roiFlag* can take the following values:

roiFlag=0: Set pixels outside the ROI to 0.

roiFlag=1: Set pixels outside the ROI as in original image.

roiFlag=2: Set pixels outside the ROI to NaN (=64).

By default *roiFlag* is set to 1 and it is then possible to use the /R flag using the abbreviated form /R=*roiWave*.

/T=*thresh* Sets the threshold value.

/W= *Twave* Sets the threshold intervals. Each interval is specified by a pair of values in the wave *Twave*. The first element in each pair is the low value and the second element is the high value. Pixel values that lie outside all the specified intervals are set to 0.

References

The automatic thresholding method (/M=1) is described in: T. W. Ridler and S. Calvard, *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8, 630-632, 1978.

The thresholding method used with /M=6 is described in: A.Z. Arifin and A Asano, "Image segmentation by histogram thresholding using hierarchical cluster analysis", *Pattern Recognition Letters* 27 (2006) 1515-1521.

See Also

For usage examples see **Threshold Examples** on page III-356. The **ImageGenerateROIMask** and **ImageEdgeDetection** operations.

ImageTransform

ImageTransform [***flags***] ***Method imageMatrix***

The ImageTransform operation performs one of a number of transformations on *imageMatrix*. The result of using most keywords is a new wave stored in the current data folder. Most flags in this operation are exclusive to the keywords in which they are mentioned.

Parameters

Method selects type of transform. It is one of the following names:

averageImage Computes an average image for a stack of images contained in 3D *imageMatrix*. The average image is stored in the wave M_AveImage and the standard deviation for each pixel is stored in the wave M_StdvImage in the current data folder. You can use this keyword together with the optional /R flag where a region of interest is defined by zero value points in a ROI wave. The operation sets to NaN the entries in M_AveImage and M_StdvImage that correspond to pixels outside the ROI. *imageMatrix* must have at least three layers.

averageRGBImages Computes an average image from a sequence of RGB images represented by a 4D wave. The average is computed separately for the R, G and B channels and the resulting RGB image is stored in the wave M_AverageRGBImage in the current data folder. The operation supports all real data types.

Added in Igor Pro 7.00.

backProjection	Reconstructs the source image from a projection slice and stores the result in the wave M_BackProjection. The projection slice should either be a wave produced by the projectionSlice keyword of this operation or a wave that would be similarly scaled. The input must be a single or double precision real 2D wave. Row scaling must range symmetrically about zero. For example, if the reconstructed image is expected to have 256 rows then the row scaling of the input should be from -128 to 127. Similarly, the column scaling of the input should range between zero and π. An equivalent implementation as a user function is provided in the demo experiment. You can use this implementation as a starting point if you want to develop filtered back projection. See the **projectionSlice** keyword and the RadonTransformDemo experiment. For algorithm details see the chapter "Reconstruction of cross-sections and the projection-slice theorem of computerized tomography" in Born and Wolf, 1999.
ccsubdivision	Performs a Catmull-Clark recursive generation of B-spline surfaces. There are two valid inputs: triangular meshes or quad meshes. Quad meshes are assumed to be in the form of a 3D wave where the first plane contains the X-values, the second the Y-values and the third the Z-values. Triangle meshes are much more complicated to convert into face-edge-vertex arrays so they are less desirable. They are stored in a three column (triplet) wave where the first column corresponds to the X coordinate, the second to the Y coordinate and the third to the Z coordinate. Each triangle is described by three rows in the wave and common vertices have to be repeated so that each sequential three rows in the triplet wave correspond to a single triangle. You can also associate a scalar value with each vertex and it will be suitably interpolated as new vertices are computed and old ones are shifted. In this case the input source wave contains one more column in the case of a triplet wave or one more plane in the case of a quad wave. The scalar value is added everywhere as an additional dimension to the spatial part of any point calculation. You can specify the number of iterations using the /I flag. By default the operation executes a single iteration. The output is saved in a quad wave M_CCBSplines that consists of 4 columns. Each row corresponds to a Quad where the 3 planes contain the X, Y, and Z components. If you are using an optional scalar in the input, the scalar result is stored in the wave M_CCBScalar. In some situations you may encounter spatial degeneracies when two or more parts of the surface meet at common vertices. This leads to one or more error messages in the history area of the command window. You can try to break a degeneracy yourself by adding a small perturbation to the input coordinates or you can use the /PRTF flag which adds a random perturbation on the order of 1e-10 times the extent of the data in each dimension. The perturbation does not affect the scalar.
cmap2rgb	Converts an image and its associated colormap wave (specified using the /C flag) into an RGB image stored in a 3D wave M_RGBOut.
CMYK2RGB	Converts a CMYK image, stored as 4 layer unsigned byte wave, into a 3 layer, standard RGB image wave. The output wave is M_CMYK2RGB that is stored in the current data folder.
compress	Compresses the data in the *imageWave* using a nonlossy algorithm and stores it in the wave W_Compressed in the current data folder. The compressed wave includes all data associated with *imageWave* including its units and wavenote. Use the decompress keyword to recover the original wave. The operation supports all numeric data types. **NOTE**: The compression format for waves greater than 2GB in size was changed in version 6.30B02. If you compressed a wave greater than 2 GB in IGOR64 6.30B01, you will need to decompress it using the same version. You can not decompress it in 6.30B02 or later.
convert2gray	Converts an arbitrary 2D wave into an 8-bit normalized 2D wave. The default output wave name is M_Image2Gray.

decompress	Decompresses a compressed wave. It saves a copy of the decompressed wave under the name W_DeCompressed in the current data folder. **NOTE**: The compression format for waves greater than 2GB in size was changed in version 6.30B02. If you compressed a wave greater than 2 GB in IGOR64 6.30B01, you will need to decompress it using the same version. You can not decompress it in 6.30B02 or later.
distance	Computes the distance transform, also called "distance map", for the input image. Added in Igor Pro 7.00. The distance transform/map is an image where every pixel belonging to the "object" is replaced by the shortest distance of that pixel from the background/boundary. *imageMatrix* must be a 2D wave of type unsigned byte (Make/B/U) where pixels corresponding to the object are set to zero and pixels corresponding to the background are set to 255. Such an image can be obtained, for example, using **ImageThreshold**. The resulting distance map is stored in the wave M_DistanceMap in the current data folder. The operation supports three metrics (Manhattan, Euclidean, Euclidean + scaling) set via the /METR flag.
extractSurface	Extracts values corresponding to a plane that intersects a 3D volume wave (*imageMatrix*). You must specify the extraction parameters using the /X flag. The volume is defined by the wave scaling of the 3D wave. The result, obtained by trilinear interpolation, is stored in the wave M_ExtractedSurface and is of the type NT_FP64. Points in the plane that lie outside the volume are set to NaN.
fht	Performs a Fast Hartley Transform subject to the /T flag. The source wave must be a 2D real matrix with a power of 2 number of rows and columns. Default output is saved in the double-precision wave M_Hartley in the current data folder. If you use the /O flag the result overwrites *imageMatrix* without changing the numeric type. If *imageMatrix* is single-precision float the conversion is straightforward. All other numeric types are scaled. Single- and double-byte types are scaled to the full dynamic range. 32 bit integers are scaled to the range of the equivalent 16 bit types (i.e., unsigned int is scaled to unsigned short range etc.). It does not support wave scaling or NaN entries.
fillImage	Fills a 2D target image wave with data from a 1D image wave (specified using /D). Both waves must be the same data type, and the number of data points in the target wave must match the number of points in the data wave. There are four fill modes that are specified via the /M flag. The operation supports all noncomplex numeric data types.
findLakes	Originally intended to identify lakes in geographical data, this operation creates a mask for a 2D wave for all the contiguous points whose values are close to each other. You can determine the minimum number of pixels within a contiguous region using the /LARA=*minPixels* flag (default is 100). You can determine how close values must be in order to belong to a contiguous region using the /LTOL=*tolerance* flag (default is zero). You can also limit the search region using the /LRCT flag. Use the flag /LTAR=*target* to set the value of the masked regions. By default, the algorithm uses 4-connectivity when looking at adjacent pixels. You can set it to 8-connectivity using the /LCVT flag. The result of the operation is saved in the wave M_LakeFill. It has the same data type as the source wave and contains all the source values outside the masked pixels.
flipCols	Rearrange pixels by exchanging columns symmetrically about a center column or the center of the image (if the number of columns is even). The exchange is performed in place and can be reverted by repeating the operation. When working with 3D waves, use the /P flag to specify the plane that you want to operate on.

flipRows	Rearrange pixels in the image by exchanging rows symmetrically about the middle row or the middle of the image (if the image has an even number of rows). The exchange is performed in place and can be reverted by repeating the operation. When working with 3D waves, use the /P flag to specify the plane that you want to operate on.
flipPlanes	Rearrange data in a 3D wave by exchanging planes symmetrically about the middle plane. The operation is performed in place and can be reverted by repeating the operation.
fuzzyClassify	Segments grayscale and color images using fuzzy logic. Iteration stops when it reaches convergence defined by /TOL or the maximum number of iterations specified by /I. It is a good practice to specify the tolerance and the number of iterations. If the number of classes is small, the operation prints the class values in the history. Use /Q to eliminate printing and increase performance. Use /CLAS to specify the number of classes and optionally use /CLAM to modify the fuzzy probability values. Use /SEG to generate the segmentation image. The classes are stored in the wave W_FuzzyClasses in the current data folder and it will be overwritten if it already exists. When *imageMatrix* is a grayscale image, each class is a single wave entry. When *imageMatrix* is a RGB image, classes are stored consecutively in the wave W_FuzzyClasses. If you request more classes than are present in *imageMatrix*, you will likely find a degeneracy where the space of a data class is spanned by more than one class. It is a good idea to compute the Euclidean distance between every possible pair of classes and eliminate degeneracies when the distance falls below some threshold. Any real data type is allowed but values in the range [0,255] are optimal. You can segment 3D waves of more than 3 layers in which a class will be a vector of dimensionality equal to the number of layers in *imageMatrix*. For examples see Examples/Imaging/fuzzyClassifyDemo.pxp.
getBeam	Extracts a beam from a 3D wave. A "beam" is a 1D array in the Z-direction. If a row is a 1D array in the first dimension and a column is a 1D array in the second dimension then a beam is a 1D array in the third dimension. The number of points in a beam is equal to the number of layers in *imageWave*. Specify the beam with the /BEAM={*row*,*column*} flag. It stores the result in the wave W_Beam in the current data folder. W_Beam has the same numeric type as *imageWave*. Use **setBeam** to set beam values. (See also, **MatrixOp** beam.)
getChunk	Extracts the chunk specified by chunk index /CHIX from *imageMatrix* and stores it in the wave M_Chunk in the current data folder. For example, if *imageMatrix* has the dimensions (10,20,3,10), the resulting M_Chunk has the dimensions (10,20,3). See also **setChunk** and **insertChunk**.
getCol	Extracts a 1D wave, named W_ExtractedCol, from any type of 2D or 3D wave. You specify the column using the /G flag. For a 3D source wave, it will use the first plane unless you specify a plane using the /P flag. *imageMatrix* can be real or complex. (See also putCol keyword, **MatrixOp** col.)
getPlane	Creates a new wave, named M_ImagePlane, that contains data in the plane specified by the /P flag. The new wave is of the same data type as the source wave. You can specify the type of plane using the /PTYP flag. (See also setPlane keyword, **MatrixOp**.)
getRow	Extracts a 1D wave, named W_ExtractedRow, from any type of 2D or 3D wave. You specify the row using the /G flag. For a 3D source wave, it will use the first plane unless you specify a plane using the /P flag. *imageMatrix* can be real or complex. (See also setRow keyword, **MatrixOp** row.)

Hough

Performs the Hough transform of the input wave. The result is saved to a 2D wave M_Hough in which the columns correspond to angle and the rows correspond to the radial domain.

By default the output consists of 180 columns. Use the /F flag to modify the angular resolution.

If the input image has N rows and M columns then the number of rows in the M_Hough is set to 1+sqrt(N^2+M^2). The output radius should be read relative to the center row.

It is assumed that the input wave has square pixels of unit size and that is binary (/B/U) where the background value is 0.

See also **Hough Transform** on page III-364.

hsl2rgb

Transforms a 3-plane HSL wave into a 3-plane RGB wave. If the source wave for this operation is of any type other than byte or unsigned byte, the HSL values are expected to be between 0 and 65535. For all source wave types the resulting RGB wave is of type unsigned short. The result of the operation is the wave M_HSL2RGB (of type unsigned word), where the RGB values are in the range 0 to 65535.

hslSegment

Creates a binary image of the same dimensions as the source image, in which all pixels that belong to all three of the specified Hue, Saturation, and Lightness ranges are set to 255 and the others to zero. You can specify the HSL ranges using the /H, /S, and /L flags. Each flag takes as an argument either a pair of values or a wave containing pairs of values. You must specify the /H flag but you can omit the /S and /L flags in which case the default values (corresponding to full range 0 to 1) are used. *imageMatrix* is assumed to be an RGB image.

imageToTexture

Transforms a 2D or 3D image wave into a 1D wave of contiguous pixel components. The transformation is useful for creating an OpenGL texture (for Gizmo) or for saving a color image in a format requiring either RGB or RGBA sequences.

The /O flag does not apply to imageToTexture.

Use the /TEXT flag to specify the type of transformation. *imageMatrix* must be an unsigned byte wave. A 1D unsigned byte wave named W_Texture is created in the current data folder.

W_Texture's wave note is set to a semicolon-separated list of keyword -value pairs that can be parsed using **StringByKey** and **NumberByKey**:

Keyword	Information Following Keyword
WIDTHPIXELS	**DimSize**(imageMatrix,0) or truncated to nearest power of 2 if /TEXT value is odd
HEIGHTPIXELS	**DimSize**(imageMatrix,1) or truncated to nearest power of 2
LAYERS	**DimSize**(imageMatrix,2)
TEXTUREMODE	val parameter from /TEXT flag
SOURCEWAVE	**GetWavesDataFolder**(imageMatrix, 2)

indexWave

Creates a 1D wave W_IndexedValues in the current data folder containing values from *imageMatrix* that are pointed to by the index wave (see /IWAV). Each row in the index wave corresponds to a single value of *imageMatrix*. If any row does not point to a valid index (within the dimensions of *imageMatrix*), the corresponding value is set to zero and the operation returns an error. Indices are zero based integers; the operation does not support interpolation and ignores wave scaling.

insertChunk — Inserts a chunk (a 3D wave specified by the /D flag) into *imageMatrix* at chunk index specified by the /CHIX flag. The dimensions of the inserted chunk must match the first three dimensions of *imageMatrix*. The wave must also have the same numeric type. The 4th dimension of *imageMatrix* is incremented by 1 to accommodate the new data. See also **getChunk** and **setChunk**.

insertImage — Inserts the image specified by the flag /INSI into *imageMatrix* starting at the position specified by the flags /INSX and /INSY. If the *imageMatrix* is a 3D wave then it inserts the image in the layer specified by the /P flag. The inserted image and *imageMatrix* must be the same data type. The inserted data is clipped to the boundaries of *imageMatrix*.

insertXplane — Inserts a 2D wave as a new plane perpendicular to the X-axis in a 3D wave. The /P flag specifies the insertion point and the /INSW flag specifies the inserted wave. The 2D wave must be the same numeric data type as the 3D wave and its dimensions must be cols x layers of the 3D wave. For example, if the 3D wave has the dimensions (10x20x30) the 2D wave must be 20x30. If you do not use the /O flag, it stores the result in the wave M_InsertedWave in the current data folder.

insertYplane — Inserts a 2D wave as a new plane perpendicular to the Y-axis in a 3D wave. The /P flag specifies the insertion point and the /INSW flag specifies the inserted wave. The 2D wave must be the same numeric data type as the 3D wave and its dimensions must be rows x layers of the 3D wave. For example, if the 3D wave has the dimensions (10x20x30) the 2D wave must be 10x30. If you do not use the /O flag, it stores the result in the wave M_InsertedWave in the current data folder.

insertZplane — Inserts a 2D wave as a new plane perpendicular to the Z-axis in a 3D wave. The /P flag specifies the insertion point and the /INSW flag specifies the inserted wave. The 2D wave must be the same numeric data type as the 3D wave and its dimensions must be rows x cols of the 3D wave. For example, if the 3D wave has the dimensions (10x20x30) the 2D wave must be 10x20. If you do not use the /O flag, it stores the result in the wave M_InsertedWave in the current data folder. This keyword is included for completeness. You can accomplish the same task using InsertPoints.

invert — Converts pixel values using the formula `newValue=255-oldValue`. Works on waves of any dimension, but only on waves of type unsigned byte. The result is stored in the wave M_Inverted unless specifying the /O flag.

matchPlanes — Finds pixels that match test conditions in all layers of a 3D wave. It creates a 2D unsigned byte output wave, M_matchPlanes, that is set to the values 0 and 255. A value of 255 indicates that the corresponding pixel has satisfied test conditions in all layers of the wave for which conditions were provided. Otherwise the pixel value is 0.

Test conditions are entered as a 2D wave using the /D flag. The condition wave must be double precision and it must contain the same number of columns as the number of layers in the 3D source wave. A condition for layer j of the source wave is specified by two rows in column j of the condition wave. The first row entry, say *A*, and the second row entry, say *B*, imply a condition on pixels in layer *j* such that $A \le x < B$. You can have more than one condition for a given layer by adding pairs of rows to the condition wave. For example, if you add in consecutive rows the values *C* and *D*, this implies the test:

$$(A \le x \le B) \,||\, (C \le x \le D).$$

If you do not have any conditions for some layer, set its corresponding condition column to NaN. Similarly, if you have two conditions for the first layer and one condition for the second layer, pad the bottom of column 1 in the condition wave with NaNs. See Examples for use of this keyword to perform hue/saturation segmentation.

offsetImage	Shifts an image in the XY plane by dx, dy pixels (specified by the /IOFF flag). Pixels outside the shifted image will be set to the specified background value. The operation works on 2D waves or on 3D waves with the optional /P flag. When shifting a 3D wave with no specified plane, it creates a 3D wave with all planes offset by the same amount. The wave M_OffsetImage contains the result in the current data folder. The /O flag is not supported with offsetImage.
padImage	Resizes the source image. When enlarged, values from the last row and column fill in the new area. The /N flag specifies the new image size in terms of the rows and columns change. The /W flag specifies whether data should be wrapped when padding the image. Unless you use the /O flag, the result is stored in the wave M_PaddedImage in the current data folder.
projectionSlice	Computes a projection slice for a parallel fan of rays going through the image at various angles. For every ray in the fan the operation computes a line integral through the image (equivalent to the sum of the line profile along the ray). The operation computes the line integrals for multiple fans defined by the number and position of the rays as well as the angle that they make with the positive X-direction. Use the /PSL flag to specify the projection parameters. The projection slice itself is stored in a 2D wave M_ProjectionSlice where the rows correspond to the rays and the columns correspond to the selected range of angles. The operation does not support wave scaling. If the source wave is 3D the projection slice currently supports slices that are perpendicular to the z-axis and specified by their plane number. See the **backProjection** keyword and the RadonTransformDemo experiment. For algorithm details see the chapter "Reconstruction of cross-sections and the projection-slice theorem of computerized tomography" in Born and Wolf, 1999.
putCol	Sets a column of *imageMatrix* to the values in the wave specified by the /D flag. Use the /G flag to specify column number and the /P flag to specify the plane. Note that if there is a mismatch in the number of entries between the specified waves, the operation uses the smaller number. See also getCol keyword.
putRow	Sets a row of *imageMatrix* to the values in the wave specified by the /D flag. Use the /G flag to specify column number and the /P flag to specify the plane. Note that if there is a mismatch in the number of entries between the specified waves, the operation uses the smaller number. See also getRow keyword.
removeXplane	Removes one or more planes perpendicular to the X-axis from a 3D wave. The /P flag specifies the starting position. By default, it removes a single plane but you can remove more planes with the /NP flag. If you do not use the /O flag, it saves the result in the wave M_ReducedWave in the current data folder.
removeYplane	Removes one or more planes perpendicular to the Y-axis from a 3D wave. The /P flag specifies the starting position. By default, it removes a single plane but you can remove more planes with the /NP flag. If you do not use the /O flag, it saves the result in the wave M_ReducedWave in the current data folder.
removeZplane	Removes one or more planes perpendicular to the Z-axis from a 3D wave. The /P flag specifies the starting position. By default, it removes a single plane but you can remove more planes with the /NP flag. If you do not use the /O flag, it saves the result in the wave M_ReducedWave in the current data folder.

rgb2cmap

Computes a default color map of 256 colors to represent the input RGB image. The colors are computed by clustering the input pixels in RGB space. The resulting color map is stored in the wave M_ColorIndex in the current data folder. The operation also saves the wave M_IndexImage which contains an index into the colormap that can be used to display the image using the commands:

```
NewImage M_IndexImage
ModifyImage M_IndexImage cindex= M_ColorIndex
```

To change the default number of colors use the /NCLR flag. When the number of colors are greater than 256, M_IndexImage will be a 16-bit unsigned integer or a 32 bit integer wave depending on the number. rgb2cmap supports input images in the form of 3D waves of type unsigned byte or single precision float. The floating point option may be used to input images in colorspaces that use signed numeric data.

rgb2gray

When the input *imageMatrix* is a 3D RGB wave, rgb2gray produces a 2D wave of type unsigned byte containing the grayscale representation of the input. By default, the operation stores the output in the wave M_RGB2Gray in the current data folder. The RGB values are converted into the luminance Y of the YIQ standard using:

```
Y = 0.299R + 0.587G + 0.114B
```

When the input imageMatrix is a 4D wave containing multiple (3 layer) RGB chunks, the conversion produces a 3D wave where each layer corresponds to the grayscale conversion of the corresponding chunk in the input wave. In this case the numeric type of the output is the same as that of the input but the conversion formula is the same. The /O flag is not supported when transforming a 4D RGB wave.

rgb2hsl

Converts an RGB image stored in a 3D wave into another 3D wave in which the three planes correspond to Hue, Saturation and Lightness in the HSL color model. Values of all components are normalized to the range 0 to 255 unless the /U flag is used or if the source wave is not 8-bit, in which case the range is 0 to 65535. The default output wave name is M_RGB2HSL.

rgb2i123

Performs a colorspace conversion of an RGB image into the following quantities:

$$I_1 = |R-G|D$$
$$I_2 = |R-B|D \quad \text{where } D = \frac{255}{|R-G|+|R-B|+|G-B|}.$$
$$I_3 = |G-B|D,$$

*I*1, *I*2, and *I*3 are stored in the wave M_I123 (using the same data type as the original RGB wave) in the current data folder. *I*1 is stored in the first layer, *I*2 in the second and *I*3 in the third. This color transformation is said to have useful applications in machine vision.

For more information see: Gevers and Smeulders (1999).

rgb2xyz

Converts a 3D RGB image wave into a 3D wave containing the XYZ color space equivalent. The conversion is based on the D65 white point and uses the following transformation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

The XYZ values are stored in a wave named "M_RGB2XYZ" unless the /O flag is used, in which case the source image is overwritten and converted into single precision wave (NT_FP32).

roiTo1D	Copies all pixels in an ROI and saves them sequentially in a 1D wave. The ROI is specified by /R. The ROI wave must have the same dimensions as *imageMatrix*. If *imageMatrix* is a 3D wave, the ROI must have as many layers as *imageMatrix*. The wave W_roi_to_1d contains the output in the current data folder, has the same numeric type as *imageMatrix*, and contains the selected pixels in a column-major order.
rotateCols	Rotates rows in place. This operation is analogous to the **Rotate** operation except that it works on images and rotates an integer number of rows. The number of rows is specified by the /G flag. When *imageMatrix* contains multiple layers you can use the /P flag to specify the layer of the wave that will undergo rotation. By default, if you do not specify the /P flag and if *imageMatrix* consists of three layers (RGB), then all three layers are rotated. Otherwise the operation rotates only the first layer of the wave.
rotateRows	Rotates columns in place. This operation is analogous to the **Rotate** operation except that it works on images and rotates an integer number of columns. The number of columns is specified by the /G flag. When *imageMatrix* contains multiple layers you can use the /P flag to specify the layer of the wave that will undergo rotation. By default, if you do not specify the /P flag and if *imageMatrix* consists of three layers (RGB), then all three layers are rotated. Otherwise the operation rotates only the first layer of the wave.
scalePlanes	Scales each plane of the 3D wave, *imageMatrix*, by a constant taken from the corresponding entry in the 1D wave specified by the /D flag. The result is stored in the wave M_ScaledPlanes unless the /O flag is specified, in which case scaling is done in place. When using /O, first redimension the wave to a different data type to make sure there are no artifacts due to type clipping. If *imageMatrix* is double precision, M_ScaledPlanes is double precision. Otherwise M_ScaledPlanes is single precision. This operation also supports the optional flag. Note that when you display M_ScaledPlanes, which has three planes that originated from scaling byte data, you will have to multiply the wave by 255 to see the image because the RGB format for single and double precision data requires values in the range 0 to 65535.
selectColor	Creates a mask for the image in which pixel values depend on the proximity of the color of the image to a given central color. The central color, the tolerance and a grayscale indicator must be specified using the /E flag. For example, `/E={174,187,75,10,1}` specifies an RGB of (174,187,75), a tolerance level of 10 and a requested grayscale output. RGB values must be provided in a range appropriate for the source image. If the source wave type is not byte or unsigned byte, then the range of the RGB components should be 0 to 65535. The color proximity is calculated in the nonuniform RGB space and the tolerance applies to the maximum component difference from the corresponding component of the central color. The tolerance, just like the central color, should be appropriate to the type of the source wave. The generated mask is stored in the wave M_SelectColor in the current data folder. If a wave by that name exists prior to the execution of this operation, it is overwritten. You can also use the /R flag with this operation to limit the color selection to pixels in the ROI wave whose value is zero.

setBeam	Sets the data of a particular beam in *imageMatrix*. A "beam" is a 1D array in the Z-direction. If a row is a 1D array in the first dimension and a column is a 1D array in the second dimension then a beam is a 1D array in the third dimension. Specify the beam with the /BEAM={*row*,*column*} flag and the 1D beam data wave with the /D flag. The beam data wave must have the same number of elements as the number of layers and same numeric type as *imageMatrix*. Use **getBeam** to extract the beam.
setChunk	Overwrites the data in the wave *imageMatrix* at chunk index specified by /CHIX with the data contained in a 3D wave specified by the /D flag. The assigned data must be contained in a wave that matches the first three dimensions of *imageMatrix* and must have the same number type. See also **getChunk** and **insertChunk**.
setPlane	Sets a plane (given by the /P flag) in the designated image with the data in a wave specified by the /D flag. It is designed as a complement of the **getPlane** keyword to provide an easier (faster) way to create multiplane images. Note that the operation supports setting a plane when the source data is smaller than the destination plane in which case the source data is placed in memory contiguously starting from the corner pixel of the destination plane. If the source data is larger than the destination plane it is clipped to the appropriate rows and columns. If you are setting all planes in the destination wave using algorithmically named source waves you could use the **stackImages** keyword instead. See also **getPlane** keyword.
shading	Calculates relative reflectance of a surface for a light source position defined by the /A flag. The operation estimates the slope of the surface and then computes a relative reflectance defined as the dot product of the direction of the light and the normal to the surface at the point of interest. Reflectivity is scaled using the expression: *outPixel* = *shadingA* * (*sunDirection* · *surfaceNormal*) + *shadingB* By default *shadingA*=1, *shadingB*=0. The result is stored in the wave M_ShadedImage, which has the same data type as the source wave. If the source wave is any integer type, and the value of *shadingA*=1 the operation sets that value to 255. The smallest supported wave size is 4x4 elements. Values along the boundary (1 pixel wide) are arbitrary because there are no derivatives calculable for those pixels, so these pixels are filled with duplicates of the inner rows and columns.
shrinkBox	Shrinks 3D imageMatrix to include only the minimum three dimension rectangular range that contains all the voxels whose values are different from an outer value. The outer value is specified with the /F flag. This feature is useful in situations where ImageSeedFill has set the voxels around an object of interest to some outer value and it is desired to extract the smallest box that contains interesting data. The output is stored in the wave M_shrunkBox. Added in Igor Pro 7.00.
shrinkRect	Shrinks *imageMatrix* to include only the minimum rectangle that contains all the pixels whose value is different from an outer value. The outer value is specified with the /F flag. This is useful in situations where **ImageSeedFill** has set the pixels around the object of interest to some outer value and it is desired to extract the smallest rectangle that contains interesting data. The output is stored in the wave M_Shrunk.

stackImages — Creates a 3D or 4D stack from individual image waves in the current data folder. The waves should be of the form *baseNameN*, where *N* is a numeric suffix specifying the sequence order. *imageMatrix* should be the name of the first wave that you want to add to the stack. You can use the /NP flag to specify the number of waves that you want to add to the stack.

The result is a 3D or 4D wave named M_Stack, which overwrites any existing wave of that name in the current data folder.

With /K, it kills all waves copied into the stack.

sumAllCols — Creates a wave W_sumCols in which every entry is the sum of the pixels on the corresponding image column. For a 3D wave, unless you specify a plane using the /P flag it will use the first plane by default.

sumAllRows — Creates a wave W_sumRows in which every entry is the sum of the pixels on the corresponding image row. For a 3D wave, unless you specify a plane using the /P flag it will use the first plane by default.

sumCol — Stores in the variable V_value the sum of the elements in the column specified by /G flag and optionally the /P flag.

sumPlane — Stores in the variable V_value the sum of the elements in the plane specified by the /P flag.

sumPlanes — Creates a 2D wave M_SumPlanes which contains the same number of rows and columns as the 3D source wave. Each entry in M_SumPlanes is the sum of the corresponding pixels in all the planes of the source wave. M_SumPlanes is a double precision wave if the source wave is double precision. Otherwise it is a single precision wave.

sumRow — Stores in the variable V_value the sum of the elements of a row specified by /G flag and optionally the /P flag.

swap — Swaps image data following a 2D FFT. The transform swaps diagonal quadrants of the image in one or more planes. This keyword does not support any flags. The swapping is done in place and it overwrites the source wave.

swap3D — Swaps data following a 3D FFT. The transform swaps diagonal quadrants of the data. This keyword does not support any flags. The swapping is done in place and the source wave is overwritten.

transpose4D — Converts a 4D wave into a new 4D wave where the data are reordered by dimensions specified by the /TM4D flag. The results are stored in the wave M_4DTranspose in the current data folder. There is no option to overwrite the input wave so /O has no effect with transpose4D.

Added in Igor Pro 7.00.

transposeVol — Transposes a 3D wave. The transposed wave is stored in M_VolumeTranspose. The /O flag does not apply. The operation supports the following 5 transpose modes which are specified using the /G flag:

mode	**Equivalent Command**
1	`M_VolumeTranspose=`*`imageMatrix`*`[p][r][q]`
2	`M_VolumeTranspose=`*`imageMatrix`*`[r][p][q]`
3	`M_VolumeTranspose=`*`imageMatrix`*`[r][q][p]`
4	`M_VolumeTranspose=`*`imageMatrix`*`[q][r][p]`
5	`M_VolumeTranspose=`*`imageMatrix`*`[q][p][r]`

vol2surf — Creates a quad-wave output (appropriate for display in Gizmo) that wraps around 3D "particles". A particle is defined as a region of nonzero value voxels in a 3D wave. The algorithm effectively computes a box at the resolution of the input wave which completely encloses the data. The output wave M_Boxy is a 2D single precision wave of 12 columns where each row corresponds to one disjoint quad and the columns provide the sequential X, Y, and Z coordinates of the quad vertices.

voronoi — Computes the voronoi tesselation of a convex domain defined by the X, Y positions of the input wave. *imageMatrix* must be a triplet wave where the first column contains the X-values, the second column contains the Y-values and the third column is an arbitrary (zero is recommended) constant. The result of the operation is stored in the two column wave M_VoronoiEdges which contains sequential edges of the Voronoi polygons. Edges are separated from each other by a row of NaNs. The outer most polygons share one or more edges with a large triangle which contains the convex domain.

The operation creates two output waves:

M_Circles is a three column wave containing the center and radius of each natural neighborhood. The radius is correct only if equal scaling is applied.

M_DelaunayTriangles consists of 3 columns for the vertices of the triangles. Each triangle consists of 4 vertices. The 4th vertex is equal to the first. An extra row of NaNs used as a separator.

For an example, see Voronoi Tesselation Example below.

xProjection — Computes the projection in the X-direction and stores the result in the wave M_xProjection. See **zProjection** for more information.

xyz2rgb — Converts a 3D single precision XYZ color-space data into RGB based on the D65 white point. The transformation used is:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

If you do not specify the /O flag, the results are saved in a single precision 3D wave (NT_FP32) "M_XYZ2RGB".

Note that not all XYZ values map into positive RGB triplets (consider colors that reside outside the RGB triangle in the XYZ diagram). This operation gives you the following choices: by default, the output wave is a single precision wave that will include possible negative RGB values. If you specify the /U flag, for unsigned short output wave, the operation will set to zero all negative components and scale the remaining ones in the range 0 to 65535.

yProjection — Computes the projection in the Y-direction and stores the result in the wave M_yProjection. See **zProjection** for more information.

zDot — Computes the dot product of a beam in *srcWave* with a 1D zVector wave (specified with the /D flag). This will convert stacked images of spectral scans into RGB or XYZ depending on the scaling in zVector. The *srcWave* and zVector must be the same data type (float or double). The wave M_StackDot contains the result in the current data folder.

zProjection — Computes the projection in the Z-direction and stores the result in the wave M_zProjection. The source wave must be a 3D wave of arbitrary data type. The value of the projection depends on the method specified via the /METH flag.

Flags

/A={*azimuth*, *elevation* [, *shadingA*, *shadingB*]}

	Specifies parameters for shade. Position of the light source is given by *azimuth* (measured in degrees counter-clockwise) and *elevation* (measured in degrees above the horizon). The parameters *shadingA* and *shadingB* are optional. By default their values are 1 and 0, respectively.
/Beam={*row*,*column*}	Designates a beam in a 3D wave; both *row* and *column* are zero based.
/BPJ={*width*,*height*}	Specifies the **backProjection** parameters: *width* and *height* are the width and height of the reconstructed image and should be equal to the size of the original wave.
/C=*CMapWave*	Specifies the colormap wave for **cmap2rgb** keyword. The *CMapWave* is expected to be a 2D wave consisting of three columns corresponding to the RGB entries.
/CHIX=*chunkIndex*	Identifies the chunk index for getting, inserting or setting a chunk of data in a 4D wave. *chunkIndex* ranges from 0 to the number of chunks in *imageMatrix*.
/CLAM=*fuzzy*	Sets the value used to compute the fuzzy probability in **fuzzyClassify**. It must satisfy *fuzzy* > 1 (default is 2).
/CLAS=*num*	Sets the number of requested classes in **fuzzyClassify**. If you don't know the number of expected classes and *num* is too high, fuzzyClassify will likely produce some degenerate classes.
/D=*waveName*	Specifies a data wave. Check the appropriate keyword documentation for more information about this wave.
/F=*value*	Increases the sampling in the angle domain when used with the Hough keyword. By default *value*=1 and the operation results in single degree increments in the interval 0 to 180, and if *value*=1.5 there will be 180*1.5 rows in the transform. Specifies the outer pixel value surrounding the region of interest when used with **shrinkRect** keyword.
/FEQS	When performing Voronoi tesselation, forces the algorithm to use equal scaling for both axes. Normally the scaling for each axis is determined from its range. When the ranges of the two axes are different by an order of magnitude or more, it is helpful to force the larger range to be used for scaling both axes.
/G=*colNumber*	Specifies either the row or column number used in connection with **getRow** or **getCol** keywords. This flag also specifies the transpose *mode* with the **transposeVol** keyword.
/H={*minHue*, *maxHue*} /H=*hueWave*	Specifies the range of hue values for selecting pixels. The hue values are specified in degrees in the range 0 to 360. Hue intervals that contain the zero point should be specified with the higher value first, e.g., `/H={330,10}`. Use *hueWave* when you have more than one pair of hue values that bracket the pixels that you want to select. See **HSL Segmentation** on page III-374 for an example.
/I=*iterations*	Sets the number of iterations in ccsubdivision and in **fuzzyClassify**.
/INSI=*imageWave*	Specifies the wave, *imageWave*, to be used with the insertImage keyword. *imageWave* is a 2D wave of the same numeric data type as *imageMatrix*.
/INSW=*wave*	Specifies the 2D wave to be inserted into a 3D wave using the keywords: **insertXplane**, **insertYplane**, or **insertZplane**.
/INSX=*xPos*	Specifies the pixel position at which the first row is inserted. Ignores wave scaling.
/INSY=*yPos*	Specifies the pixel position at which the first column is inserted. Ignores wave scaling.

/IOFF={*dx*,*dy*,*bgValue*} Specifies the amount of positive or negative integer offset with *dx* and *dy* and the new background value, *bgValue*, with the offsetImage keyword.

/IWAV=*wave* Specifies the wave which provides the indices when used with the keyword **indexWave**. The wave should have as many columns as the dimensions of *imageMatrix* (2, 3, or 4). For example, to specify indices for pixels in an image, the wave should have two columns. The first column corresponds to the row designation and the second to the column designation of the pixel. The wave can be of any number type (other than complex) and entries are assumed to be integer indices; there is no support for interpolation or for wave scaling.

/L={*minLight, maxLight*} Specifies the range of lightness for selecting pixels. The lightness values are in the range 0-1. If you do not use the /L flag than the default full range is used.

/L=*lightnessWave*

Use *lightnessWave* when you have more than one pair of lightness values corresponding to the pixels that you want to select. For each pair, values should be arranged so that the smaller one is first and the larger is second. There is no restriction on the order of pairs in the wave except that they match the other waves used by the operation.

/LARA=*minPixels* Specifies the minimum number of pixels required for an area to be masked by the **findLakes** keyword. If you do not specify this flag, the default value used is 100.

/LCVT Use 8-connectivity instead of 4-connectivity.

/LRCT={*minX*,*minY*,*maxX*,*maxY*}

Sets the rectangular region of interest for the **findLakes** keyword. The operation will not affect the original data outside the specified rectangle. The X and Y values are the scaled values (i.e., using wave scaling).

/LTAR=*target* Set the target value for the masked region in the **findLakes** keyword.

/LTOL=*tol* Specifies the tolerance for the **findLakes** keyword. By default the tolerance is zero. The tolerance must be a positive number. The operation uses the tolerance by requiring neighboring pixels to have a value between that of the current pixel V and V+*tol*.

/M=*n* Specifies the method by which a 2D target image is filled with data from a 1D wave using the **fillImage** keyword.

- *n* =0: Straight column fill, which you can also accomplish by a redimension operation.
- *n*=1: Straight row fill.
- *n*=2: Serpentine column fill. The points from the data wave are sequentially loaded onto the first column and continue from the last to the first point of the second column, and then sequentially through the third column, etc.
- *n*=3: Serpentine row fill.

/METH=*method* Determines the values of the projected pixels for **xProjection**, **yProjection**, and **zProjection** keywords.

- *method*=1: Pixel (i,j) in M_zProjection is assigned the maximum value that (i,j,*) takes among all layers of *imageMatrix* (default).
- *method*=2: Pixel (i,j) in M_zProjection is assigned the average value that (i,j,*) takes among all layers of *imageMatrix*.
- *method*=3: Pixel (i,j) in M_zProjection is assigned the minimum value that (i,j,*) takes among all layers of *imageMatrix*.

/METR=*method* Sets the metric used by the distance transform. Added in Igor Pro 7.00.

method=0: Manhattan distance. Default.

method=1: Euclidean distance where the distance is measured between centers of pixels (assumed square).

method=2: Euclidean distance that also takes into account actual pixel size using the input's wave scaling.

method=0 executes faster than the other methods. *method*=2 can be 2x slower especially if different scaling is applied along the two axes.

/N={*rowsToAdd, colsToAdd*}

Creates an image that is larger or smaller by *rowsToAdd*, *colsToAdd*. The additional pixels are set by duplicating the values in the last row and the last column of the source image.

/NCLR=*M* Specifies the maximum number of colors to find with the rgb2cmap keyword. *M* must be a positive number; the default value is 256 colors.

The result of the operation is saved in the wave M_paddedImage.

/NP=*numPlanes* Specifies the number of planes to remove from a 3D wave when using the removeXplane, removeYplane, or removeZplane keywords. Specifies the number of waves to be added to the stack with the stackImages keyword.

/O Overwrites the input wave with the result except in the cases of **Hough** transform and **cmap2rgb**. Does not apply to the transposeVol parameter.

/P=*planeNum* Specifies the plane on which you want to operate with the **rgb2gray** or **getPlane** keywords. Also used for **getRow** or **getCol** if the source wave is 3D.

/PSL={*xStart*,*dx*,*Nx*,*aStart*,*da*,*Na*}

Specifies projection slice parameters. *xStart* is the first offset of the parallel rays measured from the center of the image. *dx* is the directed offset to the next ray in the fan and *Nx* is the number of rays in the fan. *aStart* is the first angle for which the projection is calculated. The angle is measured between the positive X-direction and the direction of the ray. *da* is the offset to the next angle at which the fan of rays is rotated and *Na* is the total number of angles for which the projection is computed.

/PTRF Use /PTRF with the ccsubdivision keyword to apply small perturbation to the input coordinates in order to break spatial degeneracies that may occur when multiple facets meet at some point in space. See ccsubdivision above for details. /PTRF was added in Igor Pro 9.00.

/PTYP=*num* Specifies the plane to use with the **getPlane** keyword.

num=0: XY plane.

num=1: XZ plane.

num=2: YZ plane.

/Q Quiet flag. When used with the **Hough** transform, it suppresses report to the history of the angle corresponding to the maximum.

/R=*roiWave* Specifies a region of interest (ROI) defined by *roiWave*. For use with the keywords: **averageImage**, **scalePlanes** and **selectColor**.

/S={*minSat, maxSat*}

/S=*saturationWave*

Specifies the range of saturation values for selecting pixels. The saturation values are in the range 0 to 1. If you do not use the /S flag, the default value is the full saturation range.

Use *saturationWave* when you have more than one pair of saturation values. If you use a saturation wave you must also use a lightness wave (see /L). *saturationWave* should consist of pairs of values where the first point is the lower saturation value and the second point is the higher saturation value. There is no restriction on the order of pairs within the wave.

/SEG

Computes the segmentation image for **fuzzyClassify**. The image is stored in the 2D wave M_FuzzySegments. The value of each pixel is 255**classIndex/number of classes*. Here *classIndex* is the index of the class to which the pixel belongs with the highest probability.

/T=*flag*

Use one or more of the following flags.

1: Swaps the data so that the DC is at the center of the image.

2: Calculates the power defined as: $P(f) = 0.5 \cdot (H(f)^2 + H(-f)^2)$.

/TEXT=*val*

Specifies the type of texture to create with the **imageToTexture** keyword. *val* is a binary flag that can be a combination of the following values.

val	Texture
1	Truncates each dimension to the nearest power of 2, which is required for OpenGL textures.
2	Creates a 1D texture (all other textures are for 2D applications).
4	Creates a single channel texture suitable for alpha or luminance channels.
8	Creates a RGB texture from a 3 (or more) layer data.
16	Creates a RGBA texture. If *imageMatrix* does not have a 4th layer, alpha is set to 255.

/TOL=*tolerance*

Sets the tolerance for iteration convergence with **fuzzyClassify**. Convergence is satisfied when the sum of the squared differences of all classes drops below *tolerance*, which must not be negative.

/TM4D=*mode*

Used with transpose4D to specify the format of the output wave. Here *mode* is a 4 digit integer that describes the mapping of the transformation. The digit 1 is used to represent the first dimension, 2 for the second, 4 for the third and 8 for the 4th dimension such that the original wave corresponds to mode=1248. All other modes are obtained by permuting one or more of the four digits and mode must consist of 4 distinct digits.

Added in Igor Pro 7.00.

/U

Creates an HSL wave of type unsigned short that contains values between 0 and 65535 when used with **rgb2hsl**.

/W

Pads the image by wrapping the data. If you are adding more rows or more columns than are available in the source wave, the operation cycles through the source data as many times as necessary.

/X={*Nx*,*Ny*,*x1*,*y1*,*z1*,*x2*,*y2*,*z2*,*x3*,*y3*,*z3*}

Nx and *Ny* are the rows and columns of the wave M_ExtractedSurface. The remaining parameters specify three 3D points on the extracted plane. The three points must be chosen at the vertices of the plane and entered in clock-wise order without skipping a vertex.

/Z

Ignores errors.