

# **Vector Math Library Reference**

© 2015 Sony Computer Entertainment Inc.  
All Rights Reserved.  
SCE Confidential

# Table of Contents

<b>Introduction.....</b>	<b>42</b>
Library Summary.....	43
<b>sce.....</b>	<b>44</b>
Summary.....	45
sce.....	45
<b>sce::Vectormath.....</b>	<b>46</b>
Summary .....	47
sce::Vectormath .....	47
<b>sce::Vectormath::Simd .....</b>	<b>48</b>
Summary .....	49
sce::Vectormath::Simd .....	49
Enumerated Types .....	51
RotationOrder.....	51
Type Definitions .....	52
boolInVec_arg .....	52
floatInVec_arg .....	53
boolInVec Functions .....	54
operator!=.....	54
operator& .....	55
operator&& .....	56
operator==.....	57
operator^.....	58
operator  .....	59
operator   .....	60
select .....	61
floatInVec Functions .....	62
clamp.....	62
operator!=.....	63
operator* .....	64
operator+ .....	65
operator- .....	66
operator/ .....	67
operator< .....	68
operator<=.....	69
operator==.....	70
operator> .....	71
operator>=.....	72
select .....	73
select .....	74
floatInVec Math Functions .....	75
acosf.....	75
asinf.....	76
atan2f .....	77
atanf .....	78

SCE CONFIDENTIAL

cbrtf .....	79
ceilf .....	80
copysignf .....	81
cosf .....	82
coshf .....	83
divf .....	84
exp2f .....	85
expf .....	86
expm1f .....	87
fabsf .....	88
fdimf .....	89
floorf .....	90
fmaf .....	91
fmaxf .....	92
fminf .....	93
fmodf .....	94
hypotf .....	95
log10f .....	96
log1pf .....	97
logbf .....	98
logf .....	99
logtwof .....	100
modff .....	101
negatef .....	102
powf .....	103
recipf .....	104
remainderf .....	105
rsqrtf .....	106
sincosf .....	107
sinf .....	108
sinhf .....	109
sqrftf .....	110
tanf .....	111
tanhf .....	112
truncf .....	113
<b>sce::Vectormath::Simd::Aos .....</b>	<b>114</b>
Summary .....	115
sce::Vectormath::Simd::Aos .....	115
Type Definitions .....	123
Matrix2_arg .....	123
Matrix3_arg .....	124
Matrix4_arg .....	125
Point3_arg .....	126
Quat_arg .....	127
Transform3_arg .....	128
Vector2_arg .....	129
Vector3_arg .....	130
Vector4_arg .....	131

SCE CONFIDENTIAL

Functions .....	132
absPerElem.....	132
appendScale .....	133
determinant .....	134
inverse.....	135
mulPerElem.....	136
operator* .....	137
operator* .....	138
prependScale .....	139
print .....	140
print .....	141
select.....	142
select.....	143
transpose .....	144
2D Vector Functions .....	145
absPerElem.....	145
allElemEqual .....	146
allElemGreaterThan .....	147
allElemGreaterThanOrEqual.....	148
allElemLessThan.....	149
allElemLessThanOrEqual .....	150
allElemNotEqual.....	151
angle .....	152
angleBetween .....	153
clampPerElem.....	154
copySignPerElem.....	155
determinant .....	156
divPerElem.....	157
dot .....	158
length .....	159
lengthSqr.....	160
lerp .....	161
lerp .....	162
loadHalfFloats .....	163
loadHalfFloats .....	164
loadXY.....	165
loadXY.....	166
loadXYArray .....	167
maxElem .....	168
maxPerElem.....	169
minElem .....	170
minPerElem.....	171
mulPerElem.....	172
normalize.....	173
operator* .....	174
operator* .....	175
perp .....	176
print .....	177

SCE CONFIDENTIAL

print .....	178
recipPerElem.....	179
rotate .....	180
rsqrtPerElem .....	181
select .....	182
select.....	183
slerp .....	184
slerp .....	185
sqrtPerElem .....	186
storeHalfFloats .....	187
storeHalfFloats .....	188
storeXY .....	189
storeXY .....	190
storeXYZArray .....	191
sum.....	192
<b>3D Vector Functions</b> .....	<b>193</b>
absPerElem.....	193
allElemEqual .....	194
allElemGreaterThan.....	195
allElemGreaterThanOrEqual.....	196
allElemLessThan.....	197
allElemLessThanOrEqual .....	198
allElemNotEqual.....	199
clampPerElem .....	200
copySignPerElem.....	201
cross.....	202
crossMatrix.....	203
crossMatrixMul .....	204
divPerElem.....	205
dot .....	206
length .....	207
lengthSqr .....	208
lerp .....	209
lerp .....	210
loadHalfFloats .....	211
loadHalfFloats .....	212
loadXYZ .....	213
loadXYZ .....	214
loadXYZArray .....	215
maxElem .....	216
maxPerElem.....	217
minElem .....	218
minPerElem.....	219
mulPerElem.....	220
normalize.....	221
operator* .....	222
operator* .....	223
outer .....	224

SCE CONFIDENTIAL

print .....	225
print .....	226
recipPerElem.....	227
rowMul.....	228
rsqrtPerElem .....	229
select.....	230
select.....	231
slerp .....	232
slerp .....	233
sqrtPerElem .....	234
storeHalfFloats .....	235
storeHalfFloats .....	236
storeXYZ .....	237
storeXYZ .....	238
storeXYZArray.....	239
sum.....	240
<b>4D Vector Functions .....</b>	<b>241</b>
absPerElem.....	241
allElemEqual .....	242
allElemGreaterThan.....	243
allElemGreaterThanOrEqual.....	244
allElemLessThan.....	245
allElemLessThanOrEqual .....	246
allElemNotEqual.....	247
clampPerElem .....	248
copySignPerElem.....	249
divPerElem.....	250
dot .....	251
length .....	252
lengthSqr.....	253
lerp .....	254
lerp .....	255
loadHalfFloats .....	256
loadHalfFloats .....	257
loadXYZW .....	258
loadXYZW .....	259
maxElem .....	260
maxPerElem.....	261
minElem .....	262
minPerElem.....	263
mulPerElem.....	264
normalize .....	265
operator* .....	266
operator* .....	267
outer .....	268
print .....	269
print .....	270
recipPerElem.....	271

SCE CONFIDENTIAL

rsqrtPerElem .....	272
select.....	273
select.....	274
slerp .....	275
slerp .....	276
sqrtPerElem .....	277
storeHalfFloats .....	278
storeHalfFloats .....	279
storeXYZW .....	280
storeXYZW .....	281
sum.....	282
3D Point Functions .....	283
absPerElem.....	283
allElemEqual .....	284
allElemGreaterThan.....	285
allElemGreaterThanOrEqual.....	286
allElemLessThan.....	287
allElemLessThanOrEqual .....	288
allElemNotEqual.....	289
clampPerElem .....	290
copySignPerElem.....	291
dist.....	292
distFromOrigin.....	293
distSqr .....	294
distSqrFromOrigin .....	295
divPerElem .....	296
lerp .....	297
lerp .....	298
loadHalfFloats .....	299
loadHalfFloats .....	300
loadXYZ .....	301
loadXYZ .....	302
loadXYZArray .....	303
maxElem .....	304
maxPerElem.....	305
minElem .....	306
minPerElem.....	307
mulPerElem.....	308
print .....	309
print .....	310
projection.....	311
recipPerElem.....	312
rsqrtPerElem .....	313
scale .....	314
scale .....	315
scale .....	316
select.....	317
select.....	318

SCE CONFIDENTIAL

sqrtPerElem .....	319
storeHalfFloats .....	320
storeHalfFloats .....	321
storeXYZ .....	322
storeXYZ .....	323
storeXYZArray.....	324
sum.....	325
Quaternion Functions .....	326
conj.....	326
dot .....	327
length .....	328
lengthSqr.....	329
lerp .....	330
lerp .....	331
loadXYZW .....	332
loadXYZW .....	333
norm .....	334
normalize.....	335
operator* .....	336
operator* .....	337
print .....	338
print .....	339
rotate .....	340
select.....	341
select.....	342
slerp .....	343
slerp .....	344
squad .....	345
squad .....	346
storeXYZW .....	347
storeXYZW .....	348
3x3 Matrix Functions .....	349
absPerElem.....	349
appendScale .....	350
determinant .....	351
inverse.....	352
mulPerElem .....	353
operator* .....	354
operator* .....	355
prependScale .....	356
print .....	357
print .....	358
select.....	359
select.....	360
transpose .....	361
4x4 Matrix Functions .....	362
absPerElem.....	362
affineInverse .....	363

SCE CONFIDENTIAL

appendScale .....	364
determinant .....	365
inverse.....	366
mulPerElem.....	367
operator* .....	368
operator* .....	369
orthoInverse .....	370
prependScale .....	371
print .....	372
print .....	373
select.....	374
select.....	375
transpose .....	376
<b>3x4 Transformation Matrix Functions .....</b>	<b>377</b>
absPerElem.....	377
appendScale .....	378
inverse.....	379
mulPerElem.....	380
orthoInverse .....	381
prependScale .....	382
print .....	383
print .....	384
select.....	385
select.....	386
<b>sce::Vectormath::Simd::Aos::Matrix2 .....</b>	<b>387</b>
Summary .....	388
sce::Vectormath::Simd::Aos::Matrix2 .....	388
Constructors and Destructors .....	390
Matrix2 .....	390
Matrix2 .....	391
Matrix2 .....	392
Matrix2 .....	393
Matrix2 .....	394
Matrix2 .....	395
Matrix2 .....	396
Operator Methods .....	397
operator* .....	397
operator* .....	398
operator* .....	399
operator* .....	400
operator*= .....	401
operator*= .....	402
operator*= .....	403
operator+ .....	404
operator+=.....	405
operator- .....	406
operator- .....	407
operator-=.....	408

SCE CONFIDENTIAL

---

operator=.....	409
operator[].....	410
Public Static Methods .....	411
identity.....	411
rotation .....	412
rotation .....	413
scale .....	414
zero .....	415
Public Instance Methods .....	416
get128 .....	416
getCol .....	417
getCol0.....	418
getCol1 .....	419
getElem .....	420
getRow .....	421
setCol .....	422
setCol0.....	423
setCol1 .....	424
setElem .....	425
setRow .....	426
.....	427
<b>sce::Vectormath::Simd::Aos::Matrix3.....</b>	<b>428</b>
Summary .....	429
sce::Vectormath::Simd::Aos::Matrix3.....	429
Constructors and Destructors .....	431
Matrix3 .....	431
Matrix3 .....	432
Matrix3 .....	433
Matrix3 .....	434
Matrix3 .....	435
Matrix3 .....	436
Operator Methods .....	437
operator* .....	437
operator* .....	438
operator* .....	439
operator* .....	440
operator*=.....	441
operator*=.....	442
operator*=.....	443
operator+ .....	444
operator+=.....	445
operator-.....	446
operator-.....	447
operator-=.....	448
operator= .....	449
operator[] .....	450
operator[] .....	451

SCE CONFIDENTIAL

Public Static Methods .....	452
identity .....	452
rotation .....	453
rotation .....	454
rotation .....	455
rotationX .....	456
rotationX .....	457
rotationY .....	458
rotationY .....	459
rotationZ .....	460
rotationZ .....	461
rotationZYX .....	462
scale .....	463
zero .....	464
Public Instance Methods .....	465
getCol .....	465
getCol0 .....	466
getCol1 .....	467
getCol2 .....	468
getElem .....	469
getRow .....	470
setCol .....	471
setCol0 .....	472
setCol1 .....	473
setCol2 .....	474
setElem .....	475
setElem .....	476
setRow .....	477
<b>sce::Vectormath::Simd::Aos::Matrix4 .....</b>	<b>478</b>
Summary .....	479
sce::Vectormath::Simd::Aos::Matrix4 .....	479
Constructors and Destructors .....	481
Matrix4 .....	481
Matrix4 .....	482
Matrix4 .....	483
Matrix4 .....	484
Matrix4 .....	485
Matrix4 .....	486
Matrix4 .....	487
Matrix4 .....	488
Operator Methods .....	489
operator* .....	489
operator* .....	490
operator* .....	491
operator* .....	492
operator* .....	493
operator* .....	494
operator* .....	495

SCE CONFIDENTIAL

operator*= .....	496
operator*= .....	497
operator*= .....	498
operator*= .....	499
operator+ .....	500
operator+= .....	501
operator- .....	502
operator- .....	503
operator-= .....	504
operator= .....	505
operator[] .....	506
operator[] .....	507
Public Static Methods .....	508
frustum .....	508
frustum .....	510
identity .....	511
lookAt .....	512
orthographic .....	513
orthographic .....	514
perspective .....	516
perspective .....	517
rotation .....	518
rotation .....	519
rotation .....	520
rotationX .....	521
rotationX .....	522
rotationY .....	523
rotationY .....	524
rotationZ .....	525
rotationZ .....	526
rotationZYX .....	527
scale .....	528
translation .....	529
zero .....	530
Public Instance Methods .....	531
getCol .....	531
getCol0 .....	532
getCol1 .....	533
getCol2 .....	534
getCol3 .....	535
getElem .....	536
getRow .....	537
getTranslation .....	538
getUpper3x3 .....	539
setCol .....	540
setCol0 .....	541
setCol1 .....	542
setCol2 .....	543

SCE CONFIDENTIAL

setCol3 .....	544
setElem .....	545
setElem .....	546
setRow .....	547
setTranslation .....	548
setUpper3x3 .....	549
<b>sce::Vectormath::Simd::Aos::Point3 .....</b>	<b>550</b>
Summary .....	551
sce::Vectormath::Simd::Aos::Point3 .....	551
Constructors and Destructors .....	553
Point3 .....	553
Point3 .....	554
Point3 .....	555
Point3 .....	556
Point3 .....	557
Point3 .....	558
Point3 .....	559
Point3 .....	560
Point3 .....	561
Point3 .....	562
Point3 .....	563
Operator Methods .....	564
operator vec_float3_ext .....	564
operator+ .....	565
operator+= .....	566
operator- .....	567
operator-.. .....	568
operator-= .....	569
operator= .....	570
operator[] .....	571
operator[] .....	572
Public Static Methods .....	573
origin .....	573
Public Instance Methods .....	574
get128 .....	574
getElem .....	575
getExtVector .....	576
getX .....	577
getXY .....	578
getY .....	579
getZ .....	580
set128 .....	581
setElem .....	582
setElem .....	583
setExtVector .....	584
setX .....	585
setX .....	586
setXY .....	587

SCE CONFIDENTIAL

setY .....	588
setY .....	589
setZ .....	590
setZ .....	591
Protected Instance Methods .....	592
get128Ref.....	592
<b>sce::Vectormath::Simd::Aos::Quat.....</b>	<b>593</b>
Summary .....	594
sce::Vectormath::Simd::Aos::Quat.....	594
Constructors and Destructors .....	596
Quat .....	596
Quat .....	597
Quat .....	598
Quat .....	599
Quat .....	600
Quat .....	601
Quat .....	602
Quat .....	603
Quat .....	604
Quat .....	605
Quat .....	606
Quat .....	607
Operator Methods .....	608
operator vec_float4_ext .....	608
operator* .....	609
operator* .....	610
operator* .....	611
operator*= .....	612
operator*+= .....	613
operator*+= .....	614
operator+ .....	615
operator+= .....	616
operator- .....	617
operator- .....	618
operator-= .....	619
operator/ .....	620
operator/ .....	621
operator/= .....	622
operator/= .....	623
operator= .....	624
operator[] .....	625
operator[] .....	626
Public Static Methods .....	627
axisAngle.....	627
euler .....	628
identity .....	629
rotation .....	630
rotation .....	631

SCE CONFIDENTIAL

rotation .....	632
rotation .....	633
rotationX .....	634
rotationX .....	635
rotationY .....	636
rotationY .....	637
rotationZ .....	638
rotationZ .....	639
zero .....	640
Public Instance Methods .....	641
get128 .....	641
getElem .....	642
getExtVector .....	643
getW .....	644
getX .....	645
getXY .....	646
getXYZ .....	647
getY .....	648
getZ .....	649
set128 .....	650
setElem .....	651
setElem .....	652
setExtVector .....	653
setW .....	654
setW .....	655
setX .....	656
setX .....	657
setXY .....	658
setXYZ .....	659
setY .....	660
setY .....	661
setZ .....	662
setZ .....	663
Protected Instance Methods .....	664
get128Ref .....	664
<b>sce::Vectormath::Simd::Aos::Transform3.....</b>	<b>665</b>
Summary .....	666
sce::Vectormath::Simd::Aos::Transform3 .....	666
Constructors and Destructors .....	668
Transform3 .....	668
Transform3 .....	669
Transform3 .....	670
Transform3 .....	671
Transform3 .....	672
Transform3 .....	673
Transform3 .....	674
Operator Methods .....	675
operator* .....	675

operator* .....	676
operator* .....	677
operator*= .....	678
operator= .....	679
operator[] .....	680
operator[] .....	681
Public Static Methods .....	682
identity .....	682
rotation .....	683
rotation .....	684
rotation .....	685
rotationX .....	686
rotationX .....	687
rotationY .....	688
rotationY .....	689
rotationZ .....	690
rotationZ .....	691
rotationZYX .....	692
scale .....	693
translation .....	694
Public Instance Methods .....	695
getCol .....	695
getCol0 .....	696
getCol1 .....	697
getCol2 .....	698
getCol3 .....	699
getElem .....	700
getRow .....	701
getTranslation .....	702
getUpper3x3 .....	703
setCol .....	704
setCol0 .....	705
setCol1 .....	706
setCol2 .....	707
setCol3 .....	708
setElem .....	709
setElem .....	710
setRow .....	711
setTranslation .....	712
setUpper3x3 .....	713
<b>sce::Vectormath::Simd::Aos::VecIdx .....</b>	<b>714</b>
Summary .....	715
sce::Vectormath::Simd::Aos::VecIdx .....	715
Constructors and Destructors .....	716
VecIdx .....	716
VecIdx .....	717
Operator Methods .....	718
operator float .....	718

SCE CONFIDENTIAL

operator floatInVec .....	719
operator*= .....	720
operator*= .....	721
operator+= .....	722
operator+= .....	723
operator-= .....	724
operator-= .....	725
operator/= .....	726
operator/= .....	727
operator= .....	728
operator= .....	729
operator= .....	730
Public Instance Methods .....	731
getAsFloat .....	731
<b>sce::Vectormath::Simd::Aos::Vector2 .....</b>	<b>732</b>
Summary .....	733
sce::Vectormath::Simd::Aos::Vector2 .....	733
Constructors and Destructors .....	735
Vector2 .....	735
Vector2 .....	736
Vector2 .....	737
Vector2 .....	738
Vector2 .....	739
Vector2 .....	740
Vector2 .....	741
Vector2 .....	742
Operator Methods .....	743
operator vec_float2_ext .....	743
operator* .....	744
operator* .....	745
operator*= .....	746
operator*= .....	747
operator+ .....	748
operator+= .....	749
operator- .....	750
operator- .....	751
operator-= .....	752
operator/ .....	753
operator/ .....	754
operator/= .....	755
operator/= .....	756
operator= .....	757
operator[] .....	758
Public Static Methods .....	759
xAxis .....	759
yAxis .....	760
zero .....	761

SCE CONFIDENTIAL

Public Instance Methods .....	762
get64 .....	762
getElem .....	763
getExtVector .....	764
getFastVectorType .....	765
getX .....	766
getY .....	767
set64 .....	768
setElem .....	769
setElem .....	770
setExtVector .....	771
setFastVectorType .....	772
setX .....	773
setX .....	774
setY .....	775
setY .....	776
Protected Instance Methods .....	777
get64Ref.....	777
<b>sce::Vectormath::Simd::Aos::Vector3 .....</b>	<b>778</b>
Summary .....	779
sce::Vectormath::Simd::Aos::Vector3.....	779
Constructors and Destructors .....	781
Vector3 .....	781
Vector3 .....	782
Vector3 .....	783
Vector3 .....	784
Vector3 .....	785
Vector3 .....	786
Vector3 .....	787
Vector3 .....	788
Vector3 .....	789
Vector3 .....	790
Vector3 .....	791
Operator Methods .....	792
operator vec_float3_ext .....	792
operator* .....	793
operator* .....	794
operator*= .....	795
operator*= .....	796
operator+ .....	797
operator+ .....	798
operator+= .....	799
operator- .....	800
operator- .....	801
operator-= .....	802
operator/ .....	803
operator/ .....	804
operator/= .....	805

SCE CONFIDENTIAL

operator/=.....	806
operator=.....	807
operator[].....	808
operator[].....	809
Public Static Methods .....	810
xAxis.....	810
yAxis.....	811
zAxis.....	812
zero .....	813
Public Instance Methods .....	814
get128 .....	814
getElem .....	815
getExtVector.....	816
getX .....	817
getXY .....	818
getY .....	819
getZ .....	820
set128 .....	821
setElem .....	822
setExtVector.....	824
setX .....	825
setX .....	826
setXY.....	827
setY .....	828
setY .....	829
setZ .....	830
setZ .....	831
Protected Instance Methods .....	832
get128Ref.....	832
<b>sce::Vectormath::Simd::Aos::Vector4 .....</b>	<b>833</b>
Summary .....	834
sce::Vectormath::Simd::Aos::Vector4.....	834
Constructors and Destructors .....	836
Vector4 .....	836
Vector4 .....	837
Vector4 .....	838
Vector4 .....	839
Vector4 .....	840
Vector4 .....	841
Vector4 .....	842
Vector4 .....	843
Vector4 .....	844
Vector4 .....	845
Vector4 .....	846
Vector4 .....	847
Vector4 .....	848
Vector4 .....	849

SCE CONFIDENTIAL

Vector4 .....	850
Vector4 .....	851
Operator Methods .....	852
operator vec_float4_ext .....	852
operator* .....	853
operator* .....	854
operator*= .....	855
operator*= .....	856
operator+ .....	857
operator+= .....	858
operator- .....	859
operator- .....	860
operator-= .....	861
operator/ .....	862
operator/ .....	863
operator/= .....	864
operator/= .....	865
operator= .....	866
operator[] .....	867
operator[] .....	868
Public Static Methods .....	869
wAxis .....	869
xAxis .....	870
yAxis .....	871
zAxis .....	872
zero .....	873
Public Instance Methods .....	874
get128 .....	874
getElem .....	875
getExtVector .....	876
getW .....	877
getX .....	878
getXY .....	879
getXYZ .....	880
getY .....	881
getZ .....	882
set128 .....	883
setElem .....	884
setElem .....	885
setExtVector .....	886
setW .....	887
setW .....	888
setX .....	889
setX .....	890
setXY .....	891
setXYZ .....	892
setY .....	893
setY .....	894

SCE CONFIDENTIAL

setZ .....	895
setZ .....	896
Protected Instance Methods .....	897
get128Ref.....	897
<b>sce::Vectormath::Simd::boollnVec.....</b>	<b>898</b>
Summary .....	899
sce::Vectormath::Simd::boollnVec .....	899
Constructors and Destructors .....	900
boollnVec .....	900
boollnVec .....	901
boollnVec .....	902
boollnVec .....	903
boollnVec .....	904
boollnVec .....	905
boollnVec .....	906
boollnVec .....	907
Operator Methods .....	908
operator bool .....	908
operator!.....	909
operator&= .....	910
operator=.....	911
operator^= .....	912
operator =.....	913
Public Instance Methods .....	914
get128 .....	914
get64 .....	915
getAsBool.....	916
getFastVectorType .....	917
set128 .....	918
set64 .....	919
setFastVectorType .....	920
<b>sce::Vectormath::Simd::floatlnVec.....</b>	<b>921</b>
Summary .....	922
sce::Vectormath::Simd::floatlnVec .....	922
Constructors and Destructors .....	923
floatlnVec .....	923
floatlnVec .....	924
floatlnVec .....	925
floatlnVec .....	926
floatlnVec .....	927
floatlnVec .....	928
floatlnVec .....	929
floatlnVec .....	930
Operator Methods .....	931
operator float .....	931
operator*= .....	932
operator++ .....	933
operator++ .....	934

SCE CONFIDENTIAL

operator+=.....	935
operator-.....	936
operator-- .....	937
operator-- .....	938
operator-=.....	939
operator/=.....	940
operator=.....	941
Public Instance Methods .....	942
get128 .....	942
get64 .....	943
getAsFloat.....	944
getFastVectorType .....	945
set128 .....	946
set64 .....	947
setFastVectorType .....	948
<b>sce::Vectormath::Simd::Soa .....</b>	<b>949</b>
Summary .....	950
sce::Vectormath::Simd::Soa.....	950
Type Definitions .....	956
Matrix2_arg .....	956
Matrix3_arg .....	957
Matrix4_arg .....	958
Point3_arg .....	959
Quat_arg .....	960
Transform3_arg.....	961
Vector2_arg.....	962
Vector3_arg.....	963
Vector4_arg.....	964
2D Vector Functions .....	965
absPerElem.....	965
allElemEqual .....	966
allElemGreaterThan .....	967
allElemGreaterThanOrEqual.....	968
allElemLessThan.....	969
allElemLessThanOrEqual .....	970
allElemNotEqual.....	971
angle .....	972
angleBetween .....	973
clampPerElem.....	974
copySignPerElem.....	975
determinant .....	976
divPerElem.....	977
dot .....	978
length .....	979
lengthSqr .....	980
lerp .....	981
loadXYArray .....	982
maxElem .....	983

SCE CONFIDENTIAL

maxPerElem.....	984
minElem .....	985
minPerElem.....	986
mulPerElem.....	987
normalize.....	988
operator* .....	989
perp .....	990
print .....	991
print .....	992
recipPerElem.....	993
rotate .....	994
rsqrtPerElem .....	995
select .....	996
slerp .....	997
sqrtPerElem .....	998
storeHalfFloats .....	999
storeXYZArray .....	1000
sum.....	1001
<b>3D Vector Functions .....</b>	<b>1002</b>
absPerElem.....	1002
allElemEqual .....	1003
allElemGreaterThan.....	1004
allElemGreaterThanOrEqual.....	1005
allElemLessThan.....	1006
allElemLessThanOrEqual .....	1007
allElemNotEqual.....	1008
clampPerElem.....	1009
copySignPerElem.....	1010
cross.....	1011
crossMatrix.....	1012
crossMatrixMul .....	1013
divPerElem.....	1014
dot .....	1015
length .....	1016
lengthSqr .....	1017
lerp .....	1018
loadXYZArray .....	1019
maxElem .....	1020
maxPerElem.....	1021
minElem .....	1022
minPerElem.....	1023
mulPerElem.....	1024
normalize.....	1025
operator* .....	1026
outer .....	1027
print .....	1028
print .....	1029
recipPerElem.....	1030

SCE CONFIDENTIAL

rowMul.....	1031
rsqrtPerElem .....	1032
select.....	1033
slerp .....	1034
sqrtPerElem .....	1035
storeHalfFloats .....	1036
storeXYZArray.....	1037
sum.....	1038
<b>4D Vector Functions .....</b>	<b>1039</b>
absPerElem.....	1039
allElemEqual .....	1040
allElemGreaterThan .....	1041
allElemGreaterThanOrEqual.....	1042
allElemLessThan.....	1043
allElemLessThanOrEqual .....	1044
allElemNotEqual.....	1045
clampPerElem.....	1046
copySignPerElem.....	1047
divPerElem.....	1048
dot .....	1049
length .....	1050
lengthSqr.....	1051
lerp .....	1052
maxElem .....	1053
maxPerElem.....	1054
minElem .....	1055
minPerElem.....	1056
mulPerElem.....	1057
normalize.....	1058
operator* .....	1059
outer .....	1060
print .....	1061
print .....	1062
recipPerElem.....	1063
rsqrtPerElem .....	1064
select.....	1065
slerp .....	1066
sqrtPerElem .....	1067
storeHalfFloats .....	1068
sum.....	1069
<b>3D Point Functions .....</b>	<b>1070</b>
absPerElem.....	1070
allElemEqual .....	1071
allElemGreaterThan .....	1072
allElemGreaterThanOrEqual.....	1073
allElemLessThan.....	1074
allElemLessThanOrEqual .....	1075
allElemNotEqual.....	1076

SCE CONFIDENTIAL

clampPerElem.....	1077
copySignPerElem.....	1078
dist.....	1079
distFromOrigin.....	1080
distSqr .....	1081
distSqrFromOrigin .....	1082
divPerElem.....	1083
lerp .....	1084
loadXYZArray.....	1085
maxElem .....	1086
maxPerElem.....	1087
minElem .....	1088
minPerElem.....	1089
mulPerElem.....	1090
print .....	1091
print .....	1092
projection.....	1093
recipPerElem.....	1094
rsqrtPerElem .....	1095
scale.....	1096
scale.....	1097
select.....	1098
sqrtPerElem .....	1099
storeHalfFloats .....	1100
storeXYZArray.....	1101
sum.....	1102
Quaternion Functions.....	1103
conj.....	1103
dot .....	1104
length .....	1105
lengthSqr.....	1106
lerp .....	1107
norm .....	1108
normalize.....	1109
operator* .....	1110
print .....	1111
print .....	1112
rotate .....	1113
select.....	1114
slerp .....	1115
squad .....	1116
2x2 Matrix Functions .....	1117
absPerElem.....	1117
appendScale .....	1118
determinant .....	1119
inverse.....	1120
mulPerElem.....	1121
operator* .....	1122

prependScale .....	1123
print .....	1124
print .....	1125
select .....	1126
transpose .....	1127
3x3 Matrix Functions .....	1128
absPerElem.....	1128
appendScale .....	1129
determinant .....	1130
inverse.....	1131
mulPerElem.....	1132
operator* .....	1133
prependScale .....	1134
print .....	1135
print .....	1136
select .....	1137
transpose .....	1138
4x4 Matrix Functions .....	1139
absPerElem.....	1139
affineInverse.....	1140
appendScale .....	1141
determinant .....	1142
inverse.....	1143
mulPerElem.....	1144
operator* .....	1145
orthoInverse .....	1146
prependScale .....	1147
print .....	1148
print .....	1149
select .....	1150
transpose .....	1151
3x4 Transformation Matrix Functions .....	1152
absPerElem.....	1152
appendScale .....	1153
inverse.....	1154
mulPerElem.....	1155
orthoInverse .....	1156
prependScale .....	1157
print .....	1158
print .....	1159
select .....	1160
<b>sce::Vectormath::Simd::Soa::Matrix2 .....</b>	<b>1161</b>
Summary .....	1162
sce::Vectormath::Simd::Soa::Matrix2 .....	1162
Constructors and Destructors .....	1163
Matrix2 .....	1163
Matrix2 .....	1164
Matrix2 .....	1165

SCE CONFIDENTIAL

Matrix2 .....	1166
Matrix2 .....	1167
Matrix2 .....	1168
Operator Methods .....	1169
operator* .....	1169
operator* .....	1170
operator* .....	1171
operator*= .....	1172
operator*= .....	1173
operator+ .....	1174
operator+= .....	1175
operator- .....	1176
operator- .....	1177
operator-= .....	1178
operator= .....	1179
operator[] .....	1180
operator[] .....	1181
Public Static Methods .....	1182
identity .....	1182
rotation .....	1183
scale .....	1184
zero .....	1185
Public Instance Methods .....	1186
get4Aos .....	1186
getCol .....	1187
getCol0 .....	1188
getCol1 .....	1189
getElem .....	1190
getRow .....	1191
setCol .....	1192
setCol0 .....	1193
setCol1 .....	1194
setElem .....	1195
setRow .....	1196
<b>sce::Vectormath::Simd::Soa::Matrix3 .....</b>	<b>1197</b>
Summary .....	1198
sce::Vectormath::Simd::Soa::Matrix3 .....	1198
Constructors and Destructors .....	1200
Matrix3 .....	1200
Matrix3 .....	1201
Matrix3 .....	1202
Matrix3 .....	1203
Matrix3 .....	1204
Matrix3 .....	1205
Matrix3 .....	1206
Operator Methods .....	1207
operator* .....	1207
operator* .....	1208

SCE CONFIDENTIAL

operator* .....	1209
operator*= .....	1210
operator*= .....	1211
operator+ .....	1212
operator+= .....	1213
operator- .....	1214
operator- .....	1215
operator-= .....	1216
operator= .....	1217
operator[] .....	1218
operator[] .....	1219
Public Static Methods .....	1220
identity .....	1220
rotation .....	1221
rotation .....	1222
rotationX .....	1223
rotationY .....	1224
rotationZ .....	1225
rotationZYX .....	1226
scale .....	1227
zero .....	1228
Public Instance Methods .....	1229
get4Aos .....	1229
getCol .....	1230
getCol0 .....	1231
getCol1 .....	1232
getCol2 .....	1233
getElem .....	1234
getRow .....	1235
setCol .....	1236
setCol0 .....	1237
setCol1 .....	1238
setCol2 .....	1239
setElem .....	1240
setRow .....	1241
<b>sce::Vectormath::Simd::Soa::Matrix4 .....</b>	<b>1242</b>
Summary .....	1243
sce::Vectormath::Simd::Soa::Matrix4 .....	1243
Constructors and Destructors .....	1245
Matrix4 .....	1245
Matrix4 .....	1246
Matrix4 .....	1247
Matrix4 .....	1248
Matrix4 .....	1249
Matrix4 .....	1250
Matrix4 .....	1251
Matrix4 .....	1252
Matrix4 .....	1253

SCE CONFIDENTIAL

Operator Methods .....	1254
operator* .....	1254
operator* .....	1255
operator* .....	1256
operator* .....	1257
operator* .....	1258
operator* .....	1259
operator*= .....	1260
operator*= .....	1261
operator+= .....	1262
operator+ .....	1263
operator+= .....	1264
operator- .....	1265
operator- .....	1266
operator-= .....	1267
operator= .....	1268
operator[] .....	1269
operator[] .....	1270
Public Static Methods .....	1271
frustum .....	1271
identity .....	1272
lookAt .....	1273
orthographic .....	1274
perspective .....	1275
rotation .....	1276
rotation .....	1277
rotationX .....	1278
rotationY .....	1279
rotationZ .....	1280
rotationZYX .....	1281
scale .....	1282
translation .....	1283
zero .....	1284
Public Instance Methods .....	1285
get4Aos .....	1285
getCol .....	1286
getCol0 .....	1287
getCol1 .....	1288
getCol2 .....	1289
getCol3 .....	1290
getElem .....	1291
getRow .....	1292
getTranslation .....	1293
getUpper3x3 .....	1294
setCol .....	1295
setCol0 .....	1296
setCol1 .....	1297
setCol2 .....	1298

setCol3 .....	1299
setElem .....	1300
setRow .....	1301
setTranslation .....	1302
setUpper3x3 .....	1303
<b>sce::Vectormath::Simd::Soa::Point3.....</b>	<b>1304</b>
Summary .....	1305
sce::Vectormath::Simd::Soa::Point3 .....	1305
Constructors and Destructors .....	1306
Point3 .....	1306
Point3 .....	1307
Point3 .....	1308
Point3 .....	1309
Point3 .....	1310
Point3 .....	1311
Point3 .....	1312
Point3 .....	1313
Operator Methods .....	1314
operator+ .....	1314
operator+= .....	1315
operator- .....	1316
operator- .....	1317
operator-= .....	1318
operator= .....	1319
operator[] .....	1320
operator[] .....	1321
Public Static Methods .....	1322
origin .....	1322
Public Instance Methods .....	1323
get4Aos .....	1323
getElem .....	1324
getX .....	1325
getXY .....	1326
getY .....	1327
getZ .....	1328
setElem .....	1329
setX .....	1330
setXY .....	1331
setY .....	1332
setZ .....	1333
<b>sce::Vectormath::Simd::Soa::Quat.....</b>	<b>1334</b>
Summary .....	1335
sce::Vectormath::Simd::Soa::Quat .....	1335
Constructors and Destructors .....	1337
Quat .....	1337
Quat .....	1338
Quat .....	1339
Quat .....	1340

SCE CONFIDENTIAL

Quat .....	1341
Quat .....	1342
Quat .....	1343
Quat .....	1344
Quat .....	1345
Operator Methods .....	1346
operator* .....	1346
operator* .....	1347
operator*= .....	1348
operator*= .....	1349
operator+ .....	1350
operator+= .....	1351
operator- .....	1352
operator- .....	1353
operator-= .....	1354
operator/ .....	1355
operator/= .....	1356
operator= .....	1357
operator[] .....	1358
operator[] .....	1359
Public Static Methods .....	1360
axisAngle .....	1360
euler .....	1361
identity .....	1362
rotation .....	1363
rotation .....	1364
rotation .....	1365
rotationX .....	1366
rotationY .....	1367
rotationZ .....	1368
zero .....	1369
Public Instance Methods .....	1370
get4Aos .....	1370
getElem .....	1371
getW .....	1372
getX .....	1373
getXY .....	1374
getXYZ .....	1375
getY .....	1376
getZ .....	1377
setElem .....	1378
setW .....	1379
setX .....	1380
setXY .....	1381
setXYZ .....	1382
setY .....	1383
setZ .....	1384

<b>sce::Vectormath::Simd::Soa::Transform3.....</b>	<b>1385</b>
Summary .....	1386
sce::Vectormath::Simd::Soa::Transform3 .....	1386
Constructors and Destructors .....	1388
Transform3 .....	1388
Transform3 .....	1389
Transform3 .....	1390
Transform3 .....	1391
Transform3 .....	1392
Transform3 .....	1393
Transform3 .....	1394
Transform3 .....	1395
Operator Methods .....	1396
operator* .....	1396
operator* .....	1397
operator* .....	1398
operator*= .....	1399
operator= .....	1400
operator[] .....	1401
operator[] .....	1402
Public Static Methods .....	1403
identity .....	1403
rotation .....	1404
rotation .....	1405
rotationX .....	1406
rotationY .....	1407
rotationZ .....	1408
rotationZYX .....	1409
scale .....	1410
translation .....	1411
Public Instance Methods .....	1412
get4Aos .....	1412
getCol .....	1413
getCol0 .....	1414
getCol1 .....	1415
getCol2 .....	1416
getCol3 .....	1417
getElem .....	1418
getRow .....	1419
getTranslation .....	1420
getUpper3x3 .....	1421
setCol .....	1422
setCol0 .....	1423
setCol1 .....	1424
setCol2 .....	1425
setCol3 .....	1426
setElem .....	1427
setRow .....	1428

SCE CONFIDENTIAL

setTranslation.....	1429
setUpUpper3x3.....	1430
<b>sce::Vectormath::Simd::Soa::Vector2.....</b>	<b>1431</b>
Summary .....	1432
sce::Vectormath::Simd::Soa::Vector2 .....	1432
Constructors and Destructors .....	1433
Vector2 .....	1433
Vector2 .....	1434
Vector2 .....	1435
Vector2 .....	1436
Vector2 .....	1437
Vector2 .....	1438
Operator Methods .....	1439
operator* .....	1439
operator*= .....	1440
operator+ .....	1441
operator+= .....	1442
operator- .....	1443
operator- .....	1444
operator-= .....	1445
operator/ .....	1446
operator/= .....	1447
operator= .....	1448
operator[] .....	1449
operator[] .....	1450
Public Static Methods .....	1451
xAxis.....	1451
yAxis.....	1452
zero .....	1453
Public Instance Methods.....	1454
get4Aos .....	1454
getElem .....	1455
getX .....	1456
getY .....	1457
setElem .....	1458
setX .....	1459
setY .....	1460
<b>sce::Vectormath::Simd::Soa::Vector3.....</b>	<b>1461</b>
Summary .....	1462
sce::Vectormath::Simd::Soa::Vector3 .....	1462
Constructors and Destructors .....	1464
Vector3 .....	1464
Vector3 .....	1465
Vector3 .....	1466
Vector3 .....	1467
Vector3 .....	1468
Vector3 .....	1469
Vector3 .....	1470

Vector3 .....	1471
Operator Methods .....	1472
operator* .....	1472
operator*= .....	1473
operator+ .....	1474
operator+.. .....	1475
operator+= .....	1476
operator- .....	1477
operator-.. .....	1478
operator-= .....	1479
operator/ .....	1480
operator/= .....	1481
operator= .....	1482
operator[] .....	1483
operator[] .....	1484
Public Static Methods .....	1485
xAxis .....	1485
yAxis .....	1486
zAxis .....	1487
zero .....	1488
Public Instance Methods .....	1489
get4Aos .....	1489
getElem .....	1490
getX .....	1491
getXY .....	1492
getY .....	1493
getZ .....	1494
setElem .....	1495
setX .....	1496
setXY .....	1497
setY .....	1498
setZ .....	1499
<b>sce::Vectormath::Simd::Soa::Vector4.....</b>	<b>1500</b>
Summary .....	1501
sce::Vectormath::Simd::Soa::Vector4 .....	1501
Constructors and Destructors .....	1503
Vector4 .....	1503
Vector4 .....	1504
Vector4 .....	1505
Vector4 .....	1506
Vector4 .....	1507
Vector4 .....	1508
Vector4 .....	1509
Vector4 .....	1510
Vector4 .....	1511
Vector4 .....	1512
Vector4 .....	1513
Vector4 .....	1514

SCE CONFIDENTIAL

Operator Methods .....	1515
operator* .....	1515
operator*= .....	1516
operator+ .....	1517
operator+= .....	1518
operator- .....	1519
operator-.. .....	1520
operator-= .....	1521
operator/ .....	1522
operator/= .....	1523
operator= .....	1524
operator[] .....	1525
operator[] .....	1526
Public Static Methods .....	1527
wAxis .....	1527
xAxis .....	1528
yAxis .....	1529
zAxis .....	1530
zero .....	1531
Public Instance Methods .....	1532
get4Aos .....	1532
getElem .....	1533
getW .....	1534
getX .....	1535
getXY .....	1536
getXYZ .....	1537
getY .....	1538
getZ .....	1539
setElem .....	1540
setW .....	1541
setX .....	1542
setXY .....	1543
setXYZ .....	1544
setY .....	1545
setZ .....	1546
<b>Internal Functions .....</b>	<b>1547</b>
Absolute Value .....	1548
Functions .....	1548
Addition .....	1550
Functions .....	1550
Bit-Shift Left (Logical per Vector Lane) .....	1552
Functions .....	1552
Bit-Shift Left (Logical) .....	1554
Functions .....	1554
Bit-Shift Left (Whole Vector Logical) .....	1556
Functions .....	1556
Bit-Shift Right (Arithmetic per Vector Lane) .....	1557
Functions .....	1557

SCE CONFIDENTIAL

Bit-Shift Right (Logical per Vector Lane) .....	1559
Functions.....	1559
Bit-Shift Right (Logical).....	1561
Functions.....	1561
Bit-Shift Right (Whole Vector Logical).....	1563
Functions.....	1563
Bit-Shifting Right (Arithmetic) .....	1564
Functions.....	1564
Bitwise AND.....	1566
Functions.....	1566
Bitwise AND (with scalar value) .....	1569
Functions.....	1569
Bitwise COMPLEMENT .....	1571
Functions.....	1571
Bitwise COMPLEMENT AND .....	1573
Functions.....	1573
Bitwise COMPLEMENT OR .....	1576
Functions.....	1576
Bitwise OR.....	1579
Functions.....	1579
Bitwise XOR .....	1582
Functions.....	1582
Boolean Check (All Lanes True - Bool Result).....	1585
Functions.....	1585
Boolean Check (All Lanes True) .....	1586
Functions.....	1586
Boolean Check (Any Lane True) .....	1587
Functions.....	1587
Boolean Check (Any Lanes True - Bool Result) .....	1588
Functions.....	1588
Boolean Check (Partial Lanes All True) .....	1589
Functions.....	1589
Boolean Check (Partial Lanes Any True) .....	1590
Functions.....	1590
Byte Packing .....	1591
Functions.....	1591
Byte Unpacking .....	1592
Functions.....	1592
Clearing .....	1593
Functions.....	1593
Combination .....	1595
Functions.....	1595
Comparing (Per-Lane Equal) .....	1597
Functions.....	1597
Comparing (Per-Lane Greater-Than).....	1600
Functions.....	1600
Comparing (Per-Lane Greater-Than-Or-Equal) .....	1603
Functions.....	1603

SCE CONFIDENTIAL

Comparing (Per-Lane Less-Than-Or-Equal).....	1606
Functions.....	1606
Comparing (Per-Lane Not-Equal) .....	1609
Functions.....	1609
Comparing (Per-lane Less-Than).....	1612
Functions.....	1612
Conversion (Comparison Result to 0.0/1.0 Single-Precision Floating Point Vector) .....	1615
Functions.....	1615
Conversion (Comparison Result to Double-Precision Floating Point Vector).....	1616
Functions.....	1616
Conversion (Floating Point Vector to 32-bit Signed Integer Vector) .....	1617
Functions.....	1617
Conversion (Floating Point Vector to 32-bit Unsigned Integer Vector) .....	1618
Functions.....	1618
Conversion (Half-Precision Floating Point Vector to Single-Precision Floating Point Vector) .....	1619
Functions.....	1619
Conversion (Single-Precision Floating Point Vector to Half-Precision Floating Point Vector) .....	1620
Functions.....	1620
Conversion (Vector to Double-Precision Floating Point Vector) .....	1621
Functions.....	1621
Conversion (Vector to Single-Precision Floating Point Vector).....	1623
Functions.....	1623
Copy Sign.....	1624
Functions.....	1624
Counting (Leading Zero Bits) .....	1625
Functions.....	1625
Counting (Set Bits) .....	1626
Functions.....	1626
Counting (Trailing Zero Bits) .....	1627
Functions.....	1627
Creation (From Transposed Lanes) .....	1628
Functions.....	1628
Creation (From Vector Lower Half) .....	1632
Functions.....	1632
Creation (From Vector Upper Half) .....	1634
Functions.....	1634
Cross Product .....	1636
Functions.....	1636
Division .....	1637
Functions.....	1637
Dot Product .....	1638
Functions.....	1638
Dot Product (All Lane Splatting) .....	1639
Functions.....	1639
Endian Swapping .....	1640
Functions.....	1640
Endian Swapping (Big Endian to Native Endian).....	1642
Functions.....	1642

SCE CONFIDENTIAL

Endian Swapping (Little Endian to Native Endian) .....	1644
Functions.....	1644
Endian Swapping (Native Endian to Big Endian).....	1646
Functions.....	1646
Endian Swapping (Native Endian to Little Endian) .....	1648
Functions.....	1648
Extraction .....	1650
Functions.....	1650
Floating Point Exponent Extraction.....	1653
Functions.....	1653
Floating Point Sign Bit Splatting .....	1654
Functions.....	1654
Geometric Length.....	1655
Functions.....	1655
Geometric Partial Length .....	1656
Functions.....	1656
Horizontal Maximum Value.....	1657
Functions.....	1657
Horizontal Minimum Value.....	1659
Functions.....	1659
Horizontal Partial Maximum Value .....	1661
Functions.....	1661
Horizontal Partial Minimum Value .....	1663
Functions.....	1663
Horizontal Partial Sum.....	1665
Functions.....	1665
Horizontal Sum.....	1667
Functions.....	1667
Insertion.....	1669
Functions.....	1669
Insertion (Take From First Lane).....	1672
Functions.....	1672
Integer Narrowing by Discarding Upper Bits .....	1675
Functions.....	1675
Integer Narrowing by Value Saturation .....	1677
Functions.....	1677
Lane Order Reversal.....	1679
Functions.....	1679
Lane Order Reversal (Big to Native) .....	1681
Functions.....	1681
Lane Order Reversal (Little to Native) .....	1683
Functions.....	1683
Lane Order Reversal (Native to Big) .....	1685
Functions.....	1685
Lane Order Reversal (Native to Little).....	1687
Functions.....	1687
Lane Shifting (Create from Appended).....	1689
Functions.....	1689

SCE CONFIDENTIAL

Lane Shifting (Create from Rotated) .....	1692
Functions.....	1692
Loading (Scalar Aligned) .....	1695
Functions.....	1695
Loading (Aligned with Lane Counts) .....	1697
Functions.....	1697
Loading (Aligned) .....	1699
Functions.....	1699
Loading (Four Sequential Aligned Vectors).....	1701
Functions.....	1701
Loading (No Alignment).....	1702
Functions.....	1702
Multiplication.....	1704
Functions.....	1704
Multiplication (Followed by Addition) .....	1706
Functions.....	1706
Multiplication (Followed by Subtraction).....	1708
Functions.....	1708
Multiplication (Product Subtracted) .....	1710
Functions.....	1710
Negation .....	1712
Functions.....	1712
Per-Lane Bit Test.....	1714
Functions.....	1714
Per-Lane Maximum Value .....	1716
Functions.....	1716
Per-Lane Minimum Value .....	1719
Functions.....	1719
Point Mantissa Extraction.....	1722
Functions.....	1722
Reciprocal .....	1723
Functions.....	1723
Reciprocal Square Root.....	1724
Functions.....	1724
Selection (Per-Bit) .....	1725
Functions.....	1725
Selection (Per-Lane) .....	1728
Functions.....	1728
Short Packing .....	1731
Functions.....	1731
Short Unpacking.....	1732
Functions.....	1732
Shuffle (128-bit) .....	1733
Functions.....	1733
Shuffle (64-bit) .....	1747
Functions.....	1747
Shuffle (Template Arguments) .....	1749
Functions.....	1749

SCE CONFIDENTIAL

Shuffling .....	1751
Functions.....	1751
Signed Integer Sign Bit Extension.....	1752
Functions.....	1752
Splatting (Using a Scalar Value).....	1754
Functions.....	1754
Splatting (Using an Index into Another Vector) .....	1756
Functions.....	1756
Square Root .....	1759
Functions.....	1759
Storing (4 Vector Transposition).....	1760
Functions.....	1760
Storing (Aligned - Eight Lanes Only).....	1761
Functions.....	1761
Storing (Aligned - First Lane Only).....	1762
Functions.....	1762
Storing (Aligned - Four Lanes Only).....	1765
Functions.....	1765
Storing (Aligned - Sixteen Lanes Only).....	1767
Functions.....	1767
Storing (Aligned - Three Lanes Only).....	1768
Functions.....	1768
Storing (Scalar Type Aligned - First Lane Only).....	1770
Functions.....	1770
Storing (Scalar Type Aligned - Two Lanes Only).....	1773
Functions.....	1773
Storing (Aligned).....	1775
Functions.....	1775
Storing (Aligned- Two Lanes Only).....	1778
Functions.....	1778
Storing (No Alignment - Eight Lanes Only).....	1780
Functions.....	1780
Storing (No Alignment - First Lane Only) .....	1781
Functions.....	1781
Storing (No Alignment - Four Lanes Only) .....	1784
Functions.....	1784
Storing (No Alignment - Sixteen Lanes Only).....	1786
Functions.....	1786
Storing (No Alignment - Three Lanes Only) .....	1787
Functions.....	1787
Storing (No Alignment - Two Lanes Only) .....	1789
Functions.....	1789
Storing (Scalar Type Aligned - Eight Lanes Only).....	1791
Functions.....	1791
Storing (Scalar Type Aligned - Four Lanes Only).....	1792
Functions.....	1792
Storing (Scalar Type Aligned - Sixteen Lanes Only).....	1794
Functions.....	1794

SCE CONFIDENTIAL

---

Storing (Scalar Type Aligned - Three Lanes Only) .....	1795
Functions.....	1795
Subtraction .....	1797
Functions.....	1797
Swizzle (128-bit).....	1800
Functions.....	1800
Swizzle (64-bit).....	1805
Functions.....	1805
Swizzle (Template Arguments).....	1807
Functions.....	1807
Swizzling .....	1809
Functions.....	1809
Transposition .....	1811
Functions.....	1811
Unsigned Integer Zero Extension.....	1813
Functions.....	1813

SCE CONFIDENTIAL

---

# **Introduction**

# Library Summary

## Library Contents

Item	Description
<a href="#">sce</a>	The namespace containing the sce framework.
<a href="#">sce::Vectormath</a>	The namespace containing the Vectormath library.
<a href="#">sce::Vectormath::Simd</a>	The namespace containing the Vectormath SIMD implementation.
<a href="#">sce::Vectormath::Simd::Aos</a>	The namespace containing array-of-structures (AoS) classes.
<a href="#">sce::Vectormath::Simd::Aos::Matrix2</a>	A 2x2 matrix in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Matrix3</a>	A 3x3 matrix in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Matrix4</a>	A 4x4 matrix in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Point3</a>	A 3D point in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Quat</a>	A quaternion in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Transform3</a>	A 3x4 transformation matrix in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::VecIdx</a>	A reference to an element in a SIMD vec_float4 vector.
<a href="#">sce::Vectormath::Simd::Aos::Vector2</a>	A 2D vector in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Vector3</a>	A 3D vector in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Vector4</a>	A 4D vector in array-of-structures format.
<a href="#">sce::Vectormath::Simd::boolInVec</a>	A class representing a scalar bool value contained in a vector register.
<a href="#">sce::Vectormath::Simd::floatInVec</a>	A class representing a scalar float value contained in a vector register.
<a href="#">sce::Vectormath::Simd::Soa</a>	The namespace containing structure-of-arrays (SoA) classes.
<a href="#">sce::Vectormath::Simd::Soa::Matrix2</a>	A set of four 2x2 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Matrix3</a>	A set of four 3x3 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Matrix4</a>	A set of four 4x4 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Point3</a>	A set of four 3D points in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Quat</a>	A set of four quaternions in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Transform3</a>	A set of four 3x4 transformation matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Vector2</a>	A set of four 2D vectors in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Vector3</a>	A set of four 3D vectors in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Vector4</a>	A set of four 4D vectors in structure-of-arrays format.

SCE CONFIDENTIAL

---

000004892117  
**sce**

# Summary

## sce

The namespace containing the sce framework.

### Definition

```
namespace sce { }
```

### Description

The namespace containing the sce framework.

### Inner Classes, Structures, and Namespaces

Item	Description
<a href="#">sce::Vectormath</a>	The namespace containing the <a href="#">Vectormath</a> library.

SCE CONFIDENTIAL

---

**sce::Vectormath**

# Summary

## sce::Vectormath

The namespace containing the [Vectormath](#) library.

### Definition

```
namespace Vectormath { }
```

### Description

The namespace containing the [Vectormath](#) library.

### Inner Classes, Structures, and Namespaces

Item	Description
<a href="#">sce::Vectormath::Simd</a>	The namespace containing the <a href="#">Vectormath</a> SIMD implementation.

**sce::Vectormath::Simd**

000004892117

# Summary

## sce::Vectormath::Simd

The namespace containing the [Vectormath](#) SIMD implementation.

### Definition

```
namespace Simd { }
```

### Description

The namespace containing the [Vectormath](#) SIMD implementation.

### Function Summary

Function	Description
<a href="#">acosf</a>	Compute the arc cosine of $x$ .
<a href="#">asinf</a>	Compute the arc sine of $x$ .
<a href="#">atan2f</a>	Compute the arc tangent of $y/x$ .
<a href="#">atanf</a>	Compute the arc tangent of $x$ .
<a href="#">cbrtf</a>	Compute the cube root of $x$ .
<a href="#">ceilf</a>	Compute the smallest integral value greater than or equal to $x$ .
<a href="#">clamp</a>	Clamp the value between the minimum and maximum values.
<a href="#">copysignf</a>	Compute the value with the magnitude of $x$ and the sign of $y$ .
<a href="#">cosf</a>	Compute the cosine of $x$ .
<a href="#">coshf</a>	Compute the hyperbolic cosine of $x$ .
<a href="#">divf</a>	Compute the quotient of $x/y$ .
<a href="#">exp2f</a>	Compute the base-2 exponential of $x$ .
<a href="#">expf</a>	Compute the natural exponential of $x$ .
<a href="#">expm1f</a>	Compute $\exp f(x) - 1$ .
<a href="#">fabsf</a>	Compute the absolute value of $x$ .
<a href="#">fdimf</a>	Compute the difference if positive, 0 otherwise.
<a href="#">floorf</a>	Compute the largest integral value less than or equal to $x$ .
<a href="#">fmaf</a>	Compute $(x*y)+z$ , rounded as one ternary operation.
<a href="#">fmaxf</a>	Compute the maximum value of two values.
<a href="#">fminf</a>	Compute the minimum value of two values.
<a href="#">fmodf</a>	Compute the remainder of $x/y$ .
<a href="#">hypotf</a>	Compute the Euclidean distance.
<a href="#">log10f</a>	Compute the base-10 logarithm of $x$ .
<a href="#">log1pf</a>	Compute $\log f(x+1)$ .
<a href="#">logbf</a>	Compute the exponent of $x$ .
<a href="#">logf</a>	Compute the natural logarithm of $x$ .
<a href="#">logtwof</a>	Compute the base-2 logarithm of $x$ .
<a href="#">modff</a>	Compute the integral and fractional parts of $x$ .
<a href="#">negatef</a>	Compute the negation of $x$ .
<a href="#">operator!=</a>	Not equal operator.
<a href="#">operator&amp;=</a>	Not equal operator.
<a href="#">operator&amp;</a>	And operator.
<a href="#">operator&amp;&amp;</a>	Logical And operator.
<a href="#">operator*</a>	Multiplication operator.
<a href="#">operator+</a>	Addition operator.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>operator-</u></a>	Subtraction operator.
<a href="#"><u>operator/</u></a>	Division operator.
<a href="#"><u>operator&lt;</u></a>	Less than operator.
<a href="#"><u>operator&lt;=</u></a>	Less than or equal operator.
<a href="#"><u>operator==</u></a>	Equal operator.
<a href="#"><u>operator==</u></a>	Equal operator.
<a href="#"><u>operator&gt;</u></a>	Greater than operator.
<a href="#"><u>operator&gt;=</u></a>	Greater than or equal operator.
<a href="#"><u>operator^</u></a>	Exclusive or operator.
<a href="#"><u>operator </u></a>	Or operator.
<a href="#"><u>operator  </u></a>	Logical Or operator.
<a href="#"><u>powf</u></a>	Compute $x$ raised to the power of $y$ .
<a href="#"><u>recipf</u></a>	Compute the reciprocal of $x$ .
<a href="#"><u>remainderf</u></a>	Compute the remainder of $x/y$ .
<a href="#"><u>rsqrtf</u></a>	Compute the reciprocal of the square root of $x$ .
<a href="#"><u>select</u></a>	Conditionally select between two values.
<a href="#"><u>select</u></a>	Conditionally select between two values.
<a href="#"><u>select</u></a>	Conditionally select between two values.
<a href="#"><u>sincosf</u></a>	Compute the sine and cosine of $x$ .
<a href="#"><u>sinf</u></a>	Compute the sine of $x$ .
<a href="#"><u>sinhf</u></a>	Compute the hyperbolic sine of $x$ .
<a href="#"><u>sqrtf</u></a>	Compute the square root of $x$ .
<a href="#"><u>tanf</u></a>	Compute the tangent of $x$ .
<a href="#"><u>tanhf</u></a>	Compute the hyperbolic tangent of $x$ .
<a href="#"><u>truncf</u></a>	Compute the integral value nearest to but no larger in magnitude than $x$ .

## Inner Classes, Structures, and Namespaces

Item	Description
<a href="#"><u>sce::Vectormath::Simd::Aos</u></a>	The namespace containing array-of-structures (AoS) classes.
<a href="#"><u>sce::Vectormath::Simd::boolInVec</u></a>	A class representing a scalar bool value contained in a vector register.
<a href="#"><u>sce::Vectormath::Simd::floatInVec</u></a>	A class representing a scalar float value contained in a vector register.
<a href="#"><u>sce::Vectormath::Simd::Soa</u></a>	The namespace containing structure-of-arrays (SoA) classes.

# Enumerated Types

## RotationOrder

Enum of valid Euler rotation orders.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            enum RotationOrder {
                kXYZ = 0,
                kYZX = 1,
                kZXY = 2,
                kXZY = 3,
                kYXZ = 4,
                kZYX = 5
            };
        }
    }
}
```

### Enumeration Values

Macro	Value	Description
kXYZ	0	Indicates a rotation around the x axis first, followed by a rotation around the y axis, followed by a rotation around the z axis.
kYZX	1	Indicates a rotation around the y axis first, followed by a rotation around the z axis, followed by a rotation around the x axis.
kZXY	2	Indicates a rotation around the z axis first, followed by a rotation around the x axis, followed by a rotation around the y axis.
kXZY	3	Indicates a rotation around the x axis first, followed by a rotation around the z axis, followed by a rotation around the y axis.
kYXZ	4	Indicates a rotation around the y axis first, followed by a rotation around the x axis, followed by a rotation around the z axis.
kZYX	5	Indicates a rotation around the z axis first, followed by a rotation around the y axis, followed by a rotation around the x axis.

### Description

Enum of valid Euler rotation orders.

# Type Definitions

## **boolInVec\_arg**

Type used when passing [boolInVec](#) as a function argument.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            typedef const boolInVec SCE_VECTORMATH SIMD_AOS_BOOLINVEC_ARG
            boolInVec_arg;
        }
    }
}
```

### **Description**

Type used when passing [boolInVec](#) as a function argument.

## **floatInVec\_arg**

Type used when passing [floatInVec](#) as a function argument.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            typedef const floatInVec SCE_VECTORMATH SIMD_AOS_FLOATINVEC_ARG
            floatInVec_arg;
        }
    }
}
```

### **Description**

Type used when passing [floatInVec](#) as a function argument.

# boolInVec Functions

## **operator!=**

Not equal operator.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator!=(  

                boolInVec arg vec0,  

                boolInVec arg vec1  

            );
        }
    }
}
```

### **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

### **Return Values**

True if *vec0* is not equal to *vec1*; otherwise false

### **Description**

Not equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator&**

And operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator&(
                boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is true and *vec1* is also true; otherwise false

## **Description**

And operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator&&**

Logical And operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator&&(
                boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is true and *vec1* is also true; otherwise false

## **Description**

Logical And operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator==**

---

Equal operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator==(boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is equal to *vec1*; otherwise false

## **Description**

Equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator^**

Exclusive or operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator^(
                boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is not equal to *vec1*; otherwise false

## **Description**

Exclusive or operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator|**

Or operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator|(
                boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is true or *vec1* is true; otherwise false

## **Description**

Or operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator||**

Logical Or operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator||(boolInVec arg vec0,
                boolInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is true or *vec1* is true; otherwise false

## **Description**

Logical Or operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# select

Conditionally select between two values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec select(
                boolInVec arg vec0,
                boolInVec arg vec1,
                boolInVec arg select_vec1
            );
        }
    }
}
```

## Arguments

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).
<i>select_vec1</i>	False selects <i>vec0</i> ; true selects <i>vec1</i> .

## Return Values

Equal to *vec1* if *select\_vec1* is true; otherwise equal to *vec0*

## Description

Conditionally select between two values.

## Notes

This function uses a conditional select instruction to avoid a branch.

# floatInVec Functions

## clamp

Clamp the value between the minimum and maximum values.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec clamp(
                floatInVec arg vec,
                floatInVec arg minVec,
                floatInVec arg maxVec
            );
        }
    }
}
```

### Arguments

<i>vec</i>	Scalar value (contained in a vector register).
<i>minVec</i>	Scalar minimum value (contained in a vector register).
<i>maxVec</i>	Scalar maximum value (contained in a vector register).

### Return Values

Equal to *minVec* if *vec* is less than *minVec*  
 Equal to *maxVec* if *vec* is greater than *maxVec*  
 Else the result is equal to *vec*

### Description

Clamp the value between the minimum and maximum values.

### Notes

Result is undefined if the minimum value is greater than the maximum value  
 This function uses a conditional select instruction to avoid a branch.

# **operator!=**

Not equal operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator!=(  

                floatInVec arg vec0,  

                floatInVec arg vec1  

            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is not equal to *vec1*; otherwise false

## **Description**

Not equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator\***

Multiplication operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator*(
                floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

Product of the specified values

## **Description**

Multiplication operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator+**

Addition operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator+
                (floatInVec arg vec0,
                 floatInVec arg vec1
                );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

Sum of the specified values

## **Description**

Addition operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator-**

Subtraction operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator-
                (floatInVec arg vec0,
                 floatInVec arg vec1
                );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

Difference of the specified values

## **Description**

Subtraction operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator/**

Division operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator/(
                floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

Quotient of the specified values

## **Description**

Division operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator<**

Less than operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator<(
                floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is less than *vec1*; otherwise false

## **Description**

Less than operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator<=**

Less than or equal operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator<=
                (floatInVec arg vec0,
                 floatInVec arg vec1
                );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is less than or equal to *vec1*; otherwise false

## **Description**

Less than or equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator==**

---

Equal operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator==(floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is equal to *vec1*; otherwise false

## **Description**

Equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator>**

Greater than operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator>(
                floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is greater than *vec1*; otherwise false

## **Description**

Greater than operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator>=**

Greater than or equal operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator>=(
                floatInVec arg vec0,
                floatInVec arg vec1
            );
        }
    }
}
```

## **Arguments**

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).

## **Return Values**

True if *vec0* is greater than or equal to *vec1*; otherwise false

## **Description**

Greater than or equal operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# select

Conditionally select between two values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec select(
                floatInVec arg vec0,
                floatInVec arg vec1,
                boolInVec arg select_vec1
            );
        }
    }
}
```

## Arguments

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).
<i>select_vec1</i>	False selects <i>vec0</i> ; true selects <i>vec1</i> .

## Return Values

Equal to *vec1* if *select\_vec1* is true; otherwise equal to *vec0*

## Description

Conditionally select between two values.

## Notes

This function uses a conditional select instruction to avoid a branch.

# select

Conditionally select between two values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE const floatInVec select(
                floatInVec arg vec0,
                floatInVec arg vec1,
                bool select1
            );
        }
    }
}
```

## Arguments

<i>vec0</i>	Scalar value (contained in a vector register).
<i>vec1</i>	Scalar value (contained in a vector register).
<i>select1</i>	False selects <i>vec0</i> ; true selects <i>vec1</i> .

## Return Values

Equal to *vec1* if *select1* is true; otherwise equal to *vec0*

## Description

Conditionally select between two values.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# floatInVec Math Functions

## acosf

Compute the arc cosine of  $x$ .

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec acosf(
                floatInVec arg_x
            );
        }
    }
}
```

### Arguments

$x$	Scalar value (contained in a vector register).
-----	--

### Return Values

Arc cosine of  $x$

### Description

Compute the arc cosine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

### Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a +/-INF value.
- A NaN value is returned if  $x$  is outside the range [-1, 1].

SCE CONFIDENTIAL

# asinf

Compute the arc sine of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec asinf(
                floatInVec arg_x
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
-----	--

## Return Values

Arc sine of  $x$

## Description

Compute the arc sine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a +/-INF value.
- A NaN value is returned if  $x$  is outside the range [-1, 1].
- A +/-0 value is returned if  $x$  is a +/-0 value.

# atan2f

Compute the arc tangent of  $y/x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec atan2f(
                floatInVec arg y,
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

<i>y</i>	Scalar value (contained in a vector register).
<i>x</i>	Scalar value (contained in a vector register).

## Return Values

Arc tangent of  $y/x$

## Description

Compute the arc tangent of  $y/x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if *x* or *y* is a NaN value.

SCE CONFIDENTIAL

# atanf

Compute the arc tangent of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec atanf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Arc tangent of  $x$

## Description

Compute the arc tangent of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A  $+/-0$  value is returned if  $x$  is a  $+/-0$  value.

A  $+/-\text{HalfPi}$  value is returned if  $x$  is a  $+/-\text{INF}$  value.

SCE CONFIDENTIAL

# **cbrtf**

Compute the cube root of  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec cbrtf(
                floatInVec arg x
            );
        }
    }
}
```

## **Arguments**

$x$	Scalar value (contained in a vector register).
-----	--

## **Return Values**

Cube root of $x$
------------------

## **Description**

Compute the cube root of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A +/-INF value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# ceilf

Compute the smallest integral value greater than or equal to  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec ceilf(
                floatInVec arg
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
-----	--

## Return Values

Smallest integral value greater than or equal to  $x$

## Description

Compute the smallest integral value greater than or equal to  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A +/-INF value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

# copysignf

Compute the value with the magnitude of  $x$  and the sign of  $y$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec copysignf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
$y$	Scalar value (contained in a vector register).

## Return Values

Value with the magnitude of  $x$  and the sign of  $y$

## Description

Compute the value with the magnitude of  $x$  and the sign of  $y$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A +/-NaN value is returned if  $x$  is a NaN value and  $y$  has a +/- sign.

SCE CONFIDENTIAL

# **cosp**

Compute the cosine of  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec cosp(
                floatInVec arg
            );
        }
    }
}
```

## **Arguments**

$x$  Scalar value (contained in a vector register).

## **Return Values**

Cosine of  $x$

## **Description**

Compute the cosine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A NaN value is returned if  $x$  is a +/-INF value.

SCE CONFIDENTIAL

# coshf

Compute the hyperbolic cosine of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec coshf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Hyperbolic cosine of  $x$

## Description

Compute the hyperbolic cosine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A +INF value is returned if  $x$  is a +/-INF value.

A +INF value is returned if the result overflows.

# divf

Compute the quotient of  $x/y$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec divf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
$y$	Scalar value (contained in a vector register).

## Return Values

Quotient of  $x/y$

## Description

Compute the quotient of  $x/y$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# exp2f

Compute the base-2 exponential of  $x$

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec exp2f(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Base-2 exponential of  $x$

## Description

Compute the base-2 exponential of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A +INF value is returned if  $x$  is a +INF value.
- A +0 value is returned if  $x$  is a -INF value.
- A +INF value is returned if the result overflows.
- A +0 value is returned if the result underflows.

SCE CONFIDENTIAL

# expf

Compute the natural exponential of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec expf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Natural exponential of  $x$

## Description

Compute the natural exponential of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A +INF value is returned if  $x$  is a +INF value.
- A +0 value is returned if  $x$  is a -INF value.
- A +INF value is returned if the result overflows.
- A +0 value is returned if the result underflows.

SCE CONFIDENTIAL

# **expm1f**

Compute  $\exp f(x) - 1$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec expm1f(
                floatInVec arg x
            );
        }
    }
}
```

## **Arguments**

*x* Scalar value (contained in a vector register).

## **Return Values**

$\exp f(x) - 1$

## **Description**

Compute  $\exp f(x) - 1$  accurately even for small values of *x*. Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

- A NaN value is returned if *x* is a NaN value.
- A +INF value is returned if *x* is a +INF value.
- A -1 value is returned if *x* is a -INF value.
- A +INF value is returned if the result overflows.
- A -1 value is returned if the result underflows.
- A +/-0 value is returned if *x* is a +/-0 value.

SCE CONFIDENTIAL

# fabsf

Compute the absolute value of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec fabsf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Absolute value of  $x$

## Description

Compute the absolute value of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A +INF value is returned if  $x$  is a +/-INF value.

A +0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# fdimf

Compute the difference if positive, 0 otherwise.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec fdimf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (contained in a vector register).
<i>y</i>	Scalar value (contained in a vector register).

## Return Values

Difference if positive, 0 otherwise

## Description

Compute the difference if positive, 0 otherwise. Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if *x* or *y* is a NaN value.

A +INF value is returned if the result overflows.

SCE CONFIDENTIAL

# **floorf**

Compute the largest integral value less than or equal to  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec floorf(
                floatInVec arg
            );
        }
    }
}
```

## **Arguments**

$x$	Scalar value (contained in a vector register).
-----	--

## **Return Values**

Largest integral value less than or equal to  $x$

## **Description**

Compute the largest integral value less than or equal to  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A +/-INF value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

# fmaf

Compute  $(x^*y)+z$ , rounded as one ternary operation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec fmaf(
                floatInVec arg x,
                floatInVec arg y,
                floatInVec arg z
            );
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (contained in a vector register).
<i>y</i>	Scalar value (contained in a vector register).
<i>z</i>	Scalar value (contained in a vector register).

## Return Values

$(x^*y)+z$ , rounded as one ternary operation

## Description

Compute  $(x^*y)+z$ , rounded as one ternary operation. Each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# fmaxf

Compute the maximum value of two values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec fmaxf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

x	Scalar value (contained in a vector register).
y	Scalar value (contained in a vector register).

## Return Values

Maximum value of two values

## Description

Compute the maximum value of two values. Each operand is contained in its own vector register, and the result is contained in another vector register.

# fminf

Compute the minimum value of two values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec fminf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

x	Scalar value (contained in a vector register).
y	Scalar value (contained in a vector register).

## Return Values

Minimum value of two values

## Description

Compute the minimum value of two values. Each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# fmodf

Compute the remainder of  $x/y$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec fmodf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (contained in a vector register).
<i>y</i>	Scalar value (contained in a vector register).

## Return Values

Remainder of  $x/y$

## Description

Compute the remainder of  $x/y$ ; the result has the same sign as  $x$  and magnitude less than the magnitude of  $y$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  or  $y$  is a NaN value.
- A NaN value is returned if  $x$  is a +/-INF value.
- A NaN value is returned if  $y$  is a +/-0 value.
- A +/-0 value is returned if  $x$  is a +/-0 value and  $y$  is not a +/-0 value.

# hypotf

Compute the Euclidean distance.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec hypotf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (contained in a vector register).
<i>y</i>	Scalar value (contained in a vector register).

## Return Values

Euclidean distance

## Description

Compute the Euclidean distance, which is the same as  $\sqrt{x^*x+y^*y}$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if *x* or *y* is a NaN value.

A +INF value is returned if the result overflows.

SCE CONFIDENTIAL

# log10f

Compute the base-10 logarithm of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec log10f(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Base-10 logarithm of  $x$

## Description

Compute the base-10 logarithm of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a negative value (other than -0).
- A +INF value is returned if  $x$  is a +INF value.
- A -INF value is returned if  $x$  is a +/-0 value.
- A +0 value is returned if  $x$  is a +1 value.

SCE CONFIDENTIAL

# log1pf

Compute  $\log f(x+1)$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec log1pf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

*x* Scalar value (contained in a vector register).

## Return Values

$\log f(x+1)$

## Description

Compute  $\log f(x+1)$  accurately even for small values of *x*. Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if *x* is a NaN value.
- A NaN value is returned if *x* has a value less than -1.
- A +INF value is returned if *x* is a +INF value.
- A -INF value is returned if *x* has a value equal to -1.
- A +/-0 value is returned if *x* is a +/-0 value.

SCE CONFIDENTIAL

# logbf

Compute the exponent of  $x$

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec logbf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Exponent of  $x$

## Description

Compute the exponent of  $x$ , which is the same as `floorf(log2f(x))` if `FLT_RADIX` is equal to 2. Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A +INF value is returned if  $x$  is a +/-INF value.

A -INF value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# log

Compute the natural logarithm of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec logf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Natural logarithm of  $x$

## Description

Compute the natural logarithm of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a negative value (other than -0).
- A +INF value is returned if  $x$  is a +INF value.
- A -INF value is returned if  $x$  is a +/-0 value.
- A +0 value is returned if  $x$  is a +1 value.

SCE CONFIDENTIAL

# logtwof

Compute the base-2 logarithm of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec logtwof(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Base-2 logarithm of  $x$

## Description

Compute the base-2 logarithm of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a negative value (other than -0).
- A +INF value is returned if  $x$  is a +INF value.
- A -INF value is returned if  $x$  is a +/-0 value.
- A +0 value is returned if  $x$  is a +1 value.

SCE CONFIDENTIAL

# modff

Compute the integral and fractional parts of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec modff(
                floatInVec arg x,
                floatInVec *i
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
$i$	Pointer to a vector register (containing one float).

## Return Values

Integral and fractional parts of  $x$

## Description

Compute the integral and fractional parts of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned and  $i$  will be a NaN value if  $x$  is a NaN value.

A +/-0 value is returned and  $i$  will be a +/-INF value if  $x$  is a +/-INF value.

SCE CONFIDENTIAL

# **negatef**

Compute the negation of  $x$

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec negatef(
                floatInVec arg x
            );
        }
    }
}
```

## **Arguments**

$x$  Scalar value (contained in a vector register).

## **Return Values**

Negation of  $x$

## **Description**

Compute the negation of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

# powf

Compute  $x$  raised to the power of  $y$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec powf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
$y$	Scalar value (contained in a vector register).

## Return Values

$x$  raised to the power of  $y$

## Description

Compute  $x$  raised to the power of  $y$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A +1 value is returned if  $x$  is a +1 value ( $y$  is ignored).
- A +1 value is returned if  $y$  is a +/-0 value ( $x$  is ignored).
- A +0 value is returned if  $x$  has an absolute value greater than 1 and  $y$  is a -INF value.
- A +0 value is returned if  $x$  has an absolute value less than 1 and  $y$  is a +INF value.
- A +INF value is returned if  $x$  has an absolute value less than 1 and  $y$  is a -INF value.
- A +INF value is returned if  $x$  has an absolute value greater than 1 and  $y$  is a +INF value.
- A +INF value is returned if  $x$  has a +INF value  $y$  has a value greater than 0.
- A +0 value is returned if  $x$  has a +INF value  $y$  has a value less than 0.
- A -INF value is returned if  $x$  has a -INF value  $y$  has a value greater than 0 which is an odd integer value.
- A +INF value is returned if  $x$  has a -INF value  $y$  has a value greater than 0 which is not an odd integer value.
- A -0 value is returned if  $x$  has a -INF value  $y$  has a value less than 0 which is an odd integer value.
- A +0 value is returned if  $x$  has a -INF value  $y$  has a value less than 0 which is not an odd integer value.
- A NaN value is returned if  $x$  or  $y$  is a NaN value and no other cases apply.

SCE CONFIDENTIAL

# recipf

Compute the reciprocal of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec recipf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Reciprocal of  $x$

## Description

Compute the reciprocal of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# remainderf

Compute the remainder of  $x/y$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec remainderf(
                floatInVec arg x,
                floatInVec arg y
            );
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (contained in a vector register).
<i>y</i>	Scalar value (contained in a vector register).

## Return Values

Remainder of  $x/y$

## Description

Compute the remainder of  $x/y$ , where the quotient is the integral value nearest the exact value of  $x/y$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if *x* or *y* is a NaN value.
- A NaN value is returned if *x* is a +/-INF value.
- A NaN value is returned if *y* is a +/-0 value.

SCE CONFIDENTIAL

# rsqrtf

Compute the reciprocal of the square root of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec rsqrtf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Reciprocal of the square root of  $x$

## Description

Compute the reciprocal of the square root of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

# sincosf

Compute the sine and cosine of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE void sincosf(
                floatInVec arg x,
                floatInVec *s,
                floatInVec *c
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
$s$	Pointer to a vector register (containing one float).
$c$	Pointer to a vector register (containing one float).

## Return Values

None

## Description

Compute the sine and cosine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- $s$  will be a NaN value if  $x$  is a NaN value.
- $s$  will be a NaN value if  $x$  is a +/-INF value.
- $s$  will be a +/-0 value if  $x$  is a +/-0 value.
- $c$  will be a NaN value if  $x$  is a NaN value.
- $c$  will be a NaN value if  $x$  is a +/-INF value.

SCE CONFIDENTIAL

# **sinf**

Compute the sine of  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec sinf(
                floatInVec arg x
            );
        }
    }
}
```

## **Arguments**

$x$  Scalar value (contained in a vector register).

## **Return Values**

Sine of  $x$

## **Description**

Compute the sine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A NaN value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# **sinhf**

Compute the hyperbolic sine of  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec sinhf(
                floatInVec arg x
            );
        }
    }
}
```

## **Arguments**

$x$  Scalar value (contained in a vector register).

## **Return Values**

Hyperbolic sine of  $x$

## **Description**

Compute the hyperbolic sine of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A +/-INF value is returned if  $x$  is a +/-INF value.

A +/-INF value is returned if the result overflows and  $x$  has a +/- sign.

A +/-0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# sqrtf

Compute the square root of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_ALWAYS_INLINE floatInVec sqrtf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$	Scalar value (contained in a vector register).
-----	--

## Return Values

Square root of $x$
--------------------

## Description

Compute the square root of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

- A NaN value is returned if  $x$  is a NaN value.
- A NaN value is returned if  $x$  is a negative value (other than -0).
- A +INF value is returned if  $x$  is a +INF value.
- A +/-0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# **tanf**

Compute the tangent of  $x$ .

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec tanf(
                floatInVec arg
            );
        }
    }
}
```

## **Arguments**

$x$  Scalar value (contained in a vector register).

## **Return Values**

Tangent of  $x$

## **Description**

Compute the tangent of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## **Notes**

A NaN value is returned if  $x$  is a NaN value.

A NaN value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

SCE CONFIDENTIAL

# tanhf

Compute the hyperbolic tangent of  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec tanhf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Hyperbolic tangent of  $x$

## Description

Compute the hyperbolic tangent of  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A  $+/-1$  value is returned if  $x$  is a  $+/-\text{INF}$  value.

A  $+/-0$  value is returned if  $x$  is a  $+/-0$  value.

SCE CONFIDENTIAL

# truncf

Compute the integral value nearest to but no larger in magnitude than  $x$ .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            SCE_VECTORMATH_INLINE floatInVec truncf(
                floatInVec arg x
            );
        }
    }
}
```

## Arguments

$x$  Scalar value (contained in a vector register).

## Return Values

Integral value nearest to but no larger in magnitude than  $x$

## Description

Compute the integral value nearest to but no larger in magnitude than  $x$ . Each operand is contained in its own vector register, and the result is contained in another vector register.

## Notes

A NaN value is returned if  $x$  is a NaN value.

A +/-INF value is returned if  $x$  is a +/-INF value.

A +/-0 value is returned if  $x$  is a +/-0 value.

**sce::Vectormath::Simd::Aos**

000004892117

# Summary

## sce::Vectormath::Simd::Aos

The namespace containing array-of-structures (AoS) classes.

### Definition

```
namespace Aos { }
```

### Description

The namespace containing array-of-structures (AoS) classes.

### Function Summary

Function	Description
<a href="#">absPerElem</a>	Compute the absolute value of a 2x2 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 2D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3D point per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3x3 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4x4 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3x4 transformation matrix per element.
<a href="#">affineInverse</a>	Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix.
<a href="#">allElemEqual</a>	All of the elements of the first 2D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 3D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 4D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 3D point are equal to the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 2D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 3D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 4D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

Function	Description
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 2D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 3D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 4D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 3D point are less than the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 3D point are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 2D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 3D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 4D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 3D point are not equal to the corresponding element in the second.
<a href="#"><u>angle</u></a>	Compute the angle of a 2D vector against the x-axis.
<a href="#"><u>angleBetween</u></a>	Compute the angle between two 2D vectors.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 2x2 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 3x3 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 4x4 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 2D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 3D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 4D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 3D point between corresponding elements specifying minimum and maximum values.
<a href="#"><u>conj</u></a>	Compute the conjugate of a quaternion.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 2D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 3D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 4D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 3D point to another, per element.
<a href="#"><u>cross</u></a>	Compute cross product of two 3D vectors.
<a href="#"><u>crossMatrix</u></a>	Cross-product matrix of a 3D vector.
<a href="#"><u>crossMatrixMul</u></a>	Create cross-product matrix and multiply.
<a href="#"><u>determinant</u></a>	Determinant of a 2x2 matrix.
<a href="#"><u>determinant</u></a>	Compute the determinant of the matrix created from two 2D vectors columns.
<a href="#"><u>determinant</u></a>	Determinant of a 3x3 matrix.
<a href="#"><u>determinant</u></a>	Determinant of a 4x4 matrix.
<a href="#"><u>dist</u></a>	Compute the distance between two 3D points.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>distFromOrigin</u></a>	Compute the distance of a 3D point from the coordinate-system origin.
<a href="#"><u>distSqr</u></a>	Compute the square of the distance between two 3D points.
<a href="#"><u>distSqrFromOrigin</u></a>	Compute the square of the distance of a 3D point from the coordinate-system origin.
<a href="#"><u>divPerElem</u></a>	Divide two 2D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 3D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 4D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 3D points per element.
<a href="#"><u>dot</u></a>	Compute the dot product of two 2D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two 3D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two 4D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two quaternions.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 2x2 matrix.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 3x3 matrix.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 4x4 matrix.
<a href="#"><u>inverse</u></a>	Inverse of a 3x4 transformation matrix.
<a href="#"><u>length</u></a>	Compute the length of a 2D vector.
<a href="#"><u>length</u></a>	Compute the length of a 3D vector.
<a href="#"><u>length</u></a>	Compute the length of a 4D vector.
<a href="#"><u>length</u></a>	Compute the length of a quaternion.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 2D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 3D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 4D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a quaternion.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 2D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 2D vectors (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D vectors (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4D vectors (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D points.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D points (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two quaternions.
<a href="#"><u>lerp</u></a>	Linear interpolation between two quaternions (scalar data contained in vector data type).
<a href="#"><u>loadHalfFloats</u></a>	Load eight two-half-floats as 2D vectors.
<a href="#"><u>loadHalfFloats</u></a>	Load two half-floats as a 2D vector.
<a href="#"><u>loadHalfFloats</u></a>	Load eight three-half-floats as 3D vectors.
<a href="#"><u>loadHalfFloats</u></a>	Load three half-floats as a 3D vector.
<a href="#"><u>loadHalfFloats</u></a>	Load four four-half-floats as 4D vectors.
<a href="#"><u>loadHalfFloats</u></a>	Load four half-floats as a 4D vector.
<a href="#"><u>loadHalfFloats</u></a>	Load eight three-half-floats as 3D points.
<a href="#"><u>loadHalfFloats</u></a>	Load three half-floats as a 3D point.
<a href="#"><u>loadXY</u></a>	Load $x$ and $y$ elements from the first two words of a doubleword array.
<a href="#"><u>loadXY</u></a>	Load $x$ and $y$ elements from the first two words of a float array.
<a href="#"><u>loadXYArray</u></a>	Load four two-float 2D vectors from the first two quadwords of a quadword array.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>loadXYZ</u></a>	Load $x$ , $y$ , and $z$ elements from the first three words of a quadword.
<a href="#"><u>loadXYZ</u></a>	Load $x$ , $y$ , and $z$ elements from the first three words of a float array.
<a href="#"><u>loadXYZ</u></a>	Load $x$ , $y$ , and $z$ elements from the first three words of a quadword.
<a href="#"><u>loadXYZ</u></a>	Load $x$ , $y$ , and $z$ elements from the first three words of a float array.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3D vectors, stored in three quadwords.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3D points, stored in three quadwords.
<a href="#"><u>loadXYZW</u></a>	Load $x$ , $y$ , $z$ , and $w$ elements from the first four words of a quadword array.
<a href="#"><u>loadXYZW</u></a>	Load $x$ , $y$ , $z$ , and $w$ elements from the first four words of a float array.
<a href="#"><u>loadXYZW</u></a>	Load $x$ , $y$ , $z$ , and $w$ elements from the first four words of a quadword.
<a href="#"><u>loadXYZW</u></a>	Load $x$ , $y$ , $z$ , and $w$ elements from the first four words of a float array.
<a href="#"><u>maxElem</u></a>	Maximum element of a 2D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 4D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3D point.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 2D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 4D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3D points per element.
<a href="#"><u>minElem</u></a>	Minimum element of a 2D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 4D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3D point.
<a href="#"><u>minPerElem</u></a>	Minimum of two 2D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 4D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3D points per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 2x2 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 2D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3D points per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3x3 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4x4 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3x4 transformation matrices per element.
<a href="#"><u>norm</u></a>	Compute the norm of a quaternion.
<a href="#"><u>normalize</u></a>	Normalize a 2D vector.
<a href="#"><u>normalize</u></a>	Normalize a 3D vector.
<a href="#"><u>normalize</u></a>	Normalize a 4D vector.
<a href="#"><u>normalize</u></a>	Normalize a quaternion.
<a href="#"><u>operator*</u></a>	Multiply a 2x2 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 2x2 matrix by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*</u></a>	Multiply a 2D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 2D vector by a scalar (scalar data contained in vector data type).

Function	Description
<a href="#"><u>operator*</u></a>	Multiply a 3D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 3D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*</u></a>	Multiply a 4D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 4D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*</u></a>	Multiply a quaternion by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a quaternion by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*</u></a>	Multiply a 3x3 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*</u></a>	Multiply a 4x4 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).
<a href="#"><u>orthoInverse</u></a>	Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix with an orthogonal upper-left 3x3 submatrix.
<a href="#"><u>orthoInverse</u></a>	Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.
<a href="#"><u>outer</u></a>	Outer product of two 3D vectors.
<a href="#"><u>outer</u></a>	Outer product of two 4D vectors.
<a href="#"><u>perp</u></a>	Compute a 2D vector perpendicular to the 2D vector.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 2x2 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 3x3 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 4x4 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#"><u>print</u></a>	Print a 2x2 matrix.
<a href="#"><u>print</u></a>	Print a 2x2 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 2D vector.
<a href="#"><u>print</u></a>	Print a 2D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3D vector.
<a href="#"><u>print</u></a>	Print a 3D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 4D vector.
<a href="#"><u>print</u></a>	Print a 4D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3D point.
<a href="#"><u>print</u></a>	Print a 3D point and an associated string identifier.
<a href="#"><u>print</u></a>	Print a quaternion.
<a href="#"><u>print</u></a>	Print a quaternion and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3x3 matrix.
<a href="#"><u>print</u></a>	Print a 3x3 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 4x4 matrix.
<a href="#"><u>print</u></a>	Print a 4x4 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3x4 transformation matrix.
<a href="#"><u>print</u></a>	Print a 3x4 transformation matrix and an associated string identifier.
<a href="#"><u>projection</u></a>	Scalar projection of a 3D point on a unit-length 3D vector.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 2D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 4D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3D point per element.
<a href="#"><u>rotate</u></a>	Rotate a 2D vector.
<a href="#"><u>rotate</u></a>	Use a unit-length quaternion to rotate a 3D vector.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>rowMul</u></a>	Pre-multiply a row vector by a 3x3 matrix.
<a href="#"><u>rsgrtPerElem</u></a>	Compute the reciprocal square root of a 2D vector per element.
<a href="#"><u>rsgrtPerElem</u></a>	Compute the reciprocal square root of a 3D vector per element.
<a href="#"><u>rsgrtPerElem</u></a>	Compute the reciprocal square root of a 4D vector per element.
<a href="#"><u>rsgrtPerElem</u></a>	Compute the reciprocal square root of a 3D point per element.
<a href="#"><u>scale</u></a>	Apply uniform scale to a 3D point.
<a href="#"><u>scale</u></a>	Apply uniform scale to a 3D point (scalar data contained in vector data type).
<a href="#"><u>scale</u></a>	Apply non-uniform scale to a 3D point.
<a href="#"><u>select</u></a>	Conditionally select between two 2x2 matrices.
<a href="#"><u>select</u></a>	Conditionally select between two 2x2 matrices (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 2D vectors.
<a href="#"><u>select</u></a>	Conditionally select between two 2D vectors (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 3D vectors.
<a href="#"><u>select</u></a>	Conditionally select between two 3D vectors (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 4D vectors.
<a href="#"><u>select</u></a>	Conditionally select between two 4D vectors (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 3D points.
<a href="#"><u>select</u></a>	Conditionally select between two 3D points (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two quaternions.
<a href="#"><u>select</u></a>	Conditionally select between two quaternions (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 3x3 matrices.
<a href="#"><u>select</u></a>	Conditionally select between two 3x3 matrices (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 4x4 matrices.
<a href="#"><u>select</u></a>	Conditionally select between two 4x4 matrices (scalar data contained in vector data type).
<a href="#"><u>select</u></a>	Conditionally select between two 3x4 transformation matrices.
<a href="#"><u>select</u></a>	Conditionally select between two 3x4 transformation matrices (scalar data contained in vector data type).
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 2D vectors.
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 2D vectors (scalar data contained in vector data type).
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 3D vectors.
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 3D vectors (scalar data contained in vector data type).
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 4D vectors.
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two 4D vectors (scalar data contained in vector data type).
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two quaternions.
<a href="#"><u>slerp</u></a>	Spherical linear interpolation between two quaternions (scalar data contained in vector data type).
<a href="#"><u>sqrtPerElem</u></a>	Compute the square root of a 2D vector per element.
<a href="#"><u>sqrtPerElem</u></a>	Compute the square root of a 3D vector per element.
<a href="#"><u>sqrtPerElem</u></a>	Compute the square root of a 4D vector per element.
<a href="#"><u>sqrtPerElem</u></a>	Compute the square root of a 3D point per element.
<a href="#"><u>squad</u></a>	Spherical quadrangle interpolation.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>squad</u></a>	Spherical quadrangle interpolation (scalar data contained in vector data type).
<a href="#"><u>storeHalfFloats</u></a>	Store eight 2D vectors as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store a 2D vector as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store eight 3D vectors as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store a 3D vector as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store four 4D vectors as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store a 4D vector as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store eight 3D points as half-floats.
<a href="#"><u>storeHalfFloats</u></a>	Store a 3D point as half-floats.
<a href="#"><u>storeXY</u></a>	Store $x$ and $y$ elements of a 2D vector in the first two words of a doubleword array.
<a href="#"><u>storeXY</u></a>	Store $x$ and $y$ elements of a 2D vector in the first two words of a float array.
<a href="#"><u>storeXYZArray</u></a>	Store four two-float 2D vectors in the first two quadwords of a quadword array.
<a href="#"><u>storeXYZ</u></a>	Store $x$ , $y$ , and $z$ elements of a 3D vector in the first three words of a quadword.
<a href="#"><u>storeXYZ</u></a>	Store $x$ , $y$ , and $z$ elements of a 3D vector in the first three words of a float array.
<a href="#"><u>storeXYZ</u></a>	Store $x$ , $y$ , and $z$ elements of a 3D point in the first three words of a quadword.
<a href="#"><u>storeXYZ</u></a>	Store $x$ , $y$ , and $z$ elements of a 3D point in the first three words of a float array.
<a href="#"><u>storeXYZArray</u></a>	Store four 3D vectors in three quadwords.
<a href="#"><u>storeXYZArray</u></a>	Store four 3D points in three quadwords.
<a href="#"><u>storeXYZW</u></a>	Store $x$ , $y$ , $z$ , and $w$ elements of a 4D vector in the first four words of a quadword array.
<a href="#"><u>storeXYZW</u></a>	Store $x$ , $y$ , $z$ , and $w$ elements of a 4D vector in the first four words of a float array.
<a href="#"><u>storeXYZW</u></a>	Store $x$ , $y$ , $z$ , and $w$ elements of a quaternion in the first four words of a quadword array.
<a href="#"><u>storeXYZW</u></a>	Store $x$ , $y$ , $z$ , and $w$ elements of a quaternion in the first four words of a float array.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 2D vector.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 3D vector.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 4D vector.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 3D point.
<a href="#"><u>transpose</u></a>	Transpose of a 2x2 matrix.
<a href="#"><u>transpose</u></a>	Transpose of a 3x3 matrix.
<a href="#"><u>transpose</u></a>	Transpose of a 4x4 matrix.

## Inner Classes, Structures, and Namespaces

Item	Description
<a href="#"><u>sce::Vectormath::Simd::Aos::Matrix2</u></a>	A 2x2 matrix in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::Matrix3</u></a>	A 3x3 matrix in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::Matrix4</u></a>	A 4x4 matrix in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::Point3</u></a>	A 3D point in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::Quat</u></a>	A quaternion in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::Transform3</u></a>	A 3x4 transformation matrix in array-of-structures format.
<a href="#"><u>sce::Vectormath::Simd::Aos::VecIdx</u></a>	A reference to an element in a SIMD vec_float4 vector.

SCE CONFIDENTIAL

Item	Description
<a href="#">sce::Vectormath::Simd::Aos::Vector2</a>	A 2D vector in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Vector3</a>	A 3D vector in array-of-structures format.
<a href="#">sce::Vectormath::Simd::Aos::Vector4</a>	A 4D vector in array-of-structures format.

000004892117

# Type Definitions

## Matrix2\_arg

Type used when passing [Matrix2](#) as a function argument.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Matrix2 SCE_VECTORMATH_SIMD_AOS_MATRIX_ARG
                Matrix2_arg;
            }
        }
    }
}
```

### Description

Type used when passing [Matrix2](#) as a function argument.

# **Matrix3\_arg**

Type used when passing [Matrix3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Matrix3 SCE_VECTORMATH SIMD_AOS_MATRIX_ARG
                Matrix3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Matrix3](#) as a function argument.

# **Matrix4\_arg**

Type used when passing [Matrix4](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Matrix4 SCE_VECTORMATH SIMD_AOS_MATRIX_ARG
                Matrix4_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Matrix4](#) as a function argument.

SCE CONFIDENTIAL

## **Point3\_arg**

Type used when passing [Point3](#) as a function argument.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Point3 SCE_VECTORMATH SIMD_AOS_VECTOR_ARG
                Point3_arg;
            }
        }
    }
}
```

### **Description**

Type used when passing [Point3](#) as a function argument.

SCE CONFIDENTIAL

## **Quat\_arg**

---

Type used when passing [Quat](#) as a function argument.

### **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Quat SCE_VECTORMATH_SIMD_AOS_VECTOR_ARG Quat_arg;
            }
        }
    }
}
```

### **Description**

---

Type used when passing [Quat](#) as a function argument.

# **Transform3\_arg**

Type used when passing [Transform3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Transform3 SCE_VECTORMATH SIMD_AOS_MATRIX_ARG
                Transform3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Transform3](#) as a function argument.

# **Vector2\_arg**

Type used when passing [Vector2](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Vector2 SCE_VECTORMATH SIMD_AOS_VECTOR_ARG
                Vector2_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector2](#) as a function argument.

# **Vector3\_arg**

Type used when passing [Vector3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Vector3 SCE_VECTORMATH SIMD_AOS_VECTOR_ARG
                Vector3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector3](#) as a function argument.

# **Vector4\_arg**

Type used when passing [Vector4](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                typedef const Vector4 SCE_VECTORMATH SIMD_AOS_VECTOR_ARG
                Vector4_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector4](#) as a function argument.

# Functions

## absPerElem

Compute the absolute value of a 2x2 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 absPerElem(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat*                  2x2 matrix.

### Return Values

2x2 matrix in which each element is the absolute value of the corresponding element of the specified 2x2 matrix

### Description

Compute the absolute value of each element of a 2x2 matrix.

# appendScale

Append (post-multiply) a scale transformation to a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 appendScale(
                    Matrix2 arg mat,
                    Vector2 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	2x2 matrix.
<i>scaleVec</i>	2D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 2x2 matrix by a scale transformation whose diagonal scale factors are contained in the 2D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

# determinant

Determinant of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec determinant(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 2x2 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 inverse(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      2x2 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 2x2 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# mulPerElem

Multiply two 2x2 matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 mulPerElem(
                    Matrix2 arg mat0,
                    Matrix2 arg mat1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	2x2 matrix.
<i>mat1</i>	2x2 matrix.

## Return Values

2x2 matrix in which each element is the product of the corresponding elements of the specified 2x2 matrices

## Description

Multiply two 2x2 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 2x2 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 operator*(
                    float scalar,
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	2x2 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 2x2 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 2x2 matrix by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 operator*(
                    floatInVec arg scalar,
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>mat</i>	2x2 matrix.

## **Return Values**

Scalar product of *mat* and *scalar*

## **Description**

Multiply a 2x2 matrix by a scalar.

# prependScale

Prepend (pre-multiply) a scale transformation to a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 prependScale(
                    Vector2 arg scaleVec,
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	2D vector.
<i>mat</i>	2x2 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 2x2 matrix by a scale transformation whose diagonal scale factors are contained in the 2D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      2x2 matrix.

## Return Values

None

## Description

Print a 2x2 matrix. Unlike the printing of vectors, the 2x2 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# print

Print a 2x2 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix2 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	2x2 matrix.
<i>name</i>	String printed with the 2x2 matrix.

## Return Values

None

## Description

Print a 2x2 matrix and an associated string identifier. Unlike the printing of vectors, the 2x2 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 select(
                    Matrix2 arg mat0,
                    Matrix2 arg mat1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	2x2 matrix.
<i>mat1</i>	2x2 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 2x2 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 2x2 matrices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 select(
                    Matrix2 arg mat0,
                    Matrix2 arg mat1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	2x2 matrix.
<i>mat1</i>	2x2 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 2x2 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix2 transpose(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 2x2 matrix.

# 2D Vector Functions

## absPerElem

Compute the absolute value of a 2D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 absPerElem(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec*                  2D vector.

### Return Values

2D vector in which each element is the absolute value of the corresponding element of *vec*

### Description

Compute the absolute value of each element of a 2D vector.

# allElemEqual

All of the elements of the first 2D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemEqual(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 2D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false if any element has a NaN value.

# allElemGreaterThan

All of the elements of the first 2D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThan(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 2D vector are greater than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemGreaterThanOrEqual**

All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThanOrEqual (
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## **Return Values**

True if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThan

All of the elements of the first 2D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThan(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 2D vector are less than the corresponding element in the second.

# allElemLessThanOrEqual

All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThanOrEqual(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True if all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.

# **allElemNotEqual**

All of the elements of the first 2D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemNotEqual(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## **Return Values**

True if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 2D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# angle

Compute the angle of a 2D vector against the x-axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec angle(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

`vec`      2D vector.

## Return Values

Angle of the specified 2D vector against the x-axis

## Description

Compute the angle of a 2D vector against the x-axis.

# angleBetween

Compute the angle between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec angleBetween(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Angle between two specified 2D vectors

## Description

Compute the angle between two 2D vectors.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 2D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 clampPerElem(
                    Vector2 arg vec,
                    Vector2 arg clampMin,
                    Vector2 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector to be clamped.
<i>clampMin</i>	2D vector containing the minimum values of the range.
<i>clampMax</i>	2D vector containing the maximum values of the range.

## Return Values

2D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 2D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 copySignPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

# determinant

Compute the determinant of the matrix created from two 2D vectors columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec determinant(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Determinant between two 2D vectors

## Description

Compute the determinant of the matrix created from two 2D vectors columns.

SCE CONFIDENTIAL

# divPerElem

Divide two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 divPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the quotient of the corresponding elements of the specified 2D vectors

## Description

Divide two 2D vectors element by element.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec dot(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Dot product of the specified 2D vectors

## Description

Compute the dot product of two 2D vectors.

SCE CONFIDENTIAL

# length

Compute the length of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec length(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

Length of the specified 2D vector

## Description

Compute the length of a 2D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec lengthSqr(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

Square of the length of the specified 2D vector

## **Description**

Compute the square of the length of a 2D vector.

# lerp

Linear interpolation between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 lerp(
                    float t,
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Interpolated 2D vector

## Description

Linearly interpolate between two 2D vectors.

## Notes

Does not clamp *t* between 0 and 1.

# lerp

Linear interpolation between two 2D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 lerp(
                    floatInVec arg t,
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Interpolated 2D vector

## Description

Linearly interpolate between two 2D vectors.

## Notes

Does not clamp *t* between 0 and 1.

# loadHalfFloats

Load eight two-half-floats as 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector2 &vec0,
                    Vector2 &vec1,
                    Vector2 &vec2,
                    Vector2 &vec3,
                    Vector2 &vec4,
                    Vector2 &vec5,
                    Vector2 &vec6,
                    Vector2 &vec7,
                    const vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	An output 2D vector.
<i>vec1</i>	An output 2D vector.
<i>vec2</i>	An output 2D vector.
<i>vec3</i>	An output 2D vector.
<i>vec4</i>	An output 2D vector.
<i>vec5</i>	An output 2D vector.
<i>vec6</i>	An output 2D vector.
<i>vec7</i>	An output 2D vector.
<i>twoQuads</i>	Array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Load eight two-half-floats as 2D vectors. The input is  $\{x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7\}$ .

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadHalfFloats

Load two half-floats as a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector2 &vec,
                    const uint16_t *hfptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 2D vector.
<i>hfptr</i>	Array of two half-floats.

## Return Values

None

## Description

Load two half-floats as a 2D vector.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadXY

Load *x* and *y* elements from the first two words of a doubleword array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXY(
                    Vector2 &vec,
                    const vec_float2 *dual
                );
            }
        }
    }
}
```

## Arguments

*vec*  
*dual*

An output 2D vector.  
Pointer to a doubleword array from which *x* and *y* will be loaded.

## Return Values

None

## Description

Load *x* and *y* elements from the first two words of a doubleword array.

SCE CONFIDENTIAL

# loadXY

Load *x* and *y* elements from the first two words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXY(
                    Vector2 &vec,
                    const float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 2D vector.
<i>fptr</i>	Array of float from which <i>x</i> and <i>y</i> will be loaded.

## Return Values

None

## Description

Load *x* and *y* elements from the first two words of a float array.

# **loadXYArray**

Load four two-float 2D vectors from the first two quadwords of a quadword array.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYArray(
                    Vector2 &vec0,
                    Vector2 &vec1,
                    Vector2 &vec2,
                    Vector2 &vec3,
                    const vec_float4 *twoQuads
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	First 2D vector in the quadword array.
<i>vec1</i>	Second 2D vector in the quadword array.
<i>vec2</i>	Third 2D vector in the quadword array.
<i>vec3</i>	Fourth 2D vector in the quadword array.
<i>twoQuads</i>	Pointer to a quadword array from which the four two-float 2D vectors will be loaded.

## **Return Values**

None

## **Description**

Load four two-float 2D vectors from the first two quadwords of a quadword array.

# maxElem

Maximum element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec maxElem(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 2D vector.

SCE CONFIDENTIAL

# maxPerElem

Maximum of two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 maxPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the maximum of the corresponding elements of the specified 2D vectors

## Description

Create a 2D vector in which each element is the maximum of the corresponding elements of the specified 2D vectors.

# minElem

Minimum element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec minElem(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 2D vector.

# minPerElem

Minimum of two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 minPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the minimum of the corresponding elements of the specified 2D vectors

## Description

Create a 2D vector in which each element is the minimum of the corresponding elements of two specified 2D vectors.

# mulPerElem

Multiply two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 mulPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the product of the corresponding elements of the specified 2D vectors

## Description

Multiply two 2D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 normalize(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      2D vector.

## Return Values

The specified 2D vector scaled to unit length

## Description

Compute a normalized 2D vector.

## Notes

The result is unpredictable when all elements of *vec* are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a 2D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 operator*(
                    float scalar,
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>vec</i>	2D vector.

## Return Values

Scalar product of *vec* and *scalar*

## Description

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 2D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 operator*(
                    floatInVec arg scalar,
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>vec</i>	2D vector.

## **Return Values**

Scalar product of *vec* and *scalar*

## **Description**

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# perp

Compute a 2D vector perpendicular to the 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 perp(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A 2D vector perpendicular to the specified 2D vector

## Description

Compute a 2D vector perpendicular to the 2D vector.

SCE CONFIDENTIAL

# print

Print a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      2D vector.

## Return Values

None

## Description

Print a 2D vector.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 2D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector2 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector.
<i>name</i>	String printed with the 2D vector.

## Return Values

None

## Description

Print a 2D vector and an associated string identifier.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 2D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 recipPerElem(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

2D vector in which each element is the reciprocal of the corresponding element of the specified 2D vector

## **Description**

Create a 2D vector in which each element is the reciprocal of the corresponding element of the specified 2D vector.

SCE CONFIDENTIAL

# rotate

Rotate a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 rotate(
                    floatInVec arg rot,
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>rot</i>	Angle (radians).
<i>vec</i>	2D vector.

## Return Values

The rotated 2D vector

## Description

Rotate a 2D vector by a specified angle (radians).

# rsqrtPerElem

Compute the reciprocal square root of a 2D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 rsqrtPerElem(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

2D vector in which each element is the reciprocal square root of the corresponding element of the specified 2D vector

## Description

Create a 2D vector in which each element is the reciprocal square root of the corresponding element of the specified 2D vector.

SCE CONFIDENTIAL

# select

Conditionally select between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 select(
                    Vector2_arg vec0,
                    Vector2_arg vec1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 2D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 2D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 select(
                    Vector2 arg vec0,
                    Vector2 arg vec1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 2D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 2D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 slerp(
                    float t,
                    Vector2 arg unitVec0,
                    Vector2 arg unitVec1,
                    float tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_F
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	2D vector, expected to be unit-length.
<i>unitVec1</i>	2D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 2D vector

## **Description**

Perform spherical linear interpolation between two 2D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 2D vectors (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 slerp(
                    floatInVec arg t,
                    Vector2 arg unitVec0,
                    Vector2 arg unitVec1,
                    floatInVec arg tol =
                        floatInVec(SCE_VECTORMATH_DEFAULT_SLERP_TOL_V)
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	2D vector, expected to be unit-length.
<i>unitVec1</i>	2D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 2D vector

## **Description**

Perform spherical linear interpolation between two 2D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

# **sqrtPerElem**

Compute the square root of a 2D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector2 sqrtPerElem(
                    Vector2 arg
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

2D vector in which each element is the square root of the corresponding element of the specified 2D vector

## **Description**

Create a 2D vector in which each element is the square root of the corresponding element of the specified 2D vector.

# storeHalfFloats

Store eight 2D vectors as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector2_arg vec0,
                    Vector2_arg vec1,
                    Vector2_arg vec2,
                    Vector2_arg vec3,
                    Vector2_arg vec4,
                    Vector2_arg vec5,
                    Vector2_arg vec6,
                    Vector2_arg vec7,
                    vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.
<i>vec2</i>	2D vector.
<i>vec3</i>	2D vector.
<i>vec4</i>	2D vector.
<i>vec5</i>	2D vector.
<i>vec6</i>	2D vector.
<i>vec7</i>	2D vector.
<i>twoQuads</i>	An output array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Store eight 2D vectors in two quadwords of half-float values. The output is { $x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7$ }.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeHalfFloats

Store a 2D vector as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector2<arg> vec,
                    uint16_t *hptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector.
<i>hptr</i>	An output array of two half-floats.

## Return Values

None

## Description

Store a 2D vector in a `uint16_t` array of half-float values.

## Notes

This transformation does not support either denormalized numbers or NaNs. The memory area of the previous 16 bytes and the next 32 bytes from *hptr* might be accessed.

SCE CONFIDENTIAL

# storeXY

Store  $x$  and  $y$  elements of a 2D vector in the first two words of a doubleword array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXY(
                    Vector2 arg vec,
                    vec_float2 *dual
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector.
<i>dual</i>	Pointer to a doubleword array in which $x$ and $y$ will be stored.

## Return Values

None

## Description

Store  $x$  and  $y$  elements of a 2D vector in the first two words of a doubleword array.

SCE CONFIDENTIAL

# storeXY

Store  $x$  and  $y$  elements of a 2D vector in the first two words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXY(
                    Vector2 arg vec,
                    float *fptr
                );
            }
        }
    }
}
```

## Arguments

vec	2D vector.
fptr	An output array of float in which $x$ and $y$ will be stored .

## Return Values

None

## Description

Store  $x$  and  $y$  elements of a 2D vector in the first two words of a float array.

## Notes

Memory area of previous 16 bytes and next 32 bytes from  $fptr$  might be accessed.

# **storeXYArray**

Store four two-float 2D vectors in the first two quadwords of a quadword array.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYArray(
                    Vector2_arg vec0,
                    Vector2_arg vec1,
                    Vector2_arg vec2,
                    Vector2_arg vec3,
                    vec_float4 *twoQuads
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	First 2D vector in the quadword array.
<i>vec1</i>	Second 2D vector in the quadword array.
<i>vec2</i>	Third 2D vector in the quadword array.
<i>vec3</i>	Fourth 2D vector in the quadword array.
<i>twoQuads</i>	Pointer to a quadword array in which the four two-float 2D vectors will be stored.

## **Return Values**

None

## **Description**

Store four two-float 2D vectors in the first two quadwords of a quadword array.

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 2D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec sum(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec*                  2D vector.

### Return Values

Sum of all elements of *vec*

### Description

Compute the sum of all elements of a 2D vector.

# 3D Vector Functions

## absPerElem

Compute the absolute value of a 3D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 absPerElem(
                    Vector3_arg vec
                );
            }
        }
    }
}
```

### Arguments

vec                   3D vector.

### Return Values

3D vector in which each element is the absolute value of the corresponding element of vec

### Description

Compute the absolute value of each element of a 3D vector.

# allElemEqual

All of the elements of the first 3D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemEqual(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 3D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false if any element has a NaN value.

# allElemGreaterThan

All of the elements of the first 3D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThan(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 3D vector are greater than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemGreaterThanOrEqual**

All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThanOrEqual (
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## **Return Values**

True if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThan

All of the elements of the first 3D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThan(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 3D vector are less than the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThanOrEqual

All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThanOrEqual(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True if all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.

# **allElemNotEqual**

All of the elements of the first 3D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemNotEqual(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

*vec0* 3D vector.  
*vec1* 3D vector.

## **Return Values**

True if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 3D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 3D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 clampPerElem(
                    Vector3 arg vec,
                    Vector3 arg clampMin,
                    Vector3 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector to be clamped.
<i>clampMin</i>	3D vector containing the minimum values of the range.
<i>clampMax</i>	3D vector containing the maximum values of the range.

## Return Values

3D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 3D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 copySignPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

SCE CONFIDENTIAL

## cross

Compute cross product of two 3D vectors.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 cross(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

### Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

### Return Values

Cross product of the specified 3D vectors

### Description

Compute cross product of two 3D vectors.

SCE CONFIDENTIAL

# crossMatrix

Cross-product matrix of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 crossMatrix(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Cross-product matrix of *vec*

## Description

Compute a matrix that, when multiplied by a 3D vector, produces the same result as a cross product with that 3D vector.

SCE CONFIDENTIAL

# **crossMatrixMul**

Create cross-product matrix and multiply.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 crossMatrixMul(
                    Vector3 arg vec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## **Return Values**

Product of cross-product matrix of *vec* and *mat*

## **Description**

Multiply a cross-product matrix by another matrix.

## **Notes**

Faster than separately creating a cross-product matrix and multiplying.

SCE CONFIDENTIAL

# divPerElem

Divide two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 divPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the quotient of the corresponding elements of the specified 3D vectors

## Description

Divide two 3D vectors element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec dot(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

Dot product of the specified 3D vectors

## Description

Compute the dot product of two 3D vectors.

SCE CONFIDENTIAL

# length

Compute the length of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec length(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Length of the specified 3D vector

## Description

Compute the length of a 3D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec lengthSqr(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                    3D vector.

## **Return Values**

Square of the length of the specified 3D vector

## **Description**

Compute the square of the length of a 3D vector.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 lerp(
                    float t,
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

Interpolated 3D vector

## Description

Linearly interpolate between two 3D vectors.

## Notes

Does not clamp *t* between 0 and 1.

# lerp

Linear interpolation between two 3D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 lerp(
                    floatInVec arg t,
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

Interpolated 3D vector

## Description

Linearly interpolate between two 3D vectors.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadHalfFloats

Load eight three-half-floats as 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector3 &vec0,
                    Vector3 &vec1,
                    Vector3 &vec2,
                    Vector3 &vec3,
                    Vector3 &vec4,
                    Vector3 &vec5,
                    Vector3 &vec6,
                    Vector3 &vec7,
                    const vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	An output 3D vector.
<i>vec1</i>	An output 3D vector.
<i>vec2</i>	An output 3D vector.
<i>vec3</i>	An output 3D vector.
<i>vec4</i>	An output 3D vector.
<i>vec5</i>	An output 3D vector.
<i>vec6</i>	An output 3D vector.
<i>vec7</i>	An output 3D vector.
<i>threeQuads</i>	Array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Load eight three-half-floats as 3D vectors. The input is  
 $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7\}$ .

## Notes

This transformation may not support either denormalized numbers or NaNs.

# loadHalfFloats

Load three half-floats as a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector3 &vec,
                    const uint16_t *hfptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 3D vector.
<i>hfptr</i>	Array of three half-floats.

## Return Values

None

## Description

Load three half-floats as a 3D vector.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadXYZ

Load *x*, *y*, and *z* elements from the first three words of a quadword.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZ(
                    Vector3 &vec,
                    const vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 3D vector.
<i>quad</i>	Pointer to a quadword from which <i>x</i> , <i>y</i> , and <i>z</i> will be loaded.

## Return Values

None

## Description

Load *x*, *y*, and *z* elements from the first three words of a quadword.

SCE CONFIDENTIAL

# loadXYZ

Load *x*, *y*, and *z* elements from the first three words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZ(
                    Vector3 &vec,
                    const float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 3D vector.
<i>fptr</i>	Array of float from which <i>x</i> , <i>y</i> , and <i>z</i> will be loaded.

## Return Values

None

## Description

Load *x*, *y*, and *z* elements from the first three words of a float array.

# **loadXYZArray**

Load four three-float 3D vectors, stored in three quadwords.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZArray(
                    Vector3 &vec0,
                    Vector3 &vec1,
                    Vector3 &vec2,
                    Vector3 &vec3,
                    const vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	An output 3D vector.
<i>vec1</i>	An output 3D vector.
<i>vec2</i>	An output 3D vector.
<i>vec3</i>	An output 3D vector.
<i>threeQuads</i>	Array of three quadwords containing 12 floats.

## **Return Values**

None

## **Description**

Load four three-float 3D vectors, stored in three quadwords as {x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3}, into four 3D vectors.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec maxElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 3D vector.

SCE CONFIDENTIAL

# maxPerElem

Maximum of two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 maxPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the maximum of the corresponding elements of the specified 3D vectors

## Description

Create a 3D vector in which each element is the maximum of the corresponding elements of the specified 3D vectors.

# minElem

Minimum element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec minElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 3D vector.

# minPerElem

Minimum of two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 minPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the minimum of the corresponding elements of the specified 3D vectors

## Description

Create a 3D vector in which each element is the minimum of the corresponding elements of two specified 3D vectors.

SCE CONFIDENTIAL

# mulPerElem

Multiply two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 mulPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the product of the corresponding elements of the specified 3D vectors

## Description

Multiply two 3D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 normalize(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

The specified 3D vector scaled to unit length

## Description

Compute a normalized 3D vector.

## Notes

The result is unpredictable when all elements of vec are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a 3D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 operator*(
                    float scalar,
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>vec</i>	3D vector.

## Return Values

Scalar product of *vec* and *scalar*

## Description

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 operator*(
                    floatInVec arg scalar,
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>vec</i>	3D vector.

## **Return Values**

Scalar product of *vec* and *scalar*

## **Description**

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# outer

Outer product of two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 outer(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

The 3x3 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

Compute the outer product of two 3D vectors.

## Notes

The term "outer product", which is used in documentation for the PlayStation®2 computer entertainment console, corresponds to "cross product" in this library.

SCE CONFIDENTIAL

# print

Print a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

None

## Description

Print a 3D vector. Prints the 3D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 3D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector3 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>name</i>	String printed with the 3D vector.

## Return Values

None

## Description

Print a 3D vector and an associated string identifier. Prints the 3D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 3D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 recipPerElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

`vec`      3D vector.

## **Return Values**

3D vector in which each element is the reciprocal of the corresponding element of the specified 3D vector

## **Description**

Create a 3D vector in which each element is the reciprocal of the corresponding element of the specified 3D vector.

## **Notes**

Floating-point behavior matches standard library function `recipf4`.

SCE CONFIDENTIAL

# rowMul

Pre-multiply a row vector by a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 rowMul(
                    Vector3 arg vec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## Return Values

Product of a row-vector and a 3x3 matrix

## Description

Transpose a 3D vector into a row vector and pre-multiply by 3x3 matrix.

## Notes

Slower than column post-multiply.

SCE CONFIDENTIAL

# **rsqrtPerElem**

Compute the reciprocal square root of a 3D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 rsqrtPerElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                   3D vector.

## **Return Values**

3D vector in which each element is the reciprocal square root of the corresponding element of the specified 3D vector

## **Description**

Create a 3D vector in which each element is the reciprocal square root of the corresponding element of the specified 3D vector.

## **Notes**

Floating-point behavior matches standard library function `rsqrtf4`.

# select

Conditionally select between two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 select(
                    Vector3_arg vec0,
                    Vector3_arg vec1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 3D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 3D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 select(
                    Vector3 arg vec0,
                    Vector3 arg vec1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 3D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 3D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 slerp(
                    float t,
                    Vector3 arg unitVec0,
                    Vector3 arg unitVec1,
                    float tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_F
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	3D vector, expected to be unit-length.
<i>unitVec1</i>	3D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 3D vector

## **Description**

Perform spherical linear interpolation between two 3D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 3D vectors (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 slerp(
                    floatInVec arg t,
                    Vector3 arg unitVec0,
                    Vector3 arg unitVec1,
                    floatInVec arg tol =
                        floatInVec(SCE_VECTORMATH_DEFAULT_SLERP_TOL_V)
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	3D vector, expected to be unit-length.
<i>unitVec1</i>	3D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 3D vector

## **Description**

Perform spherical linear interpolation between two 3D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

# sqrtPerElem

Compute the square root of a 3D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 sqrtPerElem(
                    Vector3 arg
                );
            }
        }
    }
}
```

## Arguments

`vec`      3D vector.

## Return Values

3D vector in which each element is the square root of the corresponding element of the specified 3D vector

## Description

Create a 3D vector in which each element is the square root of the corresponding element of the specified 3D vector.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

# storeHalfFloats

Store eight 3D vectors as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector3 arg vec0,
                    Vector3 arg vec1,
                    Vector3 arg vec2,
                    Vector3 arg vec3,
                    Vector3 arg vec4,
                    Vector3 arg vec5,
                    Vector3 arg vec6,
                    Vector3 arg vec7,
                    vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>vec2</i>	3D vector.
<i>vec3</i>	3D vector.
<i>vec4</i>	3D vector.
<i>vec5</i>	3D vector.
<i>vec6</i>	3D vector.
<i>vec7</i>	3D vector.
<i>threeQuads</i>	An output array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Store eight 3D vectors in three quadwords of half-float values. The output is  $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7\}$ .

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeHalfFloats

Store a 3D vector as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector3 arg vec,
                    uint16_t *hptr
                );
            }
        }
    }
}
```

## Arguments

vec	3D vector.
hptr	An output array of three half-floats.

## Return Values

None

## Description

Store a 3D vector in a `uint16_t` array of half-float values.

## Notes

This transformation does not support either denormalized numbers or NaNs. The memory area of the previous 16 bytes and the next 32 bytes from `hptr` might be accessed.

# storeXYZ

Store *x*, *y*, and *z* elements of a 3D vector in the first three words of a quadword.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZ(
                    Vector3 arg vec,
                    vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>quad</i>	Pointer to a quadword in which <i>x</i> , <i>y</i> , and <i>z</i> will be stored.

## Return Values

None

## Description

Store *x*, *y*, and *z* elements of a 3D vector in the first three words of a quadword. The value of the fourth word (the word with the highest address) remains unchanged.

SCE CONFIDENTIAL

# storeXYZ

Store *x*, *y*, and *z* elements of a 3D vector in the first three words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZ(
                    Vector3 arg vec,
                    float *fptr
                );
            }
        }
    }
}
```

## Arguments

vec	3D vector.
fptr	An output array of float in which <i>x</i> , <i>y</i> , and <i>z</i> will be stored.

## Return Values

None

## Description

Store *x*, *y*, and *z* elements of a 3D vector in the first three words of a float array.

## Notes

Memory area of previous 16 bytes and next 32 bytes from *fptr* might be accessed.

SCE CONFIDENTIAL

# storeXYZArray

Store four 3D vectors in three quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZArray(
                    Vector3 arg vec0,
                    Vector3 arg vec1,
                    Vector3 arg vec2,
                    Vector3 arg vec3,
                    vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>vec2</i>	3D vector.
<i>vec3</i>	3D vector.
<i>threeQuads</i>	An output array of three quadwords containing 12 floats.

## Return Values

None

## Description

Store four 3D vectors in three quadwords as {x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3}.

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 3D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec sum(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec*                    3D vector.

### Return Values

Sum of all elements of *vec*

### Description

Compute the sum of all elements of a 3D vector.

# 4D Vector Functions

## absPerElem

Compute the absolute value of a 4D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 absPerElem(
                    Vector4 arg
                );
            }
        }
    }
}
```

### Arguments

vec                  4D vector.

### Return Values

4D vector in which each element is the absolute value of the corresponding element of vec

### Description

Compute the absolute value of each element of a 4D vector.

# allElemEqual

All of the elements of the first 4D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemEqual(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 4D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false if any element has a NaN value.

# allElemGreaterThan

All of the elements of the first 4D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThan(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 4D vector are greater than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemGreaterThanOrEqual**

All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThanOrEqual (
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## **Return Values**

True if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.

# allElemLessThan

All of the elements of the first 4D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThan(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 4D vector are less than the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThanOrEqual

All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThanOrEqual(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True if all all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false.

## Description

All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.

# **allElemNotEqual**

All of the elements of the first 4D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemNotEqual(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## **Return Values**

True if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false.

## **Description**

All of the elements of the first 4D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 4D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 clampPerElem(
                    Vector4 arg vec,
                    Vector4 arg clampMin,
                    Vector4 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector to be clamped.
<i>clampMin</i>	4D vector containing the minimum values of the range.
<i>clampMax</i>	4D vector containing the maximum values of the range.

## Return Values

4D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 4D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 copySignPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

SCE CONFIDENTIAL

# divPerElem

Divide two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 divPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the quotient of the corresponding elements of the specified 4D vectors

## Description

Divide two 4D vectors element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec dot(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

Dot product of the specified 4D vectors

## Description

Compute the dot product of two 4D vectors.

SCE CONFIDENTIAL

# length

Compute the length of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec length(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Length of the specified 4D vector

## Description

Compute the length of a 4D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec lengthSqr(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

Square of the length of the specified 4D vector

## **Description**

Compute the square of the length of a 4D vector.

# lerp

Linear interpolation between two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 lerp(
                    float t,
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

Interpolated 4D vector

## Description

Linearly interpolate between two 4D vectors.

## Notes

Does not clamp *t* between 0 and 1.

# lerp

Linear interpolation between two 4D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 lerp(
                    floatInVec arg t,
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

Interpolated 4D vector

## Description

Linearly interpolate between two 4D vectors.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadHalfFloats

Load four four-half-floats as 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector4 &vec0,
                    Vector4 &vec1,
                    Vector4 &vec2,
                    Vector4 &vec3,
                    const vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	An output 4D vector.
<i>vec1</i>	An output 4D vector.
<i>vec2</i>	An output 4D vector.
<i>vec3</i>	An output 4D vector.
<i>twoQuads</i>	Array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Load four four-half-floats as 4D vectors. The input is {x0, y0, z0, w0, x1, y1, z1, w1, x2, y2, z2, w2, x3, y3, z3, w3}.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadHalfFloats

Load four half-floats as a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Vector4 &vec,
                    const uint16_t *hfptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 4D vector.
<i>hfptr</i>	Array of four half-floats.

## Return Values

None

## Description

Load four half-floats as a 4D vector.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadXYZW

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a quadword array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZW(
                    Vector4 &vec,
                    const vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 4D vector.
<i>quad</i>	Pointer to a quadword array from which $x$ , $y$ , $z$ , and $w$ will be loaded.

## Return Values

None

## Description

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a quadword array.

SCE CONFIDENTIAL

# loadXYZW

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZW(
                    Vector4 &vec,
                    const float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 4D vector.
<i>fptr</i>	Array of float from which $x$ , $y$ , $z$ , and $w$ will be loaded.

## Return Values

None

## Description

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a float array.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec maxElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  4D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 4D vector.

SCE CONFIDENTIAL

# maxPerElem

Maximum of two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 maxPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the maximum of the corresponding elements of the specified 4D vectors

## Description

Create a 4D vector in which each element is the maximum of the corresponding elements of the specified 4D vectors.

# minElem

Minimum element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec minElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  4D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 4D vector.

# minPerElem

Minimum of two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 minPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the minimum of the corresponding elements of the specified 4D vectors

## Description

Create a 4D vector in which each element is the minimum of the corresponding elements of two specified 4D vectors.

# mulPerElem

Multiply two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 mulPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the product of the corresponding elements of the specified 4D vectors

## Description

Multiply two 4D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 normalize(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      4D vector.

## Return Values

The specified 4D vector scaled to unit length

## Description

Compute a normalized 4D vector.

## Notes

The result is unpredictable when all elements of *vec* are at or near zero.

SCE CONFIDENTIAL

# **operator\***

---

Multiply a 4D vector by a scalar.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 operator*(
                    float scalar,
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	Scalar value.
<i>vec</i>	4D vector.

## **Return Values**

---

Scalar product of *vec* and *scalar*

## **Description**

---

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 4D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 operator*(
                    floatInVec arg scalar,
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>vec</i>	4D vector.

## **Return Values**

Scalar product of *vec* and *scalar*

## **Description**

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# outer

Outer product of two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 outer(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

The 4x4 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

Compute the outer product of two 4D vectors.

## Notes

The term "outer product", which is used in documentation for the PlayStation®2 computer entertainment console, corresponds to "cross product" in this library.

# print

Print a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

None

## Description

Print a 4D vector. Prints the 4D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when SCE\_VECTORMATH\_DEBUG is defined.

SCE CONFIDENTIAL

# print

Print a 4D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Vector4 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>name</i>	String printed with the 4D vector.

## Return Values

None

## Description

Print a 4D vector and an associated string identifier. Prints the 4D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 4D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 recipPerElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

4D vector in which each element is the reciprocal of the corresponding element of the specified 4D vector

## **Description**

Create a 4D vector in which each element is the reciprocal of the corresponding element of the specified 4D vector.

## **Notes**

Floating-point behavior matches standard library function `recipf4`.

SCE CONFIDENTIAL

# **rsqrtPerElem**

Compute the reciprocal square root of a 4D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 rsqrtPerElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

4D vector in which each element is the reciprocal square root of the corresponding element of the specified 4D vector

## **Description**

Create a 4D vector in which each element is the reciprocal square root of the corresponding element of the specified 4D vector.

## **Notes**

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# select

Conditionally select between two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 select(
                    Vector4_arg vec0,
                    Vector4_arg vec1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 4D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 4D vectors (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 select(
                    Vector4 arg vec0,
                    Vector4 arg vec1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.
<i>select1</i>	False selects the <i>vec0</i> argument; true selects the <i>vec1</i> argument.

## Return Values

Equal to *vec0* if *select1* is false, or to *vec1* if *select1* is true

## Description

Conditionally select one of the 4D vector arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 4D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 slerp(
                    float t,
                    Vector4 arg unitVec0,
                    Vector4 arg unitVec1,
                    float tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_F
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	4D vector, expected to be unit-length.
<i>unitVec1</i>	4D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 4D vector

## **Description**

Perform spherical linear interpolation between two 4D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 4D vectors (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 slerp(
                    floatInVec arg t,
                    Vector4 arg unitVec0,
                    Vector4 arg unitVec1,
                    floatInVec arg tol =
                        floatInVec(SCE_VECTORMATH_DEFAULT_SLERP_TOL_V)
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	4D vector, expected to be unit-length.
<i>unitVec1</i>	4D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 4D vector

## **Description**

Perform spherical linear interpolation between two 4D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# sqrtPerElem

Compute the square root of a 4D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector4 sqrtPerElem(
                    Vector4 arg
                );
            }
        }
    }
}
```

## Arguments

`vec`      4D vector.

## Return Values

4D vector in which each element is the square root of the corresponding element of the specified 4D vector

## Description

Create a 4D vector in which each element is the square root of the corresponding element of the specified 4D vector.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

# storeHalfFloats

Store four 4D vectors as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector4 arg vec0,
                    Vector4 arg vec1,
                    Vector4 arg vec2,
                    Vector4 arg vec3,
                    vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.
<i>vec2</i>	4D vector.
<i>vec3</i>	4D vector.
<i>twoQuads</i>	An output array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Store four 4D vectors in two quadwords of half-float values. The output is {x0, y0, z0, w0, x1, y1, z1, w1, x2, y2, z2, w2, x3, y3, z3, w3}.

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeHalfFloats

Store a 4D vector as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector4 arg vec,
                    uint16_t *hptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>hptr</i>	An output array of four half-floats.

## Return Values

None

## Description

Store a 4D vector in a `uint16_t` array of half-float values.

## Notes

This transformation may not support either denormalized numbers or NaNs. The memory area of the previous 16 bytes and the next 32 bytes from *hptr* might be accessed.

SCE CONFIDENTIAL

# storeXYZW

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a 4D vector in the first four words of a quadword array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZW(
                    Vector4 arg vec,
                    vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>quad</i>	Pointer to a quadword array in which $x$ , $y$ , $z$ , and $w$ will be stored.

## Return Values

None

## Description

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a 4D vector in the first four words of a quadword array.

SCE CONFIDENTIAL

# storeXYZW

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a 4D vector in the first four words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZW(
                    Vector4 arg vec,
                    float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>fptr</i>	An output array of float in which $x$ , $y$ , $z$ , and $w$ will be stored.

## Return Values

None

## Description

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a 4D vector in the first four words of a float array.

## Notes

Memory area of previous 16 bytes and next 32 bytes from *fptr* might be accessed.

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 4D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec sum(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec* 4D vector.

### Return Values

Sum of all elements of *vec*

### Description

Compute the sum of all elements of a 4D vector.

# 3D Point Functions

## absPerElem

Compute the absolute value of a 3D point per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 absPerElem(
                    Point3_arg pnt
                );
            }
        }
    }
}
```

### Arguments

*pnt*                    3D point.

### Return Values

3D point in which each element is the absolute value of the corresponding element of *pnt*

### Description

Compute the absolute value of each element of a 3D point.

# **allElemEqual**

All of the elements of the first 3D point are equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemEqual(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True if all elements of *pnt0* are equal to the corresponding element in *pnt1*; otherwise false.

## **Description**

All of the elements of the first 3D point are equal to the corresponding element in the second.

## **Notes**

Comparing a 3D point against itself will return false if any element has a NaN value.

# allElemGreaterThan

All of the elements of the first 3D point are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThan(
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

True if all elements of *pnt0* are greater than the corresponding element in *pnt1*; otherwise false.

## Description

All of the elements of the first 3D point are greater than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemGreaterThanOrEqualTo**

All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemGreaterThanOrEqualTo (
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True if all elements of *pnt0* are greater than or equal to the corresponding element in *pnt1*; otherwise false.

## **Description**

All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

# allElemLessThan

All of the elements of the first 3D point are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThan(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

True if all elements of *pnt0* are less than the corresponding element in *pnt1*; otherwise false.

## Description

All of the elements of the first 3D point are less than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemLessThanOrEqual**

All of the elements of the first 3D point are less than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemLessThanOrEqual (
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True if all elements of *pnt0* are less than or equal to the corresponding element in *pnt1*; otherwise false.

## **Description**

All of the elements of the first 3D point are less than or equal to the corresponding element in the second.

# **allElemNotEqual**

All of the elements of the first 3D point are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const boolInVec allElemNotEqual(
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True if all elements of *pnt0* are not equal to the corresponding element in *pnt1*; otherwise false.

## **Description**

All of the elements of the first 3D point are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 3D point between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 clampPerElem(
                    Point3 arg pnt,
                    Point3 arg clampMin,
                    Point3 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point to be clamped.
<i>clampMin</i>	3D point containing the minimum values of the range.
<i>clampMax</i>	3D point containing the maximum values of the range.

## Return Values

3D point in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 3D point to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 copySignPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element has the magnitude of the corresponding element of *pnt0* and the sign of the corresponding element of *pnt1*

## Description

For each element, create a value composed of the magnitude of *pnt0* and the sign of *pnt1*.

SCE CONFIDENTIAL

# dist

Compute the distance between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec dist(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Distance between two 3D points

## Description

Compute the distance between two 3D points.

SCE CONFIDENTIAL

# distFromOrigin

Compute the distance of a 3D point from the coordinate-system origin.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec distFromOrigin(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt* 3D point.

## Return Values

Distance of a 3D point from the origin

## Description

Compute the distance of a 3D point from the coordinate-system origin.

# **distSqr**

Compute the square of the distance between two 3D points.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec distSqr(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

Square of the distance between two 3D points

## **Description**

Compute the square of the distance between two 3D points.

# distSqrFromOrigin

Compute the square of the distance of a 3D point from the coordinate-system origin.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec distSqrFromOrigin(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Square of the distance of a 3D point from the origin

## Description

Compute the square of the distance of a 3D point from the coordinate-system origin.

# divPerElem

Divide two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 divPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the quotient of the corresponding elements of the specified 3D points

## Description

Divide two 3D points element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

# lerp

Linear interpolation between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 lerp(
                    float t,
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Interpolated 3D point

## Description

Linearly interpolate between two 3D points.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two 3D points (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 lerp(
                    floatInVec arg t,
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Interpolated 3D point

## Description

Linearly interpolate between two 3D points.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadHalfFloats

Load eight three-half-floats as 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Point3 &pnt0,
                    Point3 &pnt1,
                    Point3 &pnt2,
                    Point3 &pnt3,
                    Point3 &pnt4,
                    Point3 &pnt5,
                    Point3 &pnt6,
                    Point3 &pnt7,
                    const vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	An output 3D point.
<i>pnt1</i>	An output 3D point.
<i>pnt2</i>	An output 3D point.
<i>pnt3</i>	An output 3D point.
<i>pnt4</i>	An output 3D point.
<i>pnt5</i>	An output 3D point.
<i>pnt6</i>	An output 3D point.
<i>pnt7</i>	An output 3D point.
<i>threeQuads</i>	Array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Load eight three-half-floats as 3D points. The input is  
 $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7\}$ .

## Notes

This transformation may not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadHalfFloats

Load three half-floats as a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadHalfFloats(
                    Point3 &pnt,
                    const uint16_t *hfptr
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	An output 3D point.
<i>hfptr</i>	Array of three half-floats.

## Return Values

None

## Description

Load three half-floats as a 3D point.

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# loadXYZ

Load *x*, *y*, and *z* elements from the first three words of a quadword.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZ(
                    Point3 &pnt,
                    const vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

*pnt*

*quad*

An output 3D point.

Pointer to a quadword from which *x*, *y*, and *z* will be loaded.

## Return Values

None

## Description

Load *x*, *y*, and *z* elements from the first three words of a quadword.

SCE CONFIDENTIAL

# loadXYZ

Load *x*, *y*, and *z* elements from the first three words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZ(
                    Point3 &pnt,
                    const float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	An output 3D point.
<i>fptr</i>	Array of float from which <i>x</i> , <i>y</i> , and <i>z</i> will be loaded.

## Return Values

None

## Description

Load *x*, *y*, and *z* elements from the first three words of a float array.

SCE CONFIDENTIAL

# **loadXYZArray**

Load four three-float 3D points, stored in three quadwords.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZArray(
                    Point3 &pnt0,
                    Point3 &pnt1,
                    Point3 &pnt2,
                    Point3 &pnt3,
                    const vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	An output 3D point.
<i>pnt1</i>	An output 3D point.
<i>pnt2</i>	An output 3D point.
<i>pnt3</i>	An output 3D point.
<i>threeQuads</i>	Array of three quadwords containing 12 floats.

## **Return Values**

None

## **Description**

Load four three-float 3D points, stored in three quadwords as {x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3}, into four 3D points.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec maxElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Maximum value of all elements of *pnt*

## Description

Compute the maximum value of all elements of a 3D point.

# maxPerElem

Maximum of two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 maxPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the maximum of the corresponding elements of the specified 3D points

## Description

Create a 3D point in which each element is the maximum of the corresponding elements of the specified 3D points.

# minElem

Minimum element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec minElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Minimum value of all elements of *pnt*

## Description

Compute the minimum value of all elements of a 3D point.

# minPerElem

Minimum of two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 minPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the minimum of the corresponding elements of the specified 3D points

## Description

Create a 3D point in which each element is the minimum of the corresponding elements of two specified 3D points.

# mulPerElem

Multiply two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 mulPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the product of the corresponding elements of the specified 3D points

## Description

Multiply two 3D points element by element.

# print

Print a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Print a 3D point. Prints the 3D point transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 3D point and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Point3 arg pnt,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>name</i>	String printed with the 3D point.

## Return Values

None

## Description

Print a 3D point and an associated string identifier. Prints the 3D point transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# projection

Scalar projection of a 3D point on a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec projection(
                    Point3 arg pnt,
                    Vector3 arg unitVec
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

Scalar projection of the 3D point on the unit-length 3D vector

## Description

Scalar projection of a 3D point on a unit-length 3D vector (dot product).

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 3D point per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 recipPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

3D point in which each element is the reciprocal of the corresponding element of the specified 3D point

## **Description**

Create a 3D point in which each element is the reciprocal of the corresponding element of the specified 3D point.

## **Notes**

Floating-point behavior matches standard library function `recipf4`.

SCE CONFIDENTIAL

# rsqrtPerElem

Compute the reciprocal square root of a 3D point per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 rsqrtPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt* 3D point.

## Return Values

3D point in which each element is the reciprocal square root of the corresponding element of the specified 3D point

## Description

Create a 3D point in which each element is the reciprocal square root of the corresponding element of the specified 3D point.

## Notes

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# scale

Apply uniform scale to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 scale(
                    Point3 arg pnt,
                    float scaleVal
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>scaleVal</i>	Scalar value.

## Return Values

3D point in which every element is multiplied by the scalar value

## Description

Apply uniform scale to a 3D point.

# scale

Apply uniform scale to a 3D point (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 scale(
                    Point3 arg pnt,
                    floatInVec arg scaleVal
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>scaleVal</i>	Scalar value (stored in vector data type).

## Return Values

3D point in which every element is multiplied by the scalar value

## Description

Apply uniform scale to a 3D point.

SCE CONFIDENTIAL

# scale

Apply non-uniform scale to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 scale(
                    Point3 arg pnt,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>scaleVec</i>	3D vector.

## Return Values

3D point in which each element is the product of the corresponding elements of the specified 3D point and 3D vector

## Description

Apply non-uniform scale to a 3D point.

SCE CONFIDENTIAL

# select

Conditionally select between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 select(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>select1</i>	False selects the <i>pnt0</i> argument; true selects the <i>pnt1</i> argument.

## Return Values

Equal to *pnt0* if *select1* is false, or to *pnt1* if *select1* is true.

## Description

Conditionally select one of the 3D point arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

SCE CONFIDENTIAL

# select

Conditionally select between two 3D points (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 select(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>select1</i>	False selects the <i>pnt0</i> argument; true selects the <i>pnt1</i> argument.

## Return Values

Equal to *pnt0* if *select1* is false, or to *pnt1* if *select1* is true.

## Description

Conditionally select one of the 3D point arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

# **sqrtPerElem**

Compute the square root of a 3D point per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Point3 sqrtPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

3D point in which each element is the square root of the corresponding element of the specified 3D point

## **Description**

Create a 3D point in which each element is the square root of the corresponding element of the specified 3D point.

## **Notes**

Floating-point behavior matches standard library function `sqrtf4`.

# storeHalfFloats

Store eight 3D points as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    Point3 arg pnt2,
                    Point3 arg pnt3,
                    Point3 arg pnt4,
                    Point3 arg pnt5,
                    Point3 arg pnt6,
                    Point3 arg pnt7,
                    vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>pnt2</i>	3D point.
<i>pnt3</i>	3D point.
<i>pnt4</i>	3D point.
<i>pnt5</i>	3D point.
<i>pnt6</i>	3D point.
<i>pnt7</i>	3D point.
<i>threeQuads</i>	An output array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Store eight 3D points in three quadwords of half-float values. The output is  $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7\}$ .

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeHalfFloats

Store a 3D point as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Point3 arg pnt,
                    uint16_t *hfptr
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>hfptr</i>	An output array of three half-floats.

## Return Values

None

## Description

Store a 3D point in a `uint16_t` array of half-float values.

## Notes

This transformation does not support either denormalized numbers or NaNs. The memory area of the previous 16 bytes and the next 32 bytes from *hfptr* might be accessed.

SCE CONFIDENTIAL

# storeXYZ

Store *x*, *y*, and *z* elements of a 3D point in the first three words of a quadword.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZ(
                    Point3 arg pnt,
                    vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>quad</i>	Pointer to a quadword in which <i>x</i> , <i>y</i> , and <i>z</i> will be stored.

## Return Values

None

## Description

Store *x*, *y*, and *z* elements of a 3D point in the first three words of a quadword. The value of the fourth word (the word with the highest address) remains unchanged.

SCE CONFIDENTIAL

# storeXYZ

Store *x*, *y*, and *z* elements of a 3D point in the first three words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZ(
                    Point3 arg pnt,
                    float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>fptr</i>	An output array of float in which <i>x</i> , <i>y</i> , and <i>z</i> will be stored.

## Return Values

None

## Description

Store *x*, *y*, and *z* elements of a 3D point in the first three words of a float array.

## Notes

Memory area of previous 16 bytes and next 32 bytes from *fptr* might be accessed.

SCE CONFIDENTIAL

# storeXYZArray

Store four 3D points in three quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZArray(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    Point3 arg pnt2,
                    Point3 arg pnt3,
                    vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>pnt2</i>	3D point.
<i>pnt3</i>	3D point.
<i>threeQuads</i>	An output array of three quadwords containing 12 floats.

## Return Values

None

## Description

Store four 3D points in three quadwords as {x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3}.

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 3D point.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec sum(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

### Arguments

*pnt*                    3D point.

### Return Values

Sum of all elements of *pnt*

### Description

Compute the sum of all elements of a 3D point.

# Quaternion Functions

## **conj**

Compute the conjugate of a quaternion.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat conj(
                    Quat_arg quat
                );
            }
        }
    }
}
```

### **Arguments**

*quat*      Quaternion.

### **Return Values**

Conjugate of the specified quaternion

### **Description**

Compute the conjugate of a quaternion.

SCE CONFIDENTIAL

# dot

Compute the dot product of two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec dot(
                    Quat arg quat0,
                    Quat arg quat1
                );
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.

## Return Values

Dot product of the specified quaternions

## Description

Compute the dot product of two quaternions.

SCE CONFIDENTIAL

# length

Compute the length of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec length(
                    Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

Length of the specified quaternion

## Description

Compute the length of a quaternion.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a quaternion.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec lengthSqr(  
                    Quat arg quat  
                );
            }
        }
    }
}
```

## **Arguments**

*quat*                  Quaternion.

## **Return Values**

Square of the length of the specified quaternion.

## **Description**

Compute the square of the length of a quaternion.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat lerp(
                    float t,
                    Quat_arg quat0,
                    Quat_arg quat1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.

## Return Values

Interpolated quaternion

## Description

Linearly interpolate between two quaternions.

## Notes

Does not clamp *t* between 0 and 1.

# lerp

Linear interpolation between two quaternions (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat lerp(
                    floatInVec arg t,
                    Quat arg quat0,
                    Quat arg quat1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.

## Return Values

Interpolated quaternion

## Description

Linearly interpolate between two quaternions.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadXYZW

Load *x*, *y*, *z*, and *w* elements from the first four words of a quadword.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZW(
                    Quat &quat,
                    const vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

*quat*  
*quad*

An output quaternion.  
Pointer to a quadword from which *x*, *y*, *z*, and *w* will be loaded.

## Return Values

None

## Description

Load *x*, *y*, *z*, and *w* elements from the first four words of a quadword.

SCE CONFIDENTIAL

# loadXYZW

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void loadXYZW(
                    Quat &quat,
                    const float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>quat</i>	An output quaternion.
<i>fptr</i>	Array of float from which $x$ , $y$ , $z$ , and $w$ will be loaded.

## Return Values

None

## Description

Load  $x$ ,  $y$ ,  $z$ , and  $w$  elements from the first four words of a float array.

SCE CONFIDENTIAL

## norm

Compute the norm of a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec norm(
                    Quat arg quat
                );
            }
        }
    }
}
```

### Arguments

*quat*      Quaternion.

### Return Values

The norm of the specified quaternion

### Description

Compute the norm, equal to the square of the length, of a quaternion.

SCE CONFIDENTIAL

# normalize

Normalize a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat normalize(
                    Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

The specified quaternion scaled to unit length

## Description

Compute a normalized quaternion.

## Notes

The result is unpredictable when all elements of *quat* are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a quaternion by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat operator*(
                    float scalar,
                    Quat_arg quat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>quat</i>	Quaternion.

## Return Values

Scalar product of *quat* and *scalar*

## Description

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a quaternion by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat operator*(
                    floatInVec arg scalar,
                    Quat arg quat
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>quat</i>	Quaternion.

## **Return Values**

Scalar product of *quat* and *scalar*

## **Description**

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# print

Print a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

None

## Description

Print a quaternion.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a quaternion and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Quat arg quat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>quat</i>	Quaternion.
<i>name</i>	String printed with the quaternion.

## Return Values

None

## Description

Print a quaternion and an associated string identifier.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# rotate

Use a unit-length quaternion to rotate a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Vector3 rotate(
                    Quat arg unitQuat,
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
<i>vec</i>	3D vector.

## Return Values

The rotated 3D vector, equivalent to  $\text{unitQuat}^* \text{Quat}(\text{vec}, 0)^* \text{conj}(\text{unitQuat})$

## Description

Rotate a 3D vector by applying a unit-length quaternion.

SCE CONFIDENTIAL

# select

Conditionally select between two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat select(
                    Quat arg quat0,
                    Quat arg quat1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.
<i>select1</i>	False selects the <i>quat0</i> argument; true selects the <i>quat1</i> argument.

## Return Values

Equal to *quat0* if *select1* is false, or to *quat1* if *select1* is true

## Description

Conditionally select one of the quaternion arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

SCE CONFIDENTIAL

# select

Conditionally select between two quaternions (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat select(
                    Quat arg quat0,
                    Quat arg quat1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.
<i>select1</i>	False selects the <i>quat0</i> argument; true selects the <i>quat1</i> argument.

## Return Values

Equal to *quat0* if *select1* is false, or to *quat1* if *select1* is true

## Description

Conditionally select one of the quaternion arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two quaternions.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat slerp(
                    float t,
                    Quat_arg unitQuat0,
                    Quat_arg unitQuat1,
                    float tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_F
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitQuat0</i>	Quaternion, expected to be unit-length.
<i>unitQuat1</i>	Quaternion, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated quaternion

## **Description**

Perform spherical linear interpolation between two quaternions.

## **Notes**

Interpolates along the shortest path between orientations. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two quaternions (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat slerp(
                    floatInVec arg t,
                    Quat arg unitQuat0,
                    Quat arg unitQuat1,
                    floatInVec arg tol =
                    floatInVec(SCE_VECTORMATH_DEFAULT_SLERP_TOL_V)
                );
            }
        }
    }
}
```

## **Arguments**

<u>t</u>	Interpolation parameter.
<u>unitQuat0</u>	Quaternion, expected to be unit-length.
<u>unitQuat1</u>	Quaternion, expected to be unit-length.
<u>tol</u>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated quaternion

## **Description**

Perform spherical linear interpolation between two quaternions.

## **Notes**

Interpolates along the shortest path between orientations. Does not clamp t between 0 and 1.

SCE CONFIDENTIAL

# squad

Spherical quadrangle interpolation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat squad(
                    float t,
                    Quat_arg unitQuat0,
                    Quat_arg unitQuat1,
                    Quat_arg unitQuat2,
                    Quat_arg unitQuat3
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>unitQuat0</i>	Quaternion, expected to be unit-length.
<i>unitQuat1</i>	Quaternion, expected to be unit-length.
<i>unitQuat2</i>	Quaternion, expected to be unit-length.
<i>unitQuat3</i>	Quaternion, expected to be unit-length.

## Return Values

Interpolated quaternion

## Description

Perform spherical quadrangle interpolation between four quaternions.

# squad

Spherical quadrangle interpolation (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Quat squad(
                    floatInVec arg t,
                    Quat arg unitQuat0,
                    Quat arg unitQuat1,
                    Quat arg unitQuat2,
                    Quat arg unitQuat3
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>unitQuat0</i>	Quaternion, expected to be unit-length.
<i>unitQuat1</i>	Quaternion, expected to be unit-length.
<i>unitQuat2</i>	Quaternion, expected to be unit-length.
<i>unitQuat3</i>	Quaternion, expected to be unit-length.

## Return Values

Interpolated quaternion

## Description

Perform spherical quadrangle interpolation between four quaternions.

SCE CONFIDENTIAL

# storeXYZW

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a quaternion in the first four words of a quadword array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZW(
                    Quat arg quat,
                    vec_float4 *quad
                );
            }
        }
    }
}
```

## Arguments

<i>quat</i>	Quaternion.
<i>quad</i>	Pointer to a doubleword array in which $x$ , $y$ , $z$ , and $w$ will be stored.

## Return Values

None

## Description

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a quaternion in the first four words of a quadword array.

SCE CONFIDENTIAL

# storeXYZW

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a quaternion in the first four words of a float array.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void storeXYZW(
                    Quat arg quat,
                    float *fptr
                );
            }
        }
    }
}
```

## Arguments

<i>quat</i>	Quaternion.
<i>fptr</i>	An output array of float in which $x$ , $y$ , $z$ , and $w$ will be stored.

## Return Values

None

## Description

Store  $x$ ,  $y$ ,  $z$ , and  $w$  elements of a quaternion in the first four words of a float array.

## Notes

Memory area of previous 16 bytes and next 32 bytes from *fptr* might be accessed.

# 3x3 Matrix Functions

## absPerElem

Compute the absolute value of a 3x3 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 absPerElem(
                    Matrix3_arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat*                    3x3 matrix.

### Return Values

3x3 matrix in which each element is the absolute value of the corresponding element of the specified 3x3 matrix

### Description

Compute the absolute value of each element of a 3x3 matrix.

# appendScale

Append (post-multiply) a scale transformation to a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 appendScale(
                    Matrix3 arg mat,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 3x3 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# determinant

Determinant of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec determinant(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 3x3 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 inverse(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 3x3 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# **mulPerElem**

Multiply two 3x3 matrices per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 mulPerElem(
                    Matrix3 arg mat0,
                    Matrix3 arg mat1
                );
            }
        }
    }
}
```

## **Arguments**

<i>mat0</i>	3x3 matrix.
<i>mat1</i>	3x3 matrix.

## **Return Values**

3x3 matrix in which each element is the product of the corresponding elements of the specified 3x3 matrices

## **Description**

Multiply two 3x3 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 3x3 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 operator*(
                    float scalar,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	3x3 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 3x3 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 operator*(
                    floatInVec arg scalar,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>mat</i>	3x3 matrix.

## **Return Values**

Scalar product of *mat* and *scalar*

## **Description**

Multiply a 3x3 matrix by a scalar.

# prependScale

Prepend (pre-multiply) a scale transformation to a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 prependScale(
                    Vector3 arg scaleVec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 3x3 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      3x3 matrix.

## Return Values

None

## Description

Print a 3x3 matrix. Unlike the printing of vectors, the 3x3 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# print

Print a 3x3 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix3 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>name</i>	String printed with the 3x3 matrix.

## Return Values

None

## Description

Print a 3x3 matrix and an associated string identifier. Unlike the printing of vectors, the 3x3 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 select(
                    Matrix3 arg mat0,
                    Matrix3 arg mat1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	3x3 matrix.
<i>mat1</i>	3x3 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 3x3 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 3x3 matrices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 select(
                    Matrix3 arg mat0,
                    Matrix3 arg mat1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	3x3 matrix.
<i>mat1</i>	3x3 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 3x3 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix3 transpose(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 3x3 matrix.

# 4x4 Matrix Functions

## absPerElem

Compute the absolute value of a 4x4 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 absPerElem(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat* 4x4 matrix.

### Return Values

4x4 matrix in which each element is the absolute value of the corresponding element of the specified 4x4 matrix

### Description

Compute the absolute value of each element of a 4x4 matrix.

SCE CONFIDENTIAL

# affineInverse

Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 affineInverse(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

Inverse of the specified 4x4 matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 4x4 matrix as *M*, and its translation component as *v*, compute a matrix whose upper-left 3x3 submatrix is `inverse(M)`, whose translation vector is `-inverse(M) * v`, and whose bottom row is (0,0,0,1).

## Notes

This can be used to achieve better performance than a general inverse when the specified 4x4 matrix meets the given restrictions. The result is unpredictable when the determinant of *mat* is equal to or near 0.

SCE CONFIDENTIAL

# appendScale

Append (post-multiply) a scale transformation to a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 appendScale(
                    Matrix4 arg mat,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	4x4 matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 4x4 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# determinant

Determinant of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const floatInVec determinant(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 4x4 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 inverse(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 4x4 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# mulPerElem

Multiply two 4x4 matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 mulPerElem(
                    Matrix4 arg mat0,
                    Matrix4 arg mat1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	4x4 matrix.
<i>mat1</i>	4x4 matrix.

## Return Values

4x4 matrix in which each element is the product of the corresponding elements of the specified 4x4 matrices

## Description

Multiply two 4x4 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 operator*(
                    float scalar,
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	4x4 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 4x4 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 operator*(
                    floatInVec arg scalar,
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
<i>mat</i>	4x4 matrix.

## **Return Values**

Scalar product of *mat* and *scalar*

## **Description**

Multiply a 4x4 matrix by a scalar.

# orthoInverse

Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix with an orthogonal upper-left 3x3 submatrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 orthoInverse(
                    Matrix4_arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Inverse of the specified 4x4 matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 4x4 matrix as  $M$ , and its translation component as  $v$ , compute a matrix whose upper-left 3x3 submatrix is  $\text{transpose}(M)$ , whose translation vector is  $-\text{transpose}(M) * v$ , and whose bottom row is  $(0,0,0,1)$ .

## Notes

This can be used to achieve better performance than a general inverse when the specified 4x4 matrix meets the given restrictions.

# prependScale

Prepend (pre-multiply) a scale transformation to a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 prependScale(
                    Vector3 arg scaleVec,
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>mat</i>	4x4 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 4x4 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      4x4 matrix.

## Return Values

None

## Description

Print a 4x4 matrix. Unlike the printing of vectors, the 4x4 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# print

Print a 4x4 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Matrix4 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	4x4 matrix.
<i>name</i>	String printed with the 4x4 matrix.

## Return Values

None

## Description

Print a 4x4 matrix and an associated string identifier. Unlike the printing of vectors, the 4x4 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 select(
                    Matrix4 arg mat0,
                    Matrix4 arg mat1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	4x4 matrix.
<i>mat1</i>	4x4 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 4x4 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

# select

Conditionally select between two 4x4 matrices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 select(
                    Matrix4 arg mat0,
                    Matrix4 arg mat1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	4x4 matrix.
<i>mat1</i>	4x4 matrix.
<i>select1</i>	False selects the <i>mat0</i> argument; true selects the <i>mat1</i> argument.

## Return Values

Equal to *mat0* if *select1* is false, or to *mat1* if *select1* is true

## Description

Conditionally select one of the 4x4 matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Matrix4 transpose(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 4x4 matrix.

# 3x4 Transformation Matrix Functions

## absPerElem

Compute the absolute value of a 3x4 transformation matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 absPerElem(
                    Transform3 arg
                );
            }
        }
    }
}
```

### Arguments

*tfrm*      3x4 transformation matrix.

### Return Values

3x4 transformation matrix in which each element is the absolute value of the corresponding element of the specified 3x4 transformation matrix

### Description

Compute the absolute value of each element of a 3x4 transformation matrix.

# appendScale

Append (post-multiply) a scale transformation to a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 appendScale(
                    Transform3 arg tfrm,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x4 transformation matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *tfrm* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 3x4 transformation matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

# inverse

Inverse of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 inverse(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

Inverse of *tfrm*

## Description

Compute the inverse of a 3x4 transformation matrix.

## Notes

Result is unpredictable when the determinant of the left 3x3 submatrix is equal to or near 0.

# **mulPerElem**

Multiply two 3x4 transformation matrices per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 mulPerElem(
                    Transform3 arg tfrm0,
                    Transform3 arg tfrm1
                );
            }
        }
    }
}
```

## **Arguments**

<i>tfrm0</i>	3x4 transformation matrix.
<i>tfrm1</i>	3x4 transformation matrix.

## **Return Values**

3x4 transformation matrix in which each element is the product of the corresponding elements of the specified 3x4 transformation matrices

## **Description**

Multiply two 3x4 transformation matrices element by element.

SCE CONFIDENTIAL

# ortholInverse

Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 orthoInverse(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*                    3x4 transformation matrix.

## Return Values

Inverse of the specified 3x4 transformation matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 3x4 transformation matrix as  $M$ , and its translation component as  $v$ , compute a matrix whose upper-left 3x3 submatrix is  $\text{transpose}(M)$ , and whose translation vector is  $-\text{transpose}(M) * v$ .

## Notes

This can be used to achieve better performance than a general inverse when the specified 3x4 transformation matrix meets the given restrictions.

# prependScale

Prepend (pre-multiply) a scale transformation to a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 prependScale(
                    Vector3 arg scaleVec,
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>tfrm</i>	3x4 transformation matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *tfrm*

## Description

Pre-multiply a 3x4 transformation matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

None

## Description

Print a 3x4 transformation matrix. Unlike the printing of vectors, the 3x4 transformation matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 3x4 transformation matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE void print(
                    Transform3 arg tfrm,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x4 transformation matrix.
<i>name</i>	String printed with the 3x4 transformation matrix.

## Return Values

None

## Description

Print a 3x4 transformation matrix and an associated string identifier. Unlike the printing of vectors, the 3x4 transformation matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 3x4 transformation matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 select(
                    Transform3 arg tfrm0,
                    Transform3 arg tfrm1,
                    bool select1
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm0</i>	3x4 transformation matrix.
<i>tfrm1</i>	3x4 transformation matrix.
<i>select1</i>	False selects the <i>tfrm0</i> argument; true selects the <i>tfrm1</i> argument.

## Return Values

Equal to *tfrm0* if *select1* is false, or to *tfrm1* if *select1* is true

## Description

Conditionally select one of the 3x4 transformation matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a SIMD register may use more processing time than a branch. Use the [boolInVec](#) version for better performance.

SCE CONFIDENTIAL

# select

Conditionally select between two 3x4 transformation matrices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                SCE_VECTORMATH_INLINE const Transform3 select(
                    Transform3 arg tfrm0,
                    Transform3 arg tfrm1,
                    boolInVec arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm0</i>	3x4 transformation matrix.
<i>tfrm1</i>	3x4 transformation matrix.
<i>select1</i>	False selects the <i>tfrm0</i> argument; true selects the <i>tfrm1</i> argument.

## Return Values

Equal to *tfrm0* if *select1* is false, or to *tfrm1* if *select1* is true

## Description

Conditionally select one of the 3x4 transformation matrix arguments.

## Notes

This function uses a conditional select instruction to avoid a branch.

**sce::Vectormath::Simd::Aos::Matrix2**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Matrix2

A 2x2 matrix in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Matrix2 {};
```

### Description

A class representing a 2x2 matrix stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a a 2x2 matrix.
<a href="#">getCol</a>	Get the column of a 2x2 matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 2x2 matrix.
<a href="#">getCol1</a>	Get column 1 of a 2x2 matrix.
<a href="#">getElem</a>	Get the element of a 2x2 matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 2x2 matrix referred to by the specified index.
<a href="#">identity</a>	Construct an identity 2x2 matrix.
<a href="#">Matrix2</a>	Default constructor; does no initialization.
<a href="#">Matrix2</a>	Copy constructor.
<a href="#">Matrix2</a>	Construct a 2x2 matrix containing the specified columns.
<a href="#">Matrix2</a>	Set all elements of a 2x2 matrix to the same scalar value.
<a href="#">Matrix2</a>	Set all elements of a 2x2 matrix to the same scalar value (scalar data contained in vector data type).
<a href="#">Matrix2</a>	Construct a 2x2 matrix containing the specified vector float data.
<a href="#">Matrix2</a>	Construct a 2x2 matrix containing the specified 4D vector data.
<a href="#">operator*</a>	Multiply a 2x2 matrix by a scalar.
<a href="#">operator*</a>	Multiply a 2x2 matrix by a scalar (scalar data contained in vector data type).
<a href="#">operator*</a>	Multiply a 2x2 matrix by a 2D vector.
<a href="#">operator*</a>	Multiply two 2x2 matrices.
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a 2x2 matrix.
<a href="#">operator+</a>	Add two 2x2 matrices.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 2x2 matrix.
<a href="#">operator-</a>	Subtract a 2x2 matrix from another 2x2 matrix.
<a href="#">operator-</a>	Negate all elements of a 2x2 matrix.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 2x2 matrix.
<a href="#">operator=</a>	Assign one 2x2 matrix to another.
<a href="#">operator[]</a>	Subscripting operator to set or get a column.
<a href="#">rotation</a>	Construct a 2x2 matrix to rotate.
<a href="#">rotation</a>	Construct a 2x2 matrix to rotate (scalar data contained in vector data type).
<a href="#">scale</a>	Construct a 2x2 matrix to perform scaling.
<a href="#">setCol</a>	Set the column of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>setCol0</u></a>	Set column 0 of a 2x2 matrix.
<a href="#"><u>setCol1</u></a>	Set column 1 of a 2x2 matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 2x2 matrix referred to by column and row indices.
<a href="#"><u>setElem</u></a>	Set the element of a 2x2 matrix referred to by column and row indices (scalar data contained in vector data type).
<a href="#"><u>setRow</u></a>	Set the row of a 2x2 matrix referred to by the specified index.
<a href="#"><u>zero</u></a>	Construct a zero 2x2 matrix.

# Constructors and Destructors

## Matrix2

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix2

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        const Matrix2 &mat
                    );
                };
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix2

Construct a 2x2 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        Vector2 arg col0,
                        Vector2 arg col1
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	2D vector.
<i>col1</i>	2D vector.

## Return Values

None

## Description

Construct a 2x2 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix2

Set all elements of a 2x2 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 2x2 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix2

Set all elements of a 2x2 matrix to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 2x2 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix2

Construct a 2x2 matrix containing the specified vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        vec_float4_arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat* Vector float.

## Return Values

None

## Description

Construct a 2x2 matrix containing the specified vector float data. The first column is constructed from elements *x* and *y* from the specified vector float. The second column is constructed from elements *z* and *w* from the specified vector float.

SCE CONFIDENTIAL

# Matrix2

Construct a 2x2 matrix containing the specified 4D vector data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        Vector4 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                  Vector float.

## Return Values

None

## Description

Construct a 2x2 matrix containing the specified 4D vector data. The first column is constructed from elements *x* and *y* from the specified 4D vector. The second column is constructed from elements *z* and *w* from the specified 4D vector.

# Operator Methods

## **operator\***

Multiply a 2x2 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*            Scalar value.

### **Return Values**

Product of the specified 2x2 matrix and scalar

### **Description**

Multiply a 2x2 matrix by a scalar.

SCE CONFIDENTIAL

# operator\*

Multiply a 2x2 matrix by a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

Product of the specified 2x2 matrix and scalar

## Description

Multiply a 2x2 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 2x2 matrix by a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator*(
                        Vector2 arg
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

vec                  2D vector.

## **Return Values**

Product of the specified 2x2 matrix and 2D vector

## **Description**

Multiply a 2x2 matrix by a 2D vector.

SCE CONFIDENTIAL

# **operator\***

Multiply two 2x2 matrices.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator*(
                        Matrix2 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

Product of the specified 2x2 matrices

## **Description**

Multiply two 2x2 matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator*=(  

                        floatInVec arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 2x2 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator*=(  

                        Matrix2 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Perform compound assignment and multiplication by a 2x2 matrix.

SCE CONFIDENTIAL

# operator+

Add two 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator+
                        (Matrix2 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

Sum of the specified 2x2 matrices

## Description

Add two 2x2 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator+=(
                        Matrix2 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat* 2x2 matrix.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Perform compound assignment and addition with a 2x2 matrix.

SCE CONFIDENTIAL

# **operator-**

Subtract a 2x2 matrix from another 2x2 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator-
                        (Matrix2 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

Difference of the specified 2x2 matrices

## **Description**

Subtract a 2x2 matrix from another 2x2 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 2x2 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

2x2 matrix containing negated elements of the specified 2x2 matrix

## **Description**

Negate all elements of a 2x2 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator==(Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Perform compound assignment and subtraction by a 2x2 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 2x2 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2 &operator=(Matrix2 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Assign one 2x2 matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator[](int col) const;
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-1.

## **Return Values**

The column referred to by the specified index

## **Description**

Subscripting operator invoked when applied to non-const [Matrix2](#).

# Public Static Methods

## identity

Construct an identity 2x2 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2x2 matrix

### Description

Construct an identity 2x2 matrix in which non-diagonal elements are zero and diagonal elements are 1.

SCE CONFIDENTIAL

# rotation

Construct a 2x2 matrix to rotate.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 rotation(
                        float radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 2x2 matrix

## Description

Construct a 2x2 matrix to rotate by the specified radians angle.

# rotation

Construct a 2x2 matrix to rotate (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 rotation(
                        floatInVec_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value (stored in vector data type).

## Return Values

The constructed 2x2 matrix

## Description

Construct a 2x2 matrix to rotate by the specified radians angle.

SCE CONFIDENTIAL

# scale

Construct a 2x2 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 scale(
                        Vector2 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      2D vector.

## Return Values

The constructed 2x2 matrix

## Description

Construct a 2x2 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

## zero

Construct a zero 2x2 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2x2 matrix

### Description

Construct a zero 2x2 matrix in which all elements are zero.

# Public Instance Methods

## get128

Get vector float data from a a 2x2 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a a 2x2 matrix. Elements *x* and *y* are constructed from the first column. Elements *z* and *w* are constructed from the second column.

SCE CONFIDENTIAL

# getCol

Get the column of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-1.

## Return Values

The column referred to by the specified index

## Description

Get the column of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 2x2 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 2x2 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 2x2 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>row</i>	Index, expected in the range 0-1.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Get the element of a 2x2 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-1.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol

Set the column of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setCol(
                        int col,
                        Vector2 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>vec</i>	2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the column of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2 &setCol0(
                        Vector2 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set column 0 of a 2x2 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2 &setCol1(
                        Vector2 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set column 1 of a 2x2 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 2x2 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setElem(
                        int col,
                        int row,
                        float val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>row</i>	Index, expected in the range 0-1.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the element of a 2x2 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setElem

Set the element of a 2x2 matrix referred to by column and row indices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setElem(
                        int col,
                        int row,
                        floatInVec arg val
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>row</i>	Index, expected in the range 0-1.
<i>val</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the element of a 2x2 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setRow(
                        int row,
                        Vector2 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-1.
<i>vec</i>	2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the row of a 2x2 matrix referred to by the specified index.

**sce::Vectormath::Simd::Aos::Matrix3**

000004892117

## Summary

# **sce::Vectormath::Simd::Aos::Matrix3**

A 3x3 matrix in array-of-structures format.

## Definition

```
#include <vectormath.h>
class Matrix3 {};
```

## Description

A class representing a 3x3 matrix stored in array-of-structures (AoS) format.

## Methods Summary

Methods	Description
<a href="#">getCol</a>	Get the column of a 3x3 matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 3x3 matrix.
<a href="#">getCol1</a>	Get column 1 of a 3x3 matrix.
<a href="#">getCol2</a>	Get column 2 of a 3x3 matrix.
<a href="#">getElem</a>	Get the element of a 3x3 matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 3x3 matrix referred to by the specified index.
<a href="#">identity</a>	Construct an identity 3x3 matrix.
<a href="#">Matrix3</a>	Default constructor; does no initialization.
<a href="#">Matrix3</a>	Copy constructor.
<a href="#">Matrix3</a>	Construct a 3x3 matrix containing the specified columns.
<a href="#">Matrix3</a>	Construct a 3x3 rotation matrix from a unit-length quaternion.
<a href="#">Matrix3</a>	Set all elements of a 3x3 matrix to the same scalar value.
<a href="#">Matrix3</a>	Set all elements of a 3x3 matrix to the same scalar value (scalar data contained in vector data type).
<a href="#">operator*</a>	Multiply a 3x3 matrix by a scalar.
<a href="#">operator*</a>	Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).
<a href="#">operator*</a>	Multiply a 3x3 matrix by a 3D vector.
<a href="#">operator*</a>	Multiply two 3x3 matrices.
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#">operator*=<a></a></a>	Perform compound assignment and multiplication by a 3x3 matrix.
<a href="#">operator+<a></a></a>	Add two 3x3 matrices.
<a href="#">operator+=<a></a></a>	Perform compound assignment and addition with a 3x3 matrix.
<a href="#">operator-<a></a></a>	Subtract a 3x3 matrix from another 3x3 matrix.
<a href="#">operator-<a></a></a>	Negate all elements of a 3x3 matrix.
<a href="#">operator-=<a></a></a>	Perform compound assignment and subtraction by a 3x3 matrix.
<a href="#">operator=<a></a></a>	Assign one 3x3 matrix to another.
<a href="#">operator[]</a>	Subscripting operator to set or get a column.
<a href="#">operator[]</a>	Subscripting operator to get a column.
<a href="#">rotation</a>	Construct a 3x3 matrix to rotate around a unit-length 3D vector.
<a href="#">rotation</a>	Construct a 3x3 matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).
<a href="#">rotation</a>	Construct a rotation matrix from a unit-length quaternion.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>rotationX</u></a>	Construct a 3x3 matrix to rotate around the x axis.
<a href="#"><u>rotationX</u></a>	Construct a 3x3 matrix to rotate around the x axis (scalar data contained in vector data type).
<a href="#"><u>rotationY</u></a>	Construct a 3x3 matrix to rotate around the y axis.
<a href="#"><u>rotationY</u></a>	Construct a 3x3 matrix to rotate around the y axis (scalar data contained in vector data type).
<a href="#"><u>rotationZ</u></a>	Construct a 3x3 matrix to rotate around the z axis.
<a href="#"><u>rotationZ</u></a>	Construct a 3x3 matrix to rotate around the z axis (scalar data contained in vector data type).
<a href="#"><u>rotationZYX</u></a>	Construct a 3x3 matrix to rotate around the x, y, and z axes.
<a href="#"><u>scale</u></a>	Construct a 3x3 matrix to perform scaling.
<a href="#"><u>setCol</u></a>	Set the column of a 3x3 matrix referred to by the specified index.
<a href="#"><u>setCol0</u></a>	Set column 0 of a 3x3 matrix.
<a href="#"><u>setCol1</u></a>	Set column 1 of a 3x3 matrix.
<a href="#"><u>setCol2</u></a>	Set column 2 of a 3x3 matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 3x3 matrix referred to by column and row indices.
<a href="#"><u>setElem</u></a>	Set the element of a 3x3 matrix referred to by column and row indices (scalar data contained in vector data type).
<a href="#"><u>setRow</u></a>	Set the row of a 3x3 matrix referred to by the specified index.
<a href="#"><u>zero</u></a>	Construct a zero 3x3 matrix.

# Constructors and Destructors

## Matrix3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3(
                        const Matrix3 &mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix3

Construct a 3x3 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3(
                        Vector3 arg col0,
                        Vector3 arg col1,
                        Vector3 arg col2
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	3D vector.
<i>col1</i>	3D vector.
<i>col2</i>	3D vector.

## Return Values

None

## Description

Construct a 3x3 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix3

Construct a 3x3 rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    explicit SCE_VECTORMATH_INLINE Matrix3(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

None

## Description

Construct a 3x3 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# Matrix3

Set all elements of a 3x3 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix3(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 3x3 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix3

Set all elements of a 3x3 matrix to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix3(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3x3 matrix with all elements set to the scalar value argument.

# Operator Methods

## **operator\***

Multiply a 3x3 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 3x3 matrix and scalar

### **Description**

Multiply a 3x3 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Product of the specified 3x3 matrix and scalar

## **Description**

Multiply a 3x3 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x3 matrix by a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        Vector3 arg
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

vec                   3D vector.

## **Return Values**

Product of the specified 3x3 matrix and 3D vector

## **Description**

Multiply a 3x3 matrix by a 3D vector.

SCE CONFIDENTIAL

# operator\*

Multiply two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator*(
                        Matrix3 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Product of the specified 3x3 matrices

## Description

Multiply two 3x3 matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator*=(  

                        floatInVec arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator*=(  

                        Matrix3 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Perform compound assignment and multiplication by a 3x3 matrix.

SCE CONFIDENTIAL

# operator+

Add two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator+
                        (Matrix3 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Sum of the specified 3x3 matrices

## Description

Add two 3x3 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator+=(  
                        Matrix3 arg mat  
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Perform compound assignment and addition with a 3x3 matrix.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3x3 matrix from another 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator-
                        (Matrix3 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

Difference of the specified 3x3 matrices

## **Description**

Subtract a 3x3 matrix from another 3x3 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

3x3 matrix containing negated elements of the specified 3x3 matrix

## **Description**

Negate all elements of a 3x3 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator==(Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Perform compound assignment and subtraction by a 3x3 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 3x3 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3 &operator=(Matrix3 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Assign one 3x3 matrix to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-2.

## Return Values

A reference to the indexed column

## Description

Subscripting operator invoked when applied to non-const [Matrix3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-2.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Matrix3](#).

# Public Static Methods

## identity

Construct an identity 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x3 matrix

### Description

Construct an identity 3x3 matrix in which non-diagonal elements are zero and diagonal elements are 1.

# rotation

Construct a 3x3 matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotation(
                        float radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a 3x3 matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotation(
                        floatInVec arg radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 3x3 matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationX(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationX

Construct a 3x3 matrix to rotate around the x axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationX(
                        floatInVec_arg radians
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 3x3 matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationY(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 3x3 matrix to rotate around the y axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationY(
                        floatInVec_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 3x3 matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationZ(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 3x3 matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationZ(
                        floatInVec_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value (stored in vector data type).

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 3x3 matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                }
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 3x3 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      3D vector.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

## zero

Construct a zero 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x3 matrix

### Description

Construct a zero 3x3 matrix in which all elements are zero.

# Public Instance Methods

## getCol

Get the column of a 3x3 matrix referred to by the specified index.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

*col* Index, expected in the range 0-2.

### Return Values

The column referred to by the specified index

### Description

Get the column of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 3x3 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 3x3 matrix.

SCE CONFIDENTIAL

## getCol2

Get column 2 of a 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol2() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Column 2

### Description

Get column 2 of a 3x3 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 3x3 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>row</i>	Index, expected in the range 0-2.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 3x3 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-2.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol

Set the column of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setCol(
                        int col,
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the column of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3 &setCol0(
                        Vector3 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 0 of a 3x3 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3 &setCol1(
                        Vector3 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 1 of a 3x3 matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3 &setCol2(
                        Vector3 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 2 of a 3x3 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 3x3 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setElem(
                        int col,
                        int row,
                        float val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the element of a 3x3 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# **setElem**

Set the element of a 3x3 matrix referred to by column and row indices (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setElem(
                        int col,
                        int row,
                        floatInVec arg val
                    );
                };
            }
        }
    }
}
```

## **Arguments**

<i>col</i>	Index, expected in the range 0-2.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value (stored in vector data type).

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Set the element of a 3x3 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setRow(
                        int row,
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-2.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the row of a 3x3 matrix referred to by the specified index.

**sce::Vectormath::Simd::Aos::Matrix4**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Matrix4

A 4x4 matrix in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Matrix4 {};
```

### Description

A class representing a 4x4 matrix stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">frustum</a>	Construct a perspective projection matrix based on frustum.
<a href="#">frustum</a>	Construct a perspective projection matrix based on frustum.
<a href="#">getCol</a>	Get the column of a 4x4 matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 4x4 matrix.
<a href="#">getCol1</a>	Get column 1 of a 4x4 matrix.
<a href="#">getCol2</a>	Get column 2 of a 4x4 matrix.
<a href="#">getCol3</a>	Get column 3 of a 4x4 matrix.
<a href="#">getElem</a>	Get the element of a 4x4 matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 4x4 matrix referred to by the specified index.
<a href="#">getTranslation</a>	Get the translation component of a 4x4 matrix.
<a href="#">getUpper3x3</a>	Get the upper-left 3x3 submatrix of a 4x4 matrix.
<a href="#">identity</a>	Construct an identity 4x4 matrix.
<a href="#">lookAt</a>	Construct viewing matrix based on eye position, position looked at, and up direction.
<a href="#">Matrix4</a>	Default constructor; does no initialization.
<a href="#">Matrix4</a>	Copy constructor.
<a href="#">Matrix4</a>	Construct a 4x4 matrix containing the specified columns.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a 3x4 transformation matrix.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a 3x3 matrix and a 3D vector.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a unit-length quaternion and a 3D vector.
<a href="#">Matrix4</a>	Set all elements of a 4x4 matrix to the same scalar value.
<a href="#">Matrix4</a>	Set all elements of a 4x4 matrix to the same scalar value (scalar data contained in vector data type).
<a href="#">operator*</a>	Multiply a 4x4 matrix by a scalar.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 4D vector.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3D vector.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3D point.
<a href="#">operator*</a>	Multiply two 4x4 matrices.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3x4 transformation matrix.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 4x4 matrix.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>operator*=</u></a>	Perform compound assignment and multiplication by a 3x4 transformation matrix.
<a href="#"><u>operator+</u></a>	Add two 4x4 matrices.
<a href="#"><u>operator+=</u></a>	Perform compound assignment and addition with a 4x4 matrix.
<a href="#"><u>operator-</u></a>	Subtract a 4x4 matrix from another 4x4 matrix.
<a href="#"><u>operator_</u></a>	Negate all elements of a 4x4 matrix.
<a href="#"><u>operator-=</u></a>	Perform compound assignment and subtraction by a 4x4 matrix.
<a href="#"><u>operator=</u></a>	Assign one 4x4 matrix to another.
<a href="#"><u>operator[]</u></a>	Subscripting operator to set or get a column.
<a href="#"><u>operator[]</u></a>	Subscripting operator to get a column.
<a href="#"><u>orthographic</u></a>	Construct an orthographic projection matrix.
<a href="#"><u>orthographic</u></a>	Construct an orthographic projection matrix.
<a href="#"><u>perspective</u></a>	Construct a perspective projection matrix.
<a href="#"><u>perspective</u></a>	Construct a perspective projection matrix.
<a href="#"><u>rotation</u></a>	Construct a 4x4 matrix to rotate around a unit-length 3D vector.
<a href="#"><u>rotation</u></a>	Construct a 4x4 matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).
<a href="#"><u>rotation</u></a>	Construct a rotation matrix from a unit-length quaternion.
<a href="#"><u>rotationX</u></a>	Construct a 4x4 matrix to rotate around the x axis.
<a href="#"><u>rotationX</u></a>	Construct a 4x4 matrix to rotate around the x axis (scalar data contained in vector data type).
<a href="#"><u>rotationY</u></a>	Construct a 4x4 matrix to rotate around the y axis.
<a href="#"><u>rotationY</u></a>	Construct a 4x4 matrix to rotate around the y axis (scalar data contained in vector data type).
<a href="#"><u>rotationZ</u></a>	Construct a 4x4 matrix to rotate around the z axis.
<a href="#"><u>rotationZ</u></a>	Construct a 4x4 matrix to rotate around the z axis (scalar data contained in vector data type).
<a href="#"><u>rotationZYX</u></a>	Construct a 4x4 matrix to rotate around the x, y, and z axes.
<a href="#"><u>scale</u></a>	Construct a 4x4 matrix to perform scaling.
<a href="#"><u>setCol</u></a>	Set the column of a 4x4 matrix referred to by the specified index.
<a href="#"><u>setCol0</u></a>	Set column 0 of a 4x4 matrix.
<a href="#"><u>setCol1</u></a>	Set column 1 of a 4x4 matrix.
<a href="#"><u>setCol2</u></a>	Set column 2 of a 4x4 matrix.
<a href="#"><u>setCol3</u></a>	Set column 3 of a 4x4 matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 4x4 matrix referred to by column and row indices.
<a href="#"><u>setElem</u></a>	Set the element of a 4x4 matrix referred to by column and row indices (scalar data contained in vector data type).
<a href="#"><u>setRow</u></a>	Set the row of a 4x4 matrix referred to by the specified index.
<a href="#"><u>setTranslation</u></a>	Set translation component.
<a href="#"><u>setUpper3x3</u></a>	Set the upper-left 3x3 submatrix.
<a href="#"><u>translation</u></a>	Construct a 4x4 matrix to perform translation.
<a href="#"><u>zero</u></a>	Construct a zero 4x4 matrix.

# Constructors and Destructors

## Matrix4

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix4

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        const Matrix4 &mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        Vector4 arg col0,
                        Vector4 arg col1,
                        Vector4 arg col2,
                        Vector4 arg col3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	4D vector.
<i>col1</i>	4D vector.
<i>col2</i>	4D vector.
<i>col3</i>	4D vector.

## Return Values

None

## Description

Construct a 4x4 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        Transform3 arg mat
                    );
                };
            }
        }
    }
}
```

## Arguments

*mat*      3x4 transformation matrix.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper 3x4 elements are equal to the 3x4 transformation matrix argument and whose bottom row is equal to (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a 3x3 matrix and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        Matrix3 arg mat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper 3x3 elements are equal to the 3x3 matrix argument, whose translation component is equal to the 3D vector argument, and whose bottom row is (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a unit-length quaternion and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Quat arg unitQuat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper-left 3x3 submatrix is a rotation matrix converted from the unit-length quaternion argument, whose translation component is equal to the 3D vector argument, and whose bottom row is (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Set all elements of a 4x4 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 4x4 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix4

Set all elements of a 4x4 matrix to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Matrix4 (
                        floatInVec arg scalar
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## Return Values

None

## Description

Construct a 4x4 matrix with all elements set to the scalar value argument.

# Operator Methods

## **operator\***

Multiply a 4x4 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 4x4 matrix and scalar

### **Description**

Multiply a 4x4 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Product of the specified 4x4 matrix and scalar

## **Description**

Multiply a 4x4 matrix by a scalar.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Vector4 arg
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Product of the specified 4x4 matrix and 4D vector

## Description

Multiply a 4x4 matrix by a 4D vector.

SCE CONFIDENTIAL

# **operator\***

---

Multiply a 4x4 matrix by a 3D vector.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Vector3 arg vec
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

---

vec                   3D vector.

## **Return Values**

---

Product of the specified 4x4 matrix and 3D vector

## **Description**

---

Multiply a 4x4 matrix by a 3D vector treated as if it were a 4D vector with the *w* element equal to 0.

SCE CONFIDENTIAL

# **operator\***

---

Multiply a 4x4 matrix by a 3D point.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Point3 arg pnt
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

---

*pnt*                    3D point.

## **Return Values**

---

Product of the specified 4x4 matrix and 3D point

## **Description**

---

Multiply a 4x4 matrix by a 3D point treated as if it were a 4D vector with the *w* element equal to 1.

SCE CONFIDENTIAL

# operator\*

Multiply two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        Matrix4 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Product of the specified 4x4 matrices

## Description

Multiply two 4x4 matrices.

SCE CONFIDENTIAL

# **operator\***

Multiply a 4x4 matrix by a 3x4 transformation matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        Transform3 arg tfrm
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*                   3x4 transformation matrix.

## **Return Values**

Product of the specified 4x4 matrix and 3x4 transformation matrix

## **Description**

Multiply a 4x4 matrix by a 3x4 transformation matrix treated as if it were a 4x4 matrix with the bottom row equal to (0,0,0,1).

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        floatInVec arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*= operator\*=**

Perform compound assignment and multiplication by a 4x4 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        Matrix4 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  4x4 matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a 4x4 matrix.

SCE CONFIDENTIAL

# **operator\*= operator\*=**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*      3x4 transformation matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

SCE CONFIDENTIAL

# operator+

Add two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator+
                        (Matrix4 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Sum of the specified 4x4 matrices

## Description

Add two 4x4 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator+=(
                        Matrix4 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Perform compound assignment and addition with a 4x4 matrix.

SCE CONFIDENTIAL

# **operator-**

Subtract a 4x4 matrix from another 4x4 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator-
                        (Matrix4 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  4x4 matrix.

## **Return Values**

Difference of the specified 4x4 matrices

## **Description**

Subtract a 4x4 matrix from another 4x4 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 4x4 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

4x4 matrix containing negated elements of the specified 4x4 matrix

## **Description**

Negate all elements of a 4x4 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator==(Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Perform compound assignment and subtraction by a 4x4 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 4x4 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &operator=(Matrix4 arg mat
                );
            }
        }
    }
}
```

## **Arguments**

*mat*                  4x4 matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Assign one 4x4 matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-3.

## **Return Values**

A reference to the indexed column

## **Description**

Subscripting operator invoked when applied to non-const [Matrix4](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Matrix4](#).

# Public Static Methods

## frustum

Construct a perspective projection matrix based on frustum.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 frustum(
                        float left,
                        float right,
                        float bottom,
                        float top,
                        float zNear,
                        float zFar
                    );
                };
            }
        }
    }
}
```

### Arguments

<i>left</i>	Scalar value.
<i>right</i>	Scalar value.
<i>bottom</i>	Scalar value.
<i>top</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

### Return Values

The constructed 4x4 matrix

### Description

Construct a perspective projection matrix based on frustum, equal to:

$2*n/(r-l)$	0	$(r+l)/(r-l)$	0
0	$2*n/(t-b)$	$(t+b)/(t-b)$	0
0	0	$-(f+n)/(f-n)$	$-2*f*n/(f-n)$
0	0	-1	0

*; l = left*  
*; r = right*  
*; b = bottom*  
*; t = top*  
*; n = zNear*  
*; f = zFar*

SCE CONFIDENTIAL

---

**Notes**

This function will assert if *zNear* is not less than *zFar*.

000004892117

SCE CONFIDENTIAL

# frustum

Construct a perspective projection matrix based on frustum.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 frustum(
                        floatInVec arg left,
                        floatInVec arg right,
                        floatInVec arg bottom,
                        floatInVec arg top,
                        floatInVec arg zNear,
                        floatInVec arg zFar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>left</i>	Scalar value (stored in vector data type).
<i>right</i>	Scalar value (stored in vector data type).
<i>bottom</i>	Scalar value (stored in vector data type).
<i>top</i>	Scalar value (stored in vector data type).
<i>zNear</i>	Scalar value (stored in vector data type).
<i>zFar</i>	Scalar value (stored in vector data type).

## Return Values

The constructed 4x4 matrix

## Description

Construct a perspective projection matrix based on frustum, equal to:

$2*n/(r-l)$	0	$(r+l)/(r-l)$	0
0	$2*n/(t-b)$	$(t+b)/(t-b)$	0
0	0	$-(f+n)/(f-n)$	$-2*f*n/(f-n)$
0	0	-1	0

*; l = left  
 ; r = right  
 ; b = bottom  
 ; t = top  
 ; n = zNear  
 ; f = zFar*

## Notes

This function will assert if *zNear* is not less than *zFar*.

SCE CONFIDENTIAL

# identity

Construct an identity 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 identity();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4x4 matrix

## Description

Construct an identity 4x4 matrix in which non-diagonal elements are zero and diagonal elements are 1.

SCE CONFIDENTIAL

# lookAt

Construct viewing matrix based on eye position, position looked at, and up direction.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 lookAt(
                        Point3_arg eyePos,
                        Point3_arg lookAtPos,
                        Vector3_arg upVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>eyePos</i>	3D point.
<i>lookAtPos</i>	3D point.
<i>upVec</i>	3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct the inverse of a coordinate frame that is centered at the eye position, with z axis directed away from *lookAtPos*, and y axis oriented to best match the up direction.

SCE CONFIDENTIAL

# orthographic

Construct an orthographic projection matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 orthographic(
                        float left,
                        float right,
                        float bottom,
                        float top,
                        float zNear,
                        float zFar
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>left</i>	Scalar value.
<i>right</i>	Scalar value.
<i>bottom</i>	Scalar value.
<i>top</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct an orthographic projection matrix, equal to:

$$\begin{array}{cccc} \frac{2}{(r-l)} & 0 & 0 & -\frac{(r+l)}{(r-l)} \\ 0 & \frac{2}{(t-b)} & 0 & -\frac{(t+b)}{(t-b)} \\ 0 & 0 & -\frac{2}{(f-n)} & -\frac{(f+n)}{(f-n)} \\ 0 & 0 & 0 & 1 \end{array}$$

```
; l = left
; r = right
; b = bottom
; t = top
; n = zNear
; f = zFar
```

## Notes

This function will assert if *zNear* is not less than *zFar*.

# orthographic

Construct an orthographic projection matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 orthographic(
                        floatInVec arg left,
                        floatInVec arg right,
                        floatInVec arg bottom,
                        floatInVec arg top,
                        floatInVec arg zNear,
                        floatInVec arg zFar
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>left</i>	Scalar value (stored in vector data type).
<i>right</i>	Scalar value (stored in vector data type).
<i>bottom</i>	Scalar value (stored in vector data type).
<i>top</i>	Scalar value (stored in vector data type).
<i>zNear</i>	Scalar value (stored in vector data type).
<i>zFar</i>	Scalar value (stored in vector data type).

## Return Values

The constructed 4x4 matrix

## Description

Construct an orthographic projection matrix, equal to:

$$\begin{matrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\ 0 & 0 & -2/(f-n) & -(f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{matrix}$$

```
; l = left
; r = right
; b = bottom
; t = top
; n = zNear
; f = zFar
```

SCE CONFIDENTIAL

---

**Notes**

This function will assert if *zNear* is not less than *zFar*.

000004892117

SCE CONFIDENTIAL

# **perspective**

Construct a perspective projection matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 perspective(
                        float fovyRadians,
                        float aspect,
                        float zNear,
                        float zFar
                    );
                };
            }
        }
    }
}
```

## **Arguments**

<i>fovyRadians</i>	Scalar value.
<i>aspect</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

## **Return Values**

The constructed 4x4 matrix

## **Description**

Construct a perspective projection matrix, equal to:

$\cot(y/2)/a$	0	0	0
0	$\cot(y/2)$	0	0
0	0	$(f+n)/(n-f)$	$2*f*n/(n-f)$
0	0	-1	0

*; y = fovyRadians  
; a = aspect  
; n = zNear  
; f = zFar*

## **Notes**

This function will assert if *zNear* is not less than *zFar*.

SCE CONFIDENTIAL

# **perspective**

Construct a perspective projection matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 perspective(
                        floatInVec arg fovyRadians,
                        floatInVec arg aspect,
                        floatInVec arg zNear,
                        floatInVec arg zFar
                    );
                };
            }
        }
    }
}
```

## **Arguments**

<i>fovyRadians</i>	Scalar value (stored in vector data type).
<i>aspect</i>	Scalar value (stored in vector data type).
<i>zNear</i>	Scalar value (stored in vector data type).
<i>zFar</i>	Scalar value (stored in vector data type).

## **Return Values**

The constructed 4x4 matrix

## **Description**

Construct a perspective projection matrix, equal to:

$$\begin{matrix} \cot(y/2)/a & 0 & 0 & 0 \\ 0 & \cot(y/2) & 0 & 0 \\ 0 & 0 & (f+n)/(n-f) & 2*f*n/(n-f) \\ 0 & 0 & -1 & 0 \end{matrix}$$

*; y = fovyRadians  
 ; a = aspect  
 ; n = zNear  
 ; f = zFar*

## **Notes**

This function will assert if *zNear* is not less than *zFar*.

SCE CONFIDENTIAL

# rotation

Construct a 4x4 matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotation(
                        float radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a 4x4 matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotation(
                        floatInVec arg radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
-----------------	---

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 4x4 matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationX(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationX

Construct a 4x4 matrix to rotate around the x axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationX(
                        floatInVec_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value (stored in vector data type).

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 4x4 matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationY(
                        float radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 4x4 matrix to rotate around the y axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationY(
                        floatInVec_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value (stored in vector data type).

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 4x4 matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationZ(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 4x4 matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationZ(
                        floatInVec_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value (stored in vector data type).

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 4x4 matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                }
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 4x4 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

# translation

Construct a 4x4 matrix to perform translation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 translation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec* 3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to perform translation, which is an identity matrix except for the translation component, with coordinates equal to those in *translateVec*.

SCE CONFIDENTIAL

## zero

Construct a zero 4x4 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 4x4 matrix

### Description

Construct a zero 4x4 matrix in which all elements are zero.

# Public Instance Methods

## getCol

Get the column of a 4x4 matrix referred to by the specified index.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

*col* Index, expected in the range 0-3.

### Return Values

The column referred to by the specified index

### Description

Get the column of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector4 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector4 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol2

Get column 2 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector4 getCol2() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 2

## Description

Get column 2 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol3

Get column 3 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector4 getCol3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 3

## Description

Get column 3 of a 4x4 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 4x4 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-3.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 4x4 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-3.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getTranslation

Get the translation component of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getTranslation()
                    const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Translation component

## Description

Get the translation component of a 4x4 matrix.

SCE CONFIDENTIAL

# getUpper3x3

Get the upper-left 3x3 submatrix of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Matrix3 getUpper3x3()
                    const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Upper-left 3x3 submatrix

## Description

Get the upper-left 3x3 submatrix of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol

Set the column of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol(
                        int col,
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the column of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setCol0(
                        Vector4 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 0 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setCol1(
                        Vector4 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 1 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setCol2(
                        Vector4 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 2 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol3

Set column 3 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setCol3(
                        Vector4 arg col3
                    );
                }
            }
        }
    }
}
```

## Arguments

*col3*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 3 of a 4x4 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 4x4 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setElem(
                        int col,
                        int row,
                        float val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-3.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the element of a 4x4 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setElem

Set the element of a 4x4 matrix referred to by column and row indices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setElem(
                        int col,
                        int row,
                        floatInVec arg val
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-3.
<i>val</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the element of a 4x4 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setRow(
                        int row,
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-3.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the row of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setTranslation

Set translation component.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setTranslation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec* 3D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the translation component of a 4x4 matrix equal to the specified 3D vector.

## Notes

This function does not change the bottom row elements.

SCE CONFIDENTIAL

# **setUpUpper3x3**

---

Set the upper-left 3x3 submatrix.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4 &setUpUpper3x3(
                        Matrix3 arg mat3
                    );
                };
            }
        }
    }
}
```

## **Arguments**

---

*mat3*      3x3 matrix.

## **Return Values**

---

A reference to the resulting 4x4 matrix

## **Description**

---

Set the upper-left 3x3 submatrix elements of a 4x4 matrix equal to the specified 3x3 matrix.

## **Notes**

---

This function does not change the bottom row elements.

**sce::Vectormath::Simd::Aos::Point3**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Point3

A 3D point in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Point3 {};
```

### Description

A class representing a 3D point stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a 3D point.
<a href="#">get128Ref</a>	Get vector float data reference from a 3D point.
<a href="#">getElem</a>	Get an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D point by index.
<a href="#">getExtVector</a>	Get clang extended vector float data from a 3D point.
<a href="#">getX</a>	Get the <i>x</i> element of a 3D point.
<a href="#">getXY</a>	Get the <i>x</i> and <i>y</i> elements of a 3D point.
<a href="#">getY</a>	Get the <i>y</i> element of a 3D point.
<a href="#">getZ</a>	Get the <i>z</i> element of a 3D point.
<a href="#">operator vec float3 ext</a>	Implicit cast to clang extended vector <code>vec float3 ext</code> .
<a href="#">operator+</a>	Add a 3D vector to a 3D point.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3D vector.
<a href="#">operator-</a>	Subtract a 3D point from another 3D point.
<a href="#">operator-</a>	Subtract a 3D vector from a 3D point.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3D vector.
<a href="#">operator=</a>	Assign one 3D point to another.
<a href="#">operator[]</a>	Subscripting operator to set or get an element.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">origin</a>	Construct a 3D point representing the origin.
<a href="#">Point3</a>	Default constructor; does no initialization.
<a href="#">Point3</a>	Copy constructor.
<a href="#">Point3</a>	Construct a 3D point from <i>x</i> , <i>y</i> , and <i>z</i> elements.
<a href="#">Point3</a>	Construct a 3D point from <i>x</i> , <i>y</i> , and <i>z</i> elements (scalar data contained in vector data type).
<a href="#">Point3</a>	Construct a 3D point from a 2D vector and a scalar.
<a href="#">Point3</a>	Construct a 3D point from a 2D vector and a scalar (scalar data contained in vector data type).
<a href="#">Point3</a>	Copy elements from a 3D vector into a 3D point.
<a href="#">Point3</a>	Set all elements of a 3D point to the same scalar value.
<a href="#">Point3</a>	Set all elements of a 3D point to the same scalar value (scalar data contained in vector data type).
<a href="#">Point3</a>	Set vector float data in a 3D point.
<a href="#">Point3</a>	Construct a 3D point from clang extended vector float data.
<a href="#">set128</a>	Set vector float data in a 3D point.
<a href="#">setElem</a>	Set an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D point by index.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>setElem</u></a>	Set an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D point by index (scalar data contained in vector data type).
<a href="#"><u>setExtVector</u></a>	Set clang extended vector float data in a 3D point.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 3D point.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 3D point (scalar data contained in vector data type).
<a href="#"><u>setXY</u></a>	Set the <i>x</i> and <i>y</i> elements of a 3D point.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 3D point.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 3D point (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 3D point.
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 3D point (scalar data contained in vector data type).

# Constructors and Destructors

## Point3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Point3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Point3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Point3(
                        const Point3 &pnt
                    );
                }
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from  $x$ ,  $y$ , and  $z$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        float x,
                        float y,
                        float z
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.

## Return Values

None

## Description

Construct a 3D point containing the specified  $x$ ,  $y$ , and  $z$  elements.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from  $x$ ,  $y$ , and  $z$  elements (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        floatInVec arg x,
                        floatInVec arg y,
                        floatInVec arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value (stored in vector data type).
$y$	Scalar value (stored in vector data type).
$z$	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D point containing the specified  $x$ ,  $y$ , and  $z$  elements.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from a 2D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        Vector2 arg xy,
                        float z
                    );
                };
            };
        };
    };
}
```

## Arguments

xy	2D vector.
z	Scalar value.

## Return Values

None

## Description

Construct a 3D point with the *x* and *y* elements of the specified 2D vector and with the *z* element set to the specified scalar.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from a 2D vector and a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        Vector2 arg xy,
                        floatInVec arg z
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D point with the *x* and *y* elements of the specified 2D vector and with the *z* element set to the specified scalar.

SCE CONFIDENTIAL

# Point3

Copy elements from a 3D vector into a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Construct a 3D point containing the  $x$ ,  $y$ , and  $z$  elements of the specified 3D vector.

SCE CONFIDENTIAL

# Point3

Set all elements of a 3D point to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 3D point with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Point3

Set all elements of a 3D point to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        floatInVec arg scalar
                    );
                };
            };
        };
    };
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D point with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Point3

Set vector float data in a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        vec_float4_arg vf4
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf4* Initial value.

## Return Values

None

## Description

Construct a 3D point whose internal vector float data is set to the vector float argument.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from clang extended vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        vec_float3_ext_arg vf3
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf3* Initial value.

## Return Values

None

## Description

Construct a 3D point whose internal vector float data is set to the clang extended vector float argument.

# Operator Methods

## operator vec\_float3\_ext

Implicit cast to clang extended vector `vec_float3_ext`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE operator vec_float3_ext() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to clang extended vector `vec_float3_ext`.

SCE CONFIDENTIAL

# operator+

Add a 3D vector to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Point3 operator+
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Sum of the specified 3D point and 3D vector

## Description

Add a 3D vector to a 3D point.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator+=(  
                        Vector3 arg vec  
                    );
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D point

## Description

Perform compound assignment and addition with a 3D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3D point from another 3D point.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-
                        (Point3 arg pnt
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

Difference of the specified 3D points

## **Description**

Subtract a 3D point from another 3D point.

SCE CONFIDENTIAL

# operator-

Subtract a 3D vector from a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Point3 operator-
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Difference of the specified 3D point and 3D vector

## Description

Subtract a 3D vector from a 3D point.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator==(Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D point

## Description

Perform compound assignment and subtraction by a 3D vector.

SCE CONFIDENTIAL

# **operator=**

Assign one 3D point to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator=(  
                        Point3 arg pnt  
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

A reference to the resulting 3D point

## **Description**

Assign one 3D point to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE VecIdx operator[](int idx)
                };
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

[VecIdx](#), which holds a reference to the selected element

## Description

Subscripting operator invoked when applied to non-const [Point3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const floatInVec operator[](
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Point3](#).

# Public Static Methods

## origin

Construct a 3D point representing the origin.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    static SCE_VECTORMATH_INLINE const Point3 origin();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3D point

### Description

Construct a 3D point equal to (0,0,0).

# Public Instance Methods

## get128

Get vector float data from a 3D point.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a 3D point.

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, or *z* element of a 3D point by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, or *z* element of a 3D point by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# getExtVector

Get clang extended vector float data from a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float3_ext getExtVector()
                    const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Clang extended vector float data

## Description

Get clang extended vector float data from a 3D point.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const floatInVec getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 3D point

## Description

Get the *x* element of a 3D point.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a 3D point's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const floatInVec getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 3D point

## Description

Get the *y* element of a 3D point.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE const floatInVec getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 3D point

## Description

Get the *z* element of a 3D point.

SCE CONFIDENTIAL

# set128

Set vector float data in a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE void set128(
                        vec_float4_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4* Vector float data.

## Return Values

None

## Description

Set vector float data in a 3D point.

# setElem

Set an *x*, *y*, or *z* element of a 3D point by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setElem(
                        int idx,
                        float value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set an *x*, *y*, or *z* element of a 3D point by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, or *z* element of a 3D point by index (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setElem(
                        int idx,
                        floatInVec arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D point

## Description

Set an *x*, *y*, or *z* element of a 3D point by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setExtVector

Set clang extended vector float data in a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE void setExtVector(
                        vec_float3_ext_arg vf3
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf3* Clang extended vector float data.

## Return Values

None

## Description

Set clang extended vector float data in a 3D point.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setX(
                        float x
                    );
                };
            };
        };
    };
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the *x* element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D point (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setX(
                        floatInVec arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D point

## Description

Set the *x* element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 3D point

## Description

Set a 3D point's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setY(
                        float y
                    );
                };
            };
        };
    };
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the  $y$  element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the *y* element of a 3D point (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setY(
                        floatInVec arg_y
                    );
                }
            }
        }
    }
}
```

## Arguments

*y* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D point

## Description

Set the *y* element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setZ(
                        float z
                    );
                };
            };
        };
    };
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the *z* element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D point (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setZ(
                        floatInVec arg z
                    );
                }
            }
        }
    }
}
```

## Arguments

*z* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D point

## Description

Set the *z* element of a 3D point to the specified scalar value.

# Protected Instance Methods

## get128Ref

Get vector float data reference from a 3D point.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 &get128Ref();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

A reference to the internal vector float data

### Description

Get internal vector float data reference from a 3D point.

**sce::Vectormath::Simd::Aos::Quat**

000004892117

# Summary

# sce::Vectormath::Simd::Aos::Quat

A quaternion in array-of-structures format.

## Definition

```
#include <vectormath.h>
class Quat {};
```

## Description

A class representing a quaternion stored in array-of-structures (AoS) format.

## Methods Summary

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>operator=</u></a>	Assign one quaternion to another.
<a href="#"><u>operator[]</u></a>	Subscripting operator to set or get an element.
<a href="#"><u>operator[]</u></a>	Subscripting operator to get an element.
<a href="#"><u>Quat</u></a>	Default constructor; does no initialization.
<a href="#"><u>Quat</u></a>	Copy constructor.
<a href="#"><u>Quat</u></a>	Construct a quaternion from $x$ , $y$ , $z$ , and $w$ elements.
<a href="#"><u>Quat</u></a>	Construct a quaternion from $x$ , $y$ , $z$ , and $w$ elements (scalar data contained in vector data type).
<a href="#"><u>Quat</u></a>	Construct a quaternion from a 3D vector and a scalar.
<a href="#"><u>Quat</u></a>	Construct a quaternion from a 3D vector and a scalar (scalar data contained in vector data type).
<a href="#"><u>Quat</u></a>	Copy elements from a 4D vector into a quaternion.
<a href="#"><u>Quat</u></a>	Convert a rotation matrix to a unit-length quaternion.
<a href="#"><u>Quat</u></a>	Set all elements of a quaternion to the same scalar value.
<a href="#"><u>Quat</u></a>	Set all elements of a quaternion to the same scalar value (scalar data contained in vector data type).
<a href="#"><u>Quat</u></a>	Set vector float data in a quaternion.
<a href="#"><u>Quat</u></a>	Construct a quaternion from clang extended vector float data.
<a href="#"><u>rotation</u></a>	Construct a quaternion from an Euler rotation.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate between two unit-length 3D vectors.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate around a unit-length 3D vector.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate around a unit-length 3D vector (scalar data contained in vector data type).
<a href="#"><u>rotationX</u></a>	Construct a quaternion to rotate around the $x$ axis.
<a href="#"><u>rotationX</u></a>	Construct a quaternion to rotate around the $x$ axis (scalar data contained in vector data type).
<a href="#"><u>rotationY</u></a>	Construct a quaternion to rotate around the $y$ axis.
<a href="#"><u>rotationY</u></a>	Construct a quaternion to rotate around the $y$ axis (scalar data contained in vector data type).
<a href="#"><u>rotationZ</u></a>	Construct a quaternion to rotate around the $z$ axis.
<a href="#"><u>rotationZ</u></a>	Construct a quaternion to rotate around the $z$ axis (scalar data contained in vector data type).
<a href="#"><u>set128</u></a>	Set vector float data in a quaternion.
<a href="#"><u>setElem</u></a>	Set an $x$ , $y$ , $z$ , or $w$ element of a quaternion by index.
<a href="#"><u>setElem</u></a>	Set an $x$ , $y$ , $z$ , or $w$ element of a quaternion by index (scalar data contained in vector data type).
<a href="#"><u>setExtVector</u></a>	Set clang extended vector float data in a quaternion.
<a href="#"><u>setW</u></a>	Set the $w$ element of a quaternion.
<a href="#"><u>setW</u></a>	Set the $w$ element of a quaternion (scalar data contained in vector data type).
<a href="#"><u>setX</u></a>	Set the $x$ element of a quaternion.
<a href="#"><u>setX</u></a>	Set the $x$ element of a quaternion (scalar data contained in vector data type).
<a href="#"><u>setXY</u></a>	Set the $x$ and $y$ elements of a quaternion.
<a href="#"><u>setXYZ</u></a>	Set the $x$ , $y$ , and $z$ elements of a quaternion.
<a href="#"><u>setY</u></a>	Set the $y$ element of a quaternion.
<a href="#"><u>setY</u></a>	Set the $y$ element of a quaternion (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the $z$ element of a quaternion.
<a href="#"><u>setZ</u></a>	Set the $z$ element of a quaternion (scalar data contained in vector data type).
<a href="#"><u>zero</u></a>	Construct a zero quaternion.

# Constructors and Destructors

## Quat

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE Quat();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Quat

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE Quat(
                        const Quat &quat
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      quaternion.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        float x,
                        float y,
                        float z,
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.
$w$	Scalar value.

## Return Values

None

## Description

Construct a quaternion containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from  $x$ ,  $y$ ,  $z$ , and  $w$  elements (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        floatInVec arg x,
                        floatInVec arg y,
                        floatInVec arg z,
                        floatInVec arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value (stored in vector data type).
$y$	Scalar value (stored in vector data type).
$z$	Scalar value (stored in vector data type).
$w$	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a quaternion containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from a 3D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        Vector3 arg xyz,
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a quaternion with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from a 3D vector and a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        Vector3 arg xyz,
                        floatInVec arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a quaternion with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Quat

Copy elements from a 4D vector into a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

None

## Description

Construct a quaternion containing the  $x$ ,  $y$ ,  $z$ , and  $w$  elements of the specified 4D vector.

SCE CONFIDENTIAL

# Quat

Convert a rotation matrix to a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        Matrix3 arg rotMat
                    );
                };
            }
        }
    }
}
```

## Arguments

*rotMat*      3x3 matrix, expected to be a rotation matrix.

## Return Values

None

## Description

Construct a unit-length quaternion representing the same transformation as a rotation matrix.

SCE CONFIDENTIAL

# Quat

Set all elements of a quaternion to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a quaternion with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Quat

Set all elements of a quaternion to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a quaternion with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Quat

Set vector float data in a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        vec_float4_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4*                   Scalar value.

## Return Values

None

## Description

Construct a quaternion whose internal vector float data is set to the vector float argument.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from clang extended vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        vec_float4_ext_arg vf4
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf4* Initial value.

## Return Values

None

## Description

Construct a quaternion whose internal vector float data is set to the clang extended vector float argument.

# Operator Methods

## **operator vec\_float4\_ext**

Implicit cast to clang extended vector `vec_float4_ext`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE operator vec_float4_ext() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to clang extended vector `vec_float4_ext`.

SCE CONFIDENTIAL

# **operator\***

Multiply two quaternions.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator*(
                        Quat arg quat
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*quat*      Quaternion.

## **Return Values**

Product of the specified quaternions

## **Description**

Multiply two quaternions.

SCE CONFIDENTIAL

# operator\*

Multiply a quaternion by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Product of the specified quaternion and scalar

## Description

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a quaternion by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Product of the specified quaternion and scalar

## **Description**

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# operator\*=

Perform compound assignment and multiplication by a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator*=(  

                        Quat arg quat  

                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and multiplication by a quaternion.

SCE CONFIDENTIAL

# operator\*=

Perform compound assignment and multiplication by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator*=(float scalar);
                };
            };
        };
    };
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator*=(floatInVec arg scalar)
                };
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting quaternion

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator+
                        (Quat arg quat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

Sum of the specified quaternions

## Description

Add two quaternions.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator+=(
                        Quat arg quat
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and addition with a quaternion.

SCE CONFIDENTIAL

# operator-

Subtract a quaternion from another quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator-(
                        Quat arg quat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

Difference of the specified quaternions

## Description

Subtract a quaternion from another quaternion.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a quaternion.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

Quaternion containing negated elements of the specified quaternion

## **Description**

Negate all elements of a quaternion.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator==(Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and subtraction by a quaternion.

SCE CONFIDENTIAL

# **operator/**

---

Divide a quaternion by a scalar.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator/(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

---

*scalar*      Scalar value.

## **Return Values**

---

Quotient of the specified quaternion and scalar

## **Description**

---

Divide a quaternion by a scalar.

SCE CONFIDENTIAL

# **operator/**

Divide a quaternion by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator/(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Quotient of the specified quaternion and scalar

## **Description**

Divide a quaternion by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator/=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator/=**

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator/=(
                        floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting quaternion

## **Description**

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one quaternion to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator=(Quat arg quat
                        );
                }
            }
        }
    }
}
```

## **Arguments**

*quat*                  Quaternion.

## **Return Values**

A reference to the resulting quaternion

## **Description**

Assign one quaternion to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE VecIdx operator[](int idx)
                };
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

[VecIdx](#), which holds a reference to the selected element

## Description

Subscripting operator invoked when applied to non-const [Quat](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec operator[](
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Quat](#).

# Public Static Methods

## axisAngle

Construct an axis-angle rotation from a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Vector4 axisAngle(
                        Quat_arg unitQuat
                    );
                }
            }
        }
    }
}
```

### Arguments

*unitQuat*      Unit rotation quaternion.

### Return Values

A 4D vector containing a 3D xyz vector, representing the unit-length axis, and a w element that specifies the rotation angle in radians.

### Description

Construct an axis-angle rotation from a quaternion.

SCE CONFIDENTIAL

# euler

Construct an Euler rotation from a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Vector3 euler(
                        Quat arg unitQuat,
                        RotationOrder order
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Unit rotation quaternion.
<i>order</i>	Euler rotation order.

## Return Values

A 3D vector containing values representing Euler angle rotations with an element ordering specified by the value *order*

## Description

Construct an Euler rotation from a quaternion.

## Notes

If your quaternion has been formed from Euler angles that suffer from pitch / pole singularities, then you will not get the same angles back that you put in.

SCE CONFIDENTIAL

# identity

Construct an identity quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat identity();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed quaternion

## Description

Construct an identity quaternion equal to (0,0,0,1).

SCE CONFIDENTIAL

# rotation

Construct a quaternion from an Euler rotation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        Vector3 arg radians,
                        RotationOrder order
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	3D vector of the Euler X, Y, and Z rotations in radians.
<i>order</i>	Euler rotation order.

## Return Values

A quaternion that represents the compounded rotation specified by the elements of *radians*

## Description

Construct a quaternion from an Euler rotation.

SCE CONFIDENTIAL

# rotation

Construct a quaternion to rotate between two unit-length 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        Vector3 arg unitVec0,
                        Vector3 arg unitVec1
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>unitVec0</i>	3D vector, expected to be unit-length.
<i>unitVec1</i>	3D vector, expected to be unit-length.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate between two unit-length 3D vectors.

## Notes

The result is unpredictable if *unitVec0* and *unitVec1* point in opposite directions.

SCE CONFIDENTIAL

# rotation

Construct a quaternion to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        float radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a quaternion to rotate around a unit-length 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        floatInVec arg radians,
                        Vector3 arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotationX

Construct a quaternion to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationX(
                        float radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationX

Construct a quaternion to rotate around the x axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationX(
                        floatInVec arg radians
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a quaternion to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationY(
                        float radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a quaternion to rotate around the y axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationY(
                        floatInVec arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a quaternion to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationZ(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a quaternion to rotate around the z axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationZ(
                        floatInVec arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

---

## zero

---

---

Construct a zero quaternion.

### Definition

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat zero();
                }
            }
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed quaternion

### Description

---

Construct a zero quaternion equal to (0,0,0,0).

# Public Instance Methods

## get128

Get vector float data from a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a quaternion.

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, *z*, or *w* element of a quaternion by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, *z*, or *w* element of a quaternion by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# getExtVector

Get clang extended vector float data from a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4_ext getExtVector()
                    const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Clang extended vector float data

## Description

Get clang extended vector float data from a quaternion.

SCE CONFIDENTIAL

# getW

Get the *w* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec getW() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*w* element of a quaternion

## Description

Get the *w* element of a quaternion.

SCE CONFIDENTIAL

# getX

Get the *x* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a quaternion

## Description

Get the *x* element of a quaternion.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a quaternion's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getXYZ

Get the *x*, *y*, and *z* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getXYZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

3D vector containing *x*, *y*, and *z* elements

## Description

Extract a quaternion's *x*, *y*, and *z* elements into a 3D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a quaternion

## Description

Get the *y* element of a quaternion.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE const floatInVec getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a quaternion

## Description

Get the *z* element of a quaternion.

SCE CONFIDENTIAL

# set128

Set vector float data in a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE void set128(
                        vec_float4_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4* Vector float data.

## Return Values

None

## Description

Set vector float data in a quaternion.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a quaternion by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setElem(
                        int idx,
                        float value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set an *x*, *y*, *z*, or *w* element of a quaternion by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a quaternion by index (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setElem(
                        int idx,
                        floatInVec arg value
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting quaternion

## Description

Set an *x*, *y*, *z*, or *w* element of a quaternion by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setExtVector

Set clang extended vector float data in a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE void setExtVector(
                        vec_float4_ext_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4* Clang extended vector float data.

## Return Values

None

## Description

Set clang extended vector float data in a quaternion.

SCE CONFIDENTIAL

# setW

Set the *w* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setW(
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

*w*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *w* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setW

Set the *w* element of a quaternion (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setW(
                        floatInVec arg_w
                    );
                }
            }
        }
    }
}
```

## Arguments

*w* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting quaternion

## Description

Set the *w* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setX(
                        float x
                    );
                };
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *x* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a quaternion (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setX(
                        floatInVec arg_x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting quaternion

## Description

Set the *x* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting quaternion

## Description

Set a quaternion's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* and *w* element.

SCE CONFIDENTIAL

# setXYZ

Set the *x*, *y*, and *z* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setXYZ(
                        Vector3 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                   3D vector.

## Return Values

A reference to the resulting quaternion

## Description

Set a quaternion's *x*, *y*, and *z* elements to those of the specified 3D vector.

## Notes

This function does not change the *w* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setY(
                        float y
                    );
                };
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the  $y$  element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the *y* element of a quaternion (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setY(
                        floatInVec arg_y
                    );
                }
            }
        }
    }
}
```

## Arguments

*y* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting quaternion

## Description

Set the *y* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setZ(
                        float z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *z* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a quaternion (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setZ(
                        floatInVec arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting quaternion

## Description

Set the *z* element of a quaternion to the specified scalar value.

# Protected Instance Methods

## get128Ref

Get vector float data reference from a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 &get128Ref();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

A reference to the internal vector float data

### Description

Get internal vector float data reference from a quaternion.

**sce::Vectormath::  
Simd::Aos::Transform3**

# Summary

## sce::Vectormath::Simd::Aos::Transform3

A 3x4 transformation matrix in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Transform3 {};
```

### Description

A class representing a 3x4 transformation matrix stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">getCol</a>	Get the column of a 3x4 transformation matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 3x4 transformation matrix.
<a href="#">getCol1</a>	Get column 1 of a 3x4 transformation matrix.
<a href="#">getCol2</a>	Get column 2 of a 3x4 transformation matrix.
<a href="#">getCol3</a>	Get column 3 of a 3x4 transformation matrix.
<a href="#">getElem</a>	Get the element of a 3x4 transformation matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 3x4 transformation matrix referred to by the specified index.
<a href="#">getTranslation</a>	Get the translation component of a 3x4 transformation matrix.
<a href="#">getUpper3x3</a>	Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.
<a href="#">identity</a>	Construct an identity 3x4 transformation matrix.
<a href="#">operator*</a>	Multiply a 3x4 transformation matrix by a 3D vector.
<a href="#">operator*</a>	Multiply a 3x4 transformation matrix by a 3D point.
<a href="#">operator*</a>	Multiply two 3x4 transformation matrices.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 3x4 transformation matrix.
<a href="#">operator=</a>	Assign one 3x4 transformation matrix to another.
<a href="#">operator[]</a>	Subscripting operator to set or get a column.
<a href="#">operator[]</a>	Subscripting operator to get a column.
<a href="#">rotation</a>	Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector.
<a href="#">rotation</a>	Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).
<a href="#">rotation</a>	Construct a rotation matrix from a unit-length quaternion.
<a href="#">rotationX</a>	Construct a 3x4 transformation matrix to rotate around the x axis.
<a href="#">rotationX</a>	Construct a 3x4 transformation matrix to rotate around the x axis (scalar data contained in vector data type).
<a href="#">rotationY</a>	Construct a 3x4 transformation matrix to rotate around the y axis.
<a href="#">rotationY</a>	Construct a 3x4 transformation matrix to rotate around the y axis (scalar data contained in vector data type).
<a href="#">rotationZ</a>	Construct a 3x4 transformation matrix to rotate around the z axis.
<a href="#">rotationZ</a>	Construct a 3x4 transformation matrix to rotate around the z axis (scalar data contained in vector data type).
<a href="#">rotationZYX</a>	Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>scale</u></a>	Construct a 3x4 transformation matrix to perform scaling.
<a href="#"><u>setCol</u></a>	Set the column of a 3x4 transformation matrix referred to by the specified index.
<a href="#"><u>setCol0</u></a>	Set column 0 of a 3x4 transformation matrix.
<a href="#"><u>setCol1</u></a>	Set column 1 of a 3x4 transformation matrix.
<a href="#"><u>setCol2</u></a>	Set column 2 of a 3x4 transformation matrix.
<a href="#"><u>setCol3</u></a>	Set column 3 of a 3x4 transformation matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 3x4 transformation matrix referred to by column and row indices.
<a href="#"><u>setElem</u></a>	Set the element of a 3x4 transformation matrix referred to by column and row indices (scalar data contained in vector data type).
<a href="#"><u>setRow</u></a>	Set the row of a 3x4 transformation matrix referred to by the specified index.
<a href="#"><u>setTranslation</u></a>	Set translation component.
<a href="#"><u>setUpper3x3</u></a>	Set the upper-left 3x3 submatrix.
<a href="#"><u>Transform3</u></a>	Default constructor; does no initialization.
<a href="#"><u>Transform3</u></a>	Copy constructor.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix containing the specified columns.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix from a 3x3 matrix and a 3D vector.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix from a unit-length quaternion and a 3D vector.
<a href="#"><u>Transform3</u></a>	Set all elements of a 3x4 transformation matrix to the same scalar value.
<a href="#"><u>Transform3</u></a>	Set all elements of a 3x4 transformation matrix to the same scalar value (scalar data contained in vector data type).
<a href="#"><u>translation</u></a>	Construct a 3x4 transformation matrix to perform translation.

# Constructors and Destructors

## Transform3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Transform3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        const Transform3 &tfrm
                    );
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        Vector3 arg col0,
                        Vector3 arg col1,
                        Vector3 arg col2,
                        Vector3 arg col3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	3D vector.
<i>col1</i>	3D vector.
<i>col2</i>	3D vector.
<i>col3</i>	3D vector.

## Return Values

None

## Description

Construct a 3x4 transformation matrix containing the specified columns.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix from a 3x3 matrix and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        Matrix3 arg tfrm,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x3 matrix.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 3x4 transformation matrix whose upper 3x3 elements are equal to the 3x3 matrix argument and whose translation component is equal to the 3D vector argument.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix from a unit-length quaternion and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Quat arg unitQuat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 3x4 transformation matrix whose upper-left 3x3 submatrix is a rotation matrix converted from the unit-length quaternion argument and whose translation component is equal to the 3D vector argument.

SCE CONFIDENTIAL

# Transform3

Set all elements of a 3x4 transformation matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 3x4 transformation matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Transform3

Set all elements of a 3x4 transformation matrix to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    explicit SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3x4 transformation matrix with all elements set to the scalar value argument.

# Operator Methods

## **operator\***

Multiply a 3x4 transformation matrix by a 3D vector.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        Vector3 arg
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*vec*                    3D vector.

### **Return Values**

Product of the specified 3x4 transformation matrix and 3D vector

### **Description**

Applies the 3x3 upper-left submatrix (but not the translation component) of a 3x4 transformation matrix to a 3D vector.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x4 transformation matrix by a 3D point.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Point3 operator*(
                        Point3 arg pnt
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

Product of the specified 3x4 transformation matrix and 3D point

## **Description**

Applies the 3x3 upper-left submatrix and the translation component of a 3x4 transformation matrix to a 3D point.

SCE CONFIDENTIAL

# operator\*

Multiply two 3x4 transformation matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Transform3 operator*(
                        Transform3 arg tfrm
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

Product of the specified 3x4 transformation matrices

## Description

Multiply two 3x4 transformation matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &operator*=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*      3x4 transformation matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 3x4 transformation matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &operator=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*      3x4 transformation matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Assign one 3x4 transformation matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-3.

## **Return Values**

A reference to the indexed column

## **Description**

Subscripting operator invoked when applied to non-const [Transform3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Transform3](#).

# Public Static Methods

## identity

Construct an identity 3x4 transformation matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x4 transformation matrix

### Description

Construct an identity 3x4 transformation matrix in which non-diagonal elements are zero and diagonal elements are 1.

# rotation

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotation(
                        float radians,
                        Vector3 arg unitVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotation(
                        floatInVec arg radians,
                        Vector3 arg unitVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 3x4 transformation matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationX(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the x axis by the specified radians angle.

# rotationX

Construct a 3x4 transformation matrix to rotate around the x axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationX(
                        floatInVec arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 3x4 transformation matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationY(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
----------------	---------------

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the y axis by the specified radians angle.

# rotationY

Construct a 3x4 transformation matrix to rotate around the y axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationY(
                        floatInVec arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 3x4 transformation matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationZ(
                        float radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
----------------	---------------

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the z axis by the specified radians angle.

# rotationZ

Construct a 3x4 transformation matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationZ(
                        floatInVec arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value (stored in vector data type).
----------------	--

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                }
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 3x4 transformation matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

scaleVec      3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of scaleVec.

SCE CONFIDENTIAL

# translation

Construct a 3x4 transformation matrix to perform translation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 translation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec* 3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to perform translation, which is an identity matrix except for the translation component, with coordinates equal to those in *translateVec*.

# Public Instance Methods

## getCol

Get the column of a 3x4 transformation matrix referred to by the specified index.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

*col* Index, expected in the range 0-3.

### Return Values

The column referred to by the specified index

### Description

Get the column of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

## getCol2

Get column 2 of a 3x4 transformation matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol2() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Column 2

### Description

Get column 2 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getCol3

Get column 3 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getCol3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 3

## Description

Get column 3 of a 3x4 transformation matrix.

# getElem

Get the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-2.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 3x4 transformation matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector4 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-2.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# getTranslation

Get the translation component of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getTranslation()
                    const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Translation component

## Description

Get the translation component of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getUpper3x3

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Matrix3 getUpper3x3()
                    const;
                };
            };
        };
    };
}
```

## Arguments

None

## Return Values

Upper-left 3x3 submatrix

## Description

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol

Set the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol(
                        int col,
                        Vector3 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the column of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setCol0(
                        Vector3 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 0 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setCol1(
                        Vector3 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 1 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setCol2(
                        Vector3 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 2 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol3

Set column 3 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setCol3(
                        Vector3 arg col3
                    );
                }
            }
        }
    }
}
```

## Arguments

*col3*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 3 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setElem(
                        int col,
                        int row,
                        float val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the element of a 3x4 transformation matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setElem

Set the element of a 3x4 transformation matrix referred to by column and row indices (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setElem(
                        int col,
                        int row,
                        floatInVec arg val
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the element of a 3x4 transformation matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setRow(
                        int row,
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-2.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the row of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# setTranslation

Set translation component.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setTranslation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec*    3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the translation component of a 3x4 transformation matrix equal to the specified 3D vector.

SCE CONFIDENTIAL

# **setUpUpper3x3**

Set the upper-left 3x3 submatrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3 &setUpUpper3x3(
                        Matrix3 arg mat3
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat3*      3x3 matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Set the upper-left 3x3 submatrix elements of a 3x4 transformation matrix equal to the specified 3x3 matrix.

**sce::Vectormath::Simd::Aos::VecIdx**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::VecIdx

A reference to an element in a SIMD `vec_float4` vector.

### Definition

```
#include <vectormath.h>
class VecIdx {};
```

### Description

A class representing a reference to an element in a SIMD `vec_float4` vector. Used in setting elements of [Vector3](#), [Vector4](#), [Point3](#), or [Quat](#) with the subscripting operator.

### Methods Summary

Methods	Description
<a href="#">getAsFloat</a>	Explicit cast to float.
<a href="#">operator float</a>	Implicit cast to float.
<a href="#">operator floatInVec</a>	Implicit cast to <code>floatInVec</code> .
<a href="#">operator*=</a>	Multiplication operator.
<a href="#">operator*=</a>	Multiplication operator.
<a href="#">operator+=</a>	Addition operator.
<a href="#">operator+=</a>	Addition operator.
<a href="#">operator-=</a>	Subtraction operator.
<a href="#">operator-=</a>	Subtraction operator.
<a href="#">operator/=</a>	Division operator.
<a href="#">operator/=</a>	Division operator.
<a href="#">operator=</a>	Assignment operator.
<a href="#">operator=</a>	Assignment operator.
<a href="#">operator=</a>	Assignment operator.
<a href="#">VecIdx</a>	Construct a reference to the vector <code>vec</code> element index <code>idx</code> .
<a href="#">VecIdx</a>	Copy constructor; copies the vector element reference.

# Constructors and Destructors

## VecIdx

Construct a reference to the vector `vec` element index `idx`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE VecIdx(
                        vec_float4 &vec,
                        int idx
                    );
                }
            }
        }
    }
}
```

### Arguments

<code>vec</code>	Vector reference.
<code>idx</code>	Vector element index.

### Return Values

None

### Description

Construct a reference to the vector `vec` element index `idx`.

SCE CONFIDENTIAL

# VecIdx

Copy constructor; copies the vector element reference.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE VecIdx(
                        const VecIdx &vidx
                    );
                };
            }
        }
    }
}
```

## Arguments

*vidx* Vector element reference.

## Return Values

None

## Description

Copy constructor; copies the vector element reference.

# Operator Methods

## **operator float**

Implicit cast to float.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE operator float() const;
                }
            }
        }
    }
}
```

### **Arguments**

None

### **Return Values**

None

### **Description**

Implicit cast to float.

### **Notes**

This function is only defined when `_SCE_VECTORMATH_NO_SCALAR_CAST` is undefined. (There is the option for the purpose of finding LHS by implicit cast.)

SCE CONFIDENTIAL

# operator floatInVec

Implicit cast to [floatInVec](#).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE operator floatInVec() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

[floatInVec](#) value

## Description

Implicit cast to [floatInVec](#).

SCE CONFIDENTIAL

# **operator\*=**

---

Multiplication operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator*=(float scalar
                );
            }
        }
    }
}
```

## **Arguments**

---

*scalar*      Scalar value.

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Multiplication operator; multiples the reference vector element.

SCE CONFIDENTIAL

# **operator\*=**

---

Multiplication operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator*=(floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	Scalar value (contained in a vector register).
---------------	--

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Multiplication operator; multiples the reference vector element.

# **operator+=**

---

Addition operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator+=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

---

*scalar*      Scalar value.

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Addition operator; adds to the reference vector element.

SCE CONFIDENTIAL

# **operator+=**

---

Addition operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator+=(floatInVec arg scalar
                        );
                }
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	Scalar value (contained in a vector register).
---------------	--

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Addition operator; adds to the reference vector element.

SCE CONFIDENTIAL

# operator-=

Subtraction operator.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator-=(float scalar);
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting value.

## Description

Subtraction operator; subtracts from the reference vector element.

SCE CONFIDENTIAL

# **operator-=**

---

Subtraction operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator-=
                        (floatInVec arg scalar
                     );
                }
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	Scalar value (contained in a vector register).
---------------	--

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Subtraction operator; subtracts from the reference vector element.

SCE CONFIDENTIAL

# **operator/=**

---

Division operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator/=(float scalar)
                };
            }
        }
    }
}
```

## **Arguments**

---

*scalar*      Scalar value.

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Division operator; divides the reference vector element.

SCE CONFIDENTIAL

# **operator/=**

---

Division operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator/=(floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	Scalar value (contained in a vector register).
---------------	--

## **Return Values**

---

The resulting value (contained in a vector register).

## **Description**

---

Division operator; divides the reference vector element.

SCE CONFIDENTIAL

# **operator=**

Assignment operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE float operator=(float scalar)
                };
            }
        }
    }
}
```

## **Arguments**

*scalar*      The scalar value to set the vector element to.

## **Return Values**

The input scalar value.

## **Description**

Assignment operator; sets the vector element value.

SCE CONFIDENTIAL

# **operator=**

---

Assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator=(floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

---

<i>scalar</i>	The scalar value (contained in a vector register) to set the vector element to.
---------------	---

## **Return Values**

---

The input scalar value (contained in a vector register).

## **Description**

---

Assignment operator; sets the vector element value.

SCE CONFIDENTIAL

# **operator=**

---

Assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE floatInVec operator=(const VecIdx &scalar);
                };
            }
        }
    }
}
```

## **Arguments**

---

*scalar*      The scalar value (contained in a vector reference) to set the vector element to.

## **Return Values**

---

The input scalar value (contained in a vector reference).

## **Description**

---

Assignment operator; sets the vector element value.

# Public Instance Methods

## getAsFloat

Explicit cast to float.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class VecIdx {
                    SCE_VECTORMATH_INLINE float getAsFloat() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Float value

### Description

Explicit cast to float.

**sce::Vectormath::Simd::Aos::Vector2**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Vector2

A 2D vector in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Vector2 {};
```

### Description

A class representing a 2D vector stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get64</a>	Get vector float data from a 2D vector.
<a href="#">get64Ref</a>	Get vector float data reference from a 2D vector.
<a href="#">getElem</a>	Get an <i>x</i> or <i>y</i> element of a 2D vector by index.
<a href="#">getExtVector</a>	Get clang extended vector float data from a 2D vector.
<a href="#">getFastVectorType</a>	Get fast vector float data from a 2D vector.
<a href="#">getX</a>	Get the <i>x</i> element of a 2D vector.
<a href="#">getY</a>	Get the <i>y</i> element of a 2D vector.
<a href="#">operator vec_float2_ext</a>	Implicit cast to clang extended vector <code>vec_float2_ext</code> .
<a href="#">operator*</a>	Multiply a 2D vector by a scalar.
<a href="#">operator*</a>	Multiply a 2D vector by a scalar (scalar data contained in vector data type).
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#">operator+</a>	Add two 2D vectors.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 2D vector.
<a href="#">operator-</a>	Subtract a 2D vector from another 2D vector.
<a href="#">operator-</a>	Negate all elements of a 2D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction with a 2D vector.
<a href="#">operator/</a>	Divide a 3D vector by a scalar.
<a href="#">operator/</a>	Divide a 2D vector by a scalar (scalar data contained in vector data type).
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar.
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar (scalar data contained in vector data type).
<a href="#">operator=</a>	Assign one 2D vector to another.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">set64</a>	Set vector float data in a 2D vector.
<a href="#">setElem</a>	Set an <i>x</i> or <i>y</i> element of a 2D vector by index.
<a href="#">setElem</a>	Set an <i>x</i> or <i>y</i> element of a 2D vector by index (scalar data contained in vector data type).
<a href="#">setExtVector</a>	Set clang extended vector float data in a 2D vector.
<a href="#">setFastVectorType</a>	Set vector float data in a 2D vector.
<a href="#">setX</a>	Set the <i>x</i> element of a 2D vector.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 2D vector (scalar data contained in vector data type).
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 2D vector.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 2D vector (scalar data contained in vector data type).
<a href="#"><u>Vector2</u></a>	Default constructor; does no initialization.
<a href="#"><u>Vector2</u></a>	Copy constructor.
<a href="#"><u>Vector2</u></a>	Construct a 2D vector from <i>x</i> and <i>y</i> elements.
<a href="#"><u>Vector2</u></a>	Construct a 2D vector from <i>x</i> and <i>y</i> elements (scalar data contained in vector data type).
<a href="#"><u>Vector2</u></a>	Set all elements of a 2D vector to the same scalar value.
<a href="#"><u>Vector2</u></a>	Set all elements of a 2D vector to the same scalar value (scalar data contained in vector data type).
<a href="#"><u>Vector2</u></a>	Set vector float data in a 2D vector.
<a href="#"><u>Vector2</u></a>	Construct a 2D vector from clang extended vector float data.
<a href="#"><u>xAxis</u></a>	Construct x axis.
<a href="#"><u>yAxis</u></a>	Construct y axis.
<a href="#"><u>zero</u></a>	Construct zero.

# Constructors and Destructors

## Vector2

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector2();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector2

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector2(
                        const Vector2 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector2

Construct a 2D vector from *x* and *y* elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2(
                        float x,
                        float y
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value.
<i>y</i>	Scalar value.

## Return Values

None

## Description

Construct a 2D vector containing the specified *x* and *y* elements.

SCE CONFIDENTIAL

# Vector2

Construct a 2D vector from *x* and *y* elements (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2(
                        floatInVec arg x,
                        floatInVec arg y
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (stored in vector data type).
<i>y</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 2D vector containing the specified *x* and *y* elements.

SCE CONFIDENTIAL

# Vector2

Set all elements of a 2D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    explicit SCE_VECTORMATH_INLINE Vector2(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 2D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector2

Set all elements of a 2D vector to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    explicit SCE_VECTORMATH_INLINE Vector2(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## Return Values

None

## Description

Construct a 2D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector2

Set vector float data in a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    explicit SCE_VECTORMATH_INLINE Vector2(
                        vec_float2_arg vf2
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf2* Initial value.

## Return Values

None

## Description

Construct a 2D vector whose internal vector float data is set to the vector float argument.

SCE CONFIDENTIAL

# Vector2

Construct a 2D vector from clang extended vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    explicit SCE_VECTORMATH_INLINE Vector2(
                        vec_float2_ext_arg vf2
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf2* Initial value.

## Return Values

None

## Description

Construct a 2D vector whose internal vector float data is set to the clang extended vector float argument.

# Operator Methods

## operator vec\_float2\_ext

Implicit cast to clang extended vector `vec_float2_ext`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE operator vec_float2_ext() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to clang extended vector `vec_float2_ext`.

SCE CONFIDENTIAL

# operator\*

Multiply a 2D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Product of the specified 2D vector and scalar

## Description

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# operator\*

Multiply a 2D vector by a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

Product of the specified 2D vector and scalar

## Description

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator*=(floatInVec arg scalar
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator+
                        (Vector2 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

Sum of the specified 2D vectors

## Description

Add two 2D vectors.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator+=(
                        Vector2 arg
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

A reference to the resulting 2D vector

## Description

Perform compound assignment and addition with a 2D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 2D vector from another 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator-
                        (Vector2 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

vec                  2D vector.

## **Return Values**

Difference of the specified 2D vectors

## **Description**

Subtract a 2D vector from another 2D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

2D vector containing negated elements of the specified 2D vector

## **Description**

Negate all elements of a 2D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction with a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator==(Vector2 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

A reference to the resulting 2D vector

## Description

Perform compound assignment and subtraction with a 2D vector.

SCE CONFIDENTIAL

# **operator/**

Divide a 3D vector by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator/(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

Quotient of the specified 3D vector and scalar

## **Description**

Divide a 3D vector by a scalar.

SCE CONFIDENTIAL

# **operator/**

Divide a 2D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator/(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Quotient of the specified 2D vector and scalar

## **Description**

Divide a 2D vector by a scalar.

SCE CONFIDENTIAL

# **operator/=**

Perform compound assignment and division by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator/=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator/=**

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator/=(
                        floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 2D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator=(Vector2 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Assign one 2D vector to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const floatInVec operator[](
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-1.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector2](#).

# Public Static Methods

## xAxis

Construct x axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 xAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2D vector

### Description

Construct a 2D vector equal to (1,0).

SCE CONFIDENTIAL

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 yAxis();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 2D vector

## Description

Construct a 2D vector equal to (0,1).

SCE CONFIDENTIAL

---

## zero

---

---

Construct zero.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2D vector

### Description

Construct a 2D vector equal to (0,0).

# Public Instance Methods

## get64

Get vector float data from a 2D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float2 get64() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a 2D vector.

SCE CONFIDENTIAL

# getElem

Get an *x* or *y* element of a 2D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-1.

## Return Values

Element selected by the specified index

## Description

Get an *x* or *y* element of a 2D vector by specifying an index of 0 or 1, respectively.

SCE CONFIDENTIAL

# getExtVector

Get clang extended vector float data from a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float2_ext getExtVector() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Clang extended vector float data

## Description

Get clang extended vector float data from a 2D vector.

SCE CONFIDENTIAL

# getFastVectorType

Get fast vector float data from a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE fast_float_vector_type
                    getFastVectorType() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Fast vector float data

## Description

Get internal vector float data from a 2D vector as fast vector float data.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const floatInVec getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 2D vector

## Description

Get the *x* element of a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const floatInVec getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 2D vector

## Description

Get the *y* element of a 2D vector.

SCE CONFIDENTIAL

# set64

Set vector float data in a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE void set64(
                        vec_float2_arg vf2
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf2*                  Vector float data.

## Return Values

None

## Description

Set vector float data in a 2D vector.

SCE CONFIDENTIAL

# setElem

Set an *x* or *y* element of a 2D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setElem(
                        int idx,
                        float value
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-1.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set an *x* or *y* element of a 2D vector by specifying an index of 0 or 1, respectively.

SCE CONFIDENTIAL

# setElem

Set an *x* or *y* element of a 2D vector by index (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setElem(
                        int idx,
                        floatInVec_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-1.
<i>value</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 2D vector

## Description

Set an *x* or *y* element of a 2D vector by specifying an index of 0 or 1, respectively.

SCE CONFIDENTIAL

# setExtVector

Set clang extended vector float data in a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE void setExtVector(
                        vec_float2_ext_arg vf2
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf2* Clang extended vector float data.

## Return Values

None

## Description

Set clang extended vector float data in a 2D vector.

SCE CONFIDENTIAL

# **setFastVectorType**

Set vector float data in a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE void setFastVectorType(
                        fast_float_vector_arg vf2
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*vf2* Fast vector float data.

## **Return Values**

None

## **Description**

Set vector float data in a 2D vector from fast vector float data.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setX(
                        float x
                    );
                };
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set the *x* element of a 2D vector to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 2D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setX(
                        floatInVec arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 2D vector

## Description

Set the *x* element of a 2D vector to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setY(
                        float y
                    );
                };
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set the  $y$  element of a 2D vector to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 2D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setY(
                        floatInVec arg_y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 2D vector

## Description

Set the  $y$  element of a 2D vector to the specified scalar value.

# Protected Instance Methods

## get64Ref

Get vector float data reference from a 2D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float2 &get64Ref();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

A reference to the internal vector float data

### Description

Get internal vector float data reference from a 2D vector.

**sce::Vectormath::Simd::Aos::Vector3**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Vector3

A 3D vector in array-of-structures format.

## Definition

```
#include <vectormath.h>
class Vector3 {};
```

## Description

A class representing a 3D vector stored in array-of-structures (AoS) format.

## Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a 3D vector.
<a href="#">get128Ref</a>	Get vector float data reference from a 3D vector.
<a href="#">getElem</a>	Get an $x$ , $y$ , or $z$ element of a 3D vector by index.
<a href="#">getExtVector</a>	Get clang extended vector float data from a 3D vector.
<a href="#">getX</a>	Get the $x$ element of a 3D vector.
<a href="#">getXY</a>	Get the $x$ and $y$ elements of a 3D vector.
<a href="#">getY</a>	Get the $y$ element of a 3D vector.
<a href="#">getZ</a>	Get the $z$ element of a 3D vector.
<a href="#">operator vec float3 ext</a>	Implicit cast to clang extended vector <code>vec float3 ext</code> .
<a href="#">operator*</a>	Multiply a 3D vector by a scalar.
<a href="#">operator*</a>	Multiply a 3D vector by a scalar (scalar data contained in vector data type).
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#">operator+</a>	Add two 3D vectors.
<a href="#">operator+</a>	Add a 3D vector to a 3D point.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3D vector.
<a href="#">operator-</a>	Subtract a 3D vector from another 3D vector.
<a href="#">operator-</a>	Negate all elements of a 3D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3D vector.
<a href="#">operator/</a>	Divide a 3D vector by a scalar.
<a href="#">operator/</a>	Divide a 3D vector by a scalar (scalar data contained in vector data type).
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar.
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar (scalar data contained in vector data type).
<a href="#">operator=</a>	Assign one 3D vector to another.
<a href="#">operator[]</a>	Subscripting operator to set or get an element.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">set128</a>	Set vector float data in a 3D vector.
<a href="#">setElem</a>	Set an $x$ , $y$ , or $z$ element of a 3D vector by index.
<a href="#">setElem</a>	Set an $x$ , $y$ , or $z$ element of a 3D vector by index (scalar data contained in vector data type).

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>setExtVector</u></a>	Set clang extended vector float data in a 3D vector.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 3D vector.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 3D vector (scalar data contained in vector data type).
<a href="#"><u>setXY</u></a>	Set the <i>x</i> and <i>y</i> elements of a 3D vector.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 3D vector.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 3D vector (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 3D vector.
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 3D vector (scalar data contained in vector data type).
<a href="#"><u>Vector3</u></a>	Default constructor; does no initialization.
<a href="#"><u>Vector3</u></a>	Copy constructor.
<a href="#"><u>Vector3</u></a>	Construct a 3D vector from <i>x</i> , <i>y</i> , and <i>z</i> elements.
<a href="#"><u>Vector3</u></a>	Construct a 3D vector from <i>x</i> , <i>y</i> , and <i>z</i> elements (scalar data contained in vector data type).
<a href="#"><u>Vector3</u></a>	Construct a 3D vector from a 2D vector and a scalar.
<a href="#"><u>Vector3</u></a>	Construct a 3D vector from a 2D vector and a scalar (scalar data contained in vector data type).
<a href="#"><u>Vector3</u></a>	Copy elements from a 3D point into a 3D vector.
<a href="#"><u>Vector3</u></a>	Set all elements of a 3D vector to the same scalar value.
<a href="#"><u>Vector3</u></a>	Set all elements of a 3D vector to the same scalar value (scalar data contained in vector data type).
<a href="#"><u>Vector3</u></a>	Set vector float data in a 3D vector.
<a href="#"><u>Vector3</u></a>	Construct a 3D vector from clang extended vector float data.
<a href="#"><u>xAxis</u></a>	Construct x axis.
<a href="#"><u>yAxis</u></a>	Construct y axis.
<a href="#"><u>zAxis</u></a>	Construct z axis.
<a href="#"><u>zero</u></a>	Construct zero.

# Constructors and Destructors

## Vector3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector3(
                        const Vector3 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from  $x$ ,  $y$ , and  $z$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        float x,
                        float y,
                        float z
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.

## Return Values

None

## Description

Construct a 3D vector containing the specified  $x$ ,  $y$ , and  $z$  elements.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from *x*, *y*, and *z* elements (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        floatInVec arg x,
                        floatInVec arg y,
                        floatInVec arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value (stored in vector data type).
<i>y</i>	Scalar value (stored in vector data type).
<i>z</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D vector containing the specified *x*, *y*, and *z* elements.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from a 2D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        Vector2 arg_xy,
                        float z
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value.

## Return Values

None

## Description

Construct a 3D vector with the *x* and *y* elements of the specified 2D vector and with the *z* element set to the specified scalar.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from a 2D vector and a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        Vector2 arg xy,
                        floatInVec arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D vector with the *x* and *y* elements of the specified 2D vector and with the *z* element set to the specified scalar.

SCE CONFIDENTIAL

# Vector3

Copy elements from a 3D point into a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        Point3 arg pnt
                    );
                };
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Construct a 3D vector containing the *x*, *y*, and *z* elements of the specified 3D point.

SCE CONFIDENTIAL

# Vector3

Set all elements of a 3D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 3D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector3

Set all elements of a 3D vector to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector3

Set vector float data in a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        vec_float4_arg vf4
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf4* Initial value.

## Return Values

None

## Description

Construct a 3D vector whose internal vector float data is set to the vector float argument.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from clang extended vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        vec_float3_ext_arg vf3
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf3* Initial value.

## Return Values

None

## Description

Construct a 3D vector whose internal vector float data is set to the clang extended vector float argument.

# Operator Methods

## operator vec\_float3\_ext

Implicit cast to clang extended vector `vec_float3_ext`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE operator vec_float3_ext() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to clang extended vector `vec_float3_ext`.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3D vector by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

Product of the specified 3D vector and scalar

## **Description**

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Product of the specified 3D vector and scalar

## **Description**

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator*=(floatInVec arg scalar
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator+
                        (Vector3 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Sum of the specified 3D vectors

## Description

Add two 3D vectors.

SCE CONFIDENTIAL

# operator+

Add a 3D vector to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Point3 operator+
                        (Point3 arg pnt
                     ) const;
                }
            }
        }
    }
}
```

## Arguments

*pnt* 3D point.

## Return Values

Sum of the specified 3D vector and 3D point

## Description

Add a 3D vector to a 3D point.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator+=(
                        Vector3 arg
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and addition with a 3D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3D vector from another 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

`vec`              3D vector.

## **Return Values**

Difference of the specified 3D vectors

## **Description**

Subtract a 3D vector from another 3D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

3D vector containing negated elements of the specified 3D vector

## **Description**

Negate all elements of a 3D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator==(Vector3 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and subtraction by a 3D vector.

SCE CONFIDENTIAL

# operator/

Divide a 3D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator/(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified 3D vector and scalar

## Description

Divide a 3D vector by a scalar.

SCE CONFIDENTIAL

# operator/

Divide a 3D vector by a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator/(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*              Scalar value (stored in vector data type).

## Return Values

Quotient of the specified 3D vector and scalar

## Description

Divide a 3D vector by a scalar.

SCE CONFIDENTIAL

# **operator/=**

Perform compound assignment and division by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator/=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*      Scalar value.

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator/=**

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator/=(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 3D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator=(Vector3 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            }
        }
    }
}
```

## **Arguments**

*vec*                   3D vector.

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Assign one 3D vector to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get an element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE VecIdx operator[](int idx)
                };
            }
        }
    }
}
```

## **Arguments**

*idx* Index, expected in the range 0-2.

## **Return Values**

[VecIdx](#), which holds a reference to the selected element

## **Description**

Subscripting operator invoked when applied to non-const [Vector3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const floatInVec operator[](
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector3](#).

# Public Static Methods

## xAxis

Construct x axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 xAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3D vector

### Description

Construct a 3D vector equal to (1,0,0).

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 yAxis();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 3D vector

## Description

Construct a 3D vector equal to (0,1,0).

# **zAxis**

Construct z axis.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 zAxis();
                };
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

The constructed 3D vector

## **Description**

Construct a 3D vector equal to (0,0,1).

SCE CONFIDENTIAL

## zero

Construct zero.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3D vector

### Description

Construct a 3D vector equal to (0,0,0).

# Public Instance Methods

## get128

Get vector float data from a 3D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a 3D vector.

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, or *z* element of a 3D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, or *z* element of a 3D vector by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# getExtVector

Get clang extended vector float data from a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float3_ext getExtVector() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Clang extended vector float data

## Description

Get clang extended vector float data from a 3D vector.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const floatInVec getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 3D vector

## Description

Get the *x* element of a 3D vector.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

A 2D vector containing *x* and *y* elements

## Description

Extract a 3D vector's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const floatInVec getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 3D vector

## Description

Get the *y* element of a 3D vector.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const floatInVec getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 3D vector

## Description

Get the *z* element of a 3D vector.

SCE CONFIDENTIAL

# set128

Set vector float data in a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE void set128(
                        vec_float4_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4*                  Vector float data.

## Return Values

None

## Description

Set vector float data in a 3D vector.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, or *z* element of a 3D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setElem(
                        int idx,
                        float value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set an *x*, *y*, or *z* element of a 3D vector by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, or *z* element of a 3D vector by index (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setElem(
                        int idx,
                        floatInVec_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D vector

## Description

Set an *x*, *y*, or *z* element of a 3D vector by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setExtVector

Set clang extended vector float data in a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE void setExtVector(
                        vec_float3_ext_arg vf3
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf3* Clang extended vector float data.

## Return Values

None

## Description

Set clang extended vector float data in a 3D vector.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setX(
                        float x
                    );
                };
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the *x* element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setX(
                        floatInVec arg_x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D vector

## Description

Set the *x* element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Set a 3D vector's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setY(
                        float y
                    );
                };
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the  $y$  element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# **setY**

Set the *y* element of a 3D vector (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setY(
                        floatInVec arg_y
                    );
                }
            }
        }
    }
}
```

## **Arguments**

<i>y</i>	Scalar value (stored in vector data type).
----------	--

## **Return Values**

A reference to the resulting 3D vector
--

## **Description**

Set the *y* element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setZ(
                        float z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the *z* element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setZ(
                        floatInVec arg z
                    );
                }
            }
        }
    }
}
```

## Arguments

*z* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 3D vector

## Description

Set the *z* element of a 3D vector to the specified scalar value.

# Protected Instance Methods

## get128Ref

Get vector float data reference from a 3D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 &get128Ref();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

A reference to the internal vector float data

### Description

Get internal vector float data reference from a 3D vector.

SCE CONFIDENTIAL

---

**sce::Vectormath::Simd::Aos::Vector4**

000004892117

# Summary

## sce::Vectormath::Simd::Aos::Vector4

A 4D vector in array-of-structures format.

### Definition

```
#include <vectormath.h>
class Vector4 {};
```

### Description

A class representing a 4D vector stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#"><u>get128</u></a>	Get vector float data from a 4D vector.
<a href="#"><u>get128Ref</u></a>	Get vector float data reference from a 4D vector.
<a href="#"><u>getElem</u></a>	Get an <i>x</i> , <i>y</i> , <i>z</i> , or <i>w</i> element of a 4D vector by index.
<a href="#"><u>getExtVector</u></a>	Get clang extended vector float data from a 4D vector.
<a href="#"><u>getW</u></a>	Get the <i>w</i> element of a 4D vector.
<a href="#"><u>getX</u></a>	Get the <i>x</i> element of a 4D vector.
<a href="#"><u>getXY</u></a>	Get the <i>x</i> and <i>y</i> elements of a 4D vector.
<a href="#"><u>getXYZ</u></a>	Get the <i>x</i> , <i>y</i> , and <i>z</i> elements of a 4D vector.
<a href="#"><u>getY</u></a>	Get the <i>y</i> element of a 4D vector.
<a href="#"><u>getZ</u></a>	Get the <i>z</i> element of a 4D vector.
<a href="#"><u>operator vec float4 ext</u></a>	Implicit cast to clang extended vector <code>vec float4 ext</code> .
<a href="#"><u>operator*</u></a>	Multiply a 4D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 4D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator*=</u></a>	Perform compound assignment and multiplication by a scalar.
<a href="#"><u>operator*=</u></a>	Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator+</u></a>	Add two 4D vectors.
<a href="#"><u>operator+=</u></a>	Perform compound assignment and addition with a 4D vector.
<a href="#"><u>operator-</u></a>	Subtract a 4D vector from another 4D vector.
<a href="#"><u>operator-</u></a>	Negate all elements of a 4D vector.
<a href="#"><u>operator-=</u></a>	Perform compound assignment and subtraction by a 4D vector.
<a href="#"><u>operator/</u></a>	Divide a 4D vector by a scalar.
<a href="#"><u>operator/</u></a>	Divide a 4D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator/=</u></a>	Perform compound assignment and division by a scalar.
<a href="#"><u>operator/=</u></a>	Perform compound assignment and division by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator=</u></a>	Assign one 4D vector to another.
<a href="#"><u>operator[]</u></a>	Subscripting operator to set or get an element.
<a href="#"><u>operator[]</u></a>	Subscripting operator to get an element.
<a href="#"><u>set128</u></a>	Set vector float data in a 4D vector.
<a href="#"><u>setElem</u></a>	Set an <i>x</i> , <i>y</i> , <i>z</i> , or <i>w</i> element of a 4D vector by index.

Methods	Description
<a href="#"><u>setElem</u></a>	Set an <i>x</i> , <i>y</i> , <i>z</i> , or <i>w</i> element of a 4D vector by index (scalar data contained in vector data type).
<a href="#"><u>setExtVector</u></a>	Set clang extended vector float data in a 4D vector.
<a href="#"><u>setW</u></a>	Set the <i>w</i> element of a 4D vector.
<a href="#"><u>setW</u></a>	Set the <i>w</i> element of a 4D vector (scalar data contained in vector data type).
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 4D vector.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a 4D vector (scalar data contained in vector data type).
<a href="#"><u>setXY</u></a>	Set the <i>x</i> and <i>y</i> elements of a 4D vector.
<a href="#"><u>setXYZ</u></a>	Set the <i>x</i> , <i>y</i> , and <i>z</i> elements of a 4D vector.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 4D vector.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a 4D vector (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 4D vector.
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a 4D vector (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Default constructor; does no initialization.
<a href="#"><u>Vector4</u></a>	Copy constructor.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from <i>x</i> , <i>y</i> , <i>z</i> , and <i>w</i> elements.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from <i>x</i> , <i>y</i> , <i>z</i> , and <i>w</i> elements (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Construct a 3D vector from a 2D vector and two scalars.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from a 2D vector and two scalars (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Construct a 3D vector from two 2D vectors.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from a 3D vector and a scalar.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from a 3D vector and a scalar (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Copy <i>x</i> , <i>y</i> , and <i>z</i> from a 3D vector into a 4D vector, and set <i>w</i> to 0.
<a href="#"><u>Vector4</u></a>	Copy <i>x</i> , <i>y</i> , and <i>z</i> from a 3D point into a 4D vector, and set <i>w</i> to 1.
<a href="#"><u>Vector4</u></a>	Copy elements from a quaternion into a 4D vector.
<a href="#"><u>Vector4</u></a>	Set all elements of a 4D vector to the same scalar value.
<a href="#"><u>Vector4</u></a>	Set all elements of a 4D vector to the same scalar value (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Set vector float data in a 4D vector.
<a href="#"><u>Vector4</u></a>	Construct a 4D vector from clang extended vector float data.
<a href="#"><u>wAxis</u></a>	Construct <i>w</i> axis.
<a href="#"><u>xAxis</u></a>	Construct <i>x</i> axis.
<a href="#"><u>yAxis</u></a>	Construct <i>y</i> axis.
<a href="#"><u>zAxis</u></a>	Construct <i>z</i> axis.
<a href="#"><u>zero</u></a>	Construct zero.

# Constructors and Destructors

## Vector4

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector4();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector4

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector4(
                        const Vector4 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        float x,
                        float y,
                        float z,
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.
$w$	Scalar value.

## Return Values

None

## Description

Construct a 4D vector containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from  $x$ ,  $y$ ,  $z$ , and  $w$  elements (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        floatInVec arg x,
                        floatInVec arg y,
                        floatInVec arg z,
                        floatInVec arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value (stored in vector data type).
$y$	Scalar value (stored in vector data type).
$z$	Scalar value (stored in vector data type).
$w$	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 4D vector containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Vector4

Construct a 3D vector from a 2D vector and two scalars.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector2 arg xy,
                        float z,
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a 3D vector with the *x* and *y* elements of the specified 2D vector and with the *z* and *w* elements set to the specified scalars.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from a 2D vector and two scalars (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector2 arg xy,
                        floatInVec arg z,
                        floatInVec arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value (stored in vector data type).
<i>w</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 3D vector with the *x* and *y* elements of the specified 2D vector and with the *z* and *w* elements set to the specified scalars.

SCE CONFIDENTIAL

# Vector4

Construct a 3D vector from two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector2 arg xy,
                        Vector2 arg zw
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>zw</i>	2D vector.

## Return Values

None

## Description

Construct a 4D vector with the *x* and *y* elements of the first specified 2D vector and with the *z* and *w* elements from the *x* and *y* elements of the second specified 2D vector.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from a 3D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector3 arg xyz,
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from a 3D vector and a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector3 arg xyz,
                        floatInVec arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Vector4

Copy x, y, and z from a 3D vector into a 4D vector, and set w to 0.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to 0.

SCE CONFIDENTIAL

# Vector4

Copy x, y, and z from a 3D point into a 4D vector, and set w to 1.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Point3 arg pnt
                    );
                };
            }
        }
    }
}
```

## Arguments

pnt                   3D point.

## Return Values

None

## Description

Construct a 4D vector with the x, y, and z elements of the specified 3D point and with the w element set to 1.

SCE CONFIDENTIAL

# Vector4

Copy elements from a quaternion into a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Quat arg quat
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*                  Quaternion.

## Return Values

None

## Description

Construct a 4D vector containing the *x*, *y*, *z*, and *w* elements of the specified quaternion.

SCE CONFIDENTIAL

# Vector4

Set all elements of a 4D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        float scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 4D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector4

Set all elements of a 4D vector to the same scalar value (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        floatInVec arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

None

## Description

Construct a 4D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector4

Set vector float data in a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        vec_float4_arg vf4
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf4* Initial value.

## Return Values

None

## Description

Construct a 4D vector whose internal vector float data is set to the vector float argument.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from clang extended vector float data.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        vec_float4_ext_arg vf4
                    );
                };
            }
        }
    }
}
```

## Arguments

*vf4* Initial value.

## Return Values

None

## Description

Construct a 4D vector whose internal vector float data is set to the clang extended vector float argument.

# Operator Methods

## operator vec\_float4\_ext

Implicit cast to clang extended vector `vec_float4_ext`.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE operator vec_float4_ext() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to clang extended vector `vec_float4_ext`.

SCE CONFIDENTIAL

# operator\*

Multiply a 4D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Product of the specified 4D vector and scalar

## Description

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 4D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Product of the specified 4D vector and scalar

## **Description**

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator*=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 4D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*=**

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator*=(floatInVec arg scalar);
                };
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

A reference to the resulting 4D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator+
                        (Vector4 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Sum of the specified 4D vectors

## Description

Add two 4D vectors.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator+=(
                        Vector4 arg
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and addition with a 4D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 4D vector from another 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator-
                        (Vector4 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

`vec`      4D vector.

## **Return Values**

Difference of the specified 4D vectors

## **Description**

Subtract a 4D vector from another 4D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

4D vector containing negated elements of the specified 4D vector

## **Description**

Negate all elements of a 4D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator==(Vector4 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and subtraction by a 4D vector.

SCE CONFIDENTIAL

# operator/

Divide a 4D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator/(
                        float scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified 4D vector and scalar

## Description

Divide a 4D vector by a scalar.

SCE CONFIDENTIAL

# **operator/**

Divide a 4D vector by a scalar (scalar data contained in vector data type).

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator/(
                        floatInVec arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value (stored in vector data type).
---------------	--

## **Return Values**

Quotient of the specified 4D vector and scalar

## **Description**

Divide a 4D vector by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator/=(
                        float scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator/=(
                        floatInVec arg scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 4D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator=(Vector4 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

A reference to the resulting 4D vector

## **Description**

Assign one 4D vector to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE VecIdx operator[](int idx)
                };
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

[VecIdx](#), which holds a reference to the selected element

## Description

Subscripting operator invoked when applied to non-const [Vector4](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec operator[](
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector4](#).

# Public Static Methods

## wAxis

Construct w axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 wAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 4D vector

### Description

Construct a 4D vector equal to (0,0,0,1).

SCE CONFIDENTIAL

# xAxis

Construct x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 xAxis();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4D vector

## Description

Construct a 4D vector equal to (1,0,0,0).

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 yAxis();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4D vector

## Description

Construct a 4D vector equal to (0,1,0,0).

SCE CONFIDENTIAL

# **zAxis**

Construct z axis.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 zAxis();
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

The constructed 4D vector

## **Description**

Construct a 4D vector equal to (0,0,1,0).

SCE CONFIDENTIAL

# zero

Construct zero.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 zero();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4D vector

## Description

Construct a 4D vector equal to (0,0,0,0).

# Public Instance Methods

## get128

Get vector float data from a 4D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get internal vector float data from a 4D vector.

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, *z*, or *w* element of a 4D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
------------	-----------------------------------

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, *z*, or *w* element of a 4D vector by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# getExtVector

Get clang extended vector float data from a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4_ext getExtVector() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Clang extended vector float data

## Description

Get clang extended vector float data from a 4D vector.

SCE CONFIDENTIAL

# getW

Get the *w* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec getW() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*w* element of a 4D vector

## Description

Get the *w* element of a 4D vector.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 4D vector

## Description

Get the *x* element of a 4D vector.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a 4D vector's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getXYZ

Get the *x*, *y*, and *z* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector3 getXYZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

3D vector containing *x*, *y*, and *z* elements

## Description

Extract a 4D vector's *x*, *y*, and *z* elements into a 3D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 4D vector

## Description

Get the *y* element of a 4D vector.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const floatInVec getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 4D vector

## Description

Get the *z* element of a 4D vector.

SCE CONFIDENTIAL

# set128

Set vector float data in a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE void set128(
                        vec_float4_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4*                  Vector float data.

## Return Values

None

## Description

Set vector float data in a 4D vector.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a 4D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setElem(
                        int idx,
                        float value
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set an *x*, *y*, *z*, or *w* element of a 4D vector by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a 4D vector by index (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setElem(
                        int idx,
                        floatInVec_arg value
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Set an *x*, *y*, *z*, or *w* element of a 4D vector by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setExtVector

Set clang extended vector float data in a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE void setExtVector(
                        vec_float4_ext_arg vf4
                    );
                }
            }
        }
    }
}
```

## Arguments

*vf4* Clang extended vector float data.

## Return Values

None

## Description

Set clang extended vector float data in a 4D vector.

SCE CONFIDENTIAL

# setW

Set the *w* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setW(
                        float w
                    );
                };
            }
        }
    }
}
```

## Arguments

*w*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *w* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setW

Set the *w* element of a 4D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setW(
                        floatInVec arg w
                    );
                }
            }
        }
    }
}
```

## Arguments

*w* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Set the *w* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setX(
                        float x
                    );
                };
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *x* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 4D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setX(
                        floatInVec arg_x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Set the *x* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Set a 4D vector's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* and *w* elements.

SCE CONFIDENTIAL

# setXYZ

Set the *x*, *y*, and *z* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setXYZ(
                        Vector3 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                   3D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Set a 4D vector's *x*, *y*, and *z* elements to those of the specified 3D vector.

## Notes

This function does not change the *w* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setY(
                        float y
                    );
                };
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the  $y$  element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 4D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setY(
                        floatInVec arg_y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Set the  $y$  element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setZ(
                        float z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *z* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 4D vector (scalar data contained in vector data type).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setZ(
                        floatInVec arg z
                    );
                }
            }
        }
    }
}
```

## Arguments

*z* Scalar value (stored in vector data type).

## Return Values

A reference to the resulting 4D vector

## Description

Set the *z* element of a 4D vector to the specified scalar value.

# Protected Instance Methods

## get128Ref

Get vector float data reference from a 4D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Aos {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE vec_float4 &get128Ref();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

A reference to the internal vector float data

### Description

Get internal vector float data reference from a 4D vector.

**sce::Vectormath::Simd::boolInVec**

0000004892117

# Summary

## sce::Vectormath::Simd::boolInVec

A class representing a scalar bool value contained in a vector register.

### Definition

```
#include <vectormath.h>
class boolInVec {};
```

### Description

A class representing a scalar bool value contained in a vector register. Bool value is in 0 word slot and other slots of a vector register are undefined.

### Methods Summary

Methods	Description
<a href="#">boolInVec</a>	Default constructor; does no initialization.
<a href="#">boolInVec</a>	Copy constructor.
<a href="#">boolInVec</a>	Construct from a value converted from float.
<a href="#">boolInVec</a>	Explicit cast from bool.
<a href="#">boolInVec</a>	Construct from a slot of vec_uint2.
<a href="#">boolInVec</a>	Construct from a slot of vec_uint4.
<a href="#">boolInVec</a>	Construct from 0 word slot of vec_uint2.
<a href="#">boolInVec</a>	Construct from 0 word slot of vec_uint4.
<a href="#">get128</a>	Get vector data; a bool value is in 0 word slot, other slots are undefined.
<a href="#">get64</a>	Get vector data; a bool value is in 0 word slot, other slots are undefined.
<a href="#">getAsBool</a>	Explicit cast to bool.
<a href="#">getFastVectorType</a>	Get fast vector data; a bool value is in 0 word slot, other slots are undefined.
<a href="#">operator bool</a>	Implicit cast to bool.
<a href="#">operator!</a>	Boolean negation operator.
<a href="#">operator&amp;=</a>	Boolean and assignment operator.
<a href="#">operator=</a>	Assignment operator.
<a href="#">operator^=</a>	Boolean exclusive or assignment operator.
<a href="#">operator =</a>	Boolean or assignment operator.
<a href="#">set128</a>	Set vector data; a bool value is extracted from the 0 word slot.
<a href="#">set64</a>	Set vector data; a bool value is extracted from the 0 word slot.
<a href="#">setFastVectorType</a>	Set fast vector data; a bool value is extracted from the 0 word slot.

# Constructors and Destructors

## boolInVec

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec();
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# boolInVec

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    const boolInVec &vec
                );
            }
        }
    }
}
```

## Arguments

`vec` Scalar value (contained in a vector register).

## Return Values

None

## Description

Copy constructor; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# boolInVec

Construct from a value converted from float.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    floatInVec arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec* Scalar float value (contained in a vector register).

## Return Values

None

## Description

Construct from a value that is generated from float by standard type conversion.

SCE CONFIDENTIAL

# boolInVec

Explicit cast from bool.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    bool scalar
                );
            };
        }
    }
}
```

## Arguments

*scalar*      Scalar bool value.

## Return Values

None

## Description

Explicit cast from bool.

SCE CONFIDENTIAL

# boolInVec

Construct from a slot of `vec_uint2`.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    vec_uint2_arg vec,
                    int slot
                );
            }
        }
    }
}
```

## Arguments

<code>vec</code>	Vector unsigned int value.
<code>slot</code>	Index, expected in the range 0-1.

## Return Values

None

## Description

Construct from a slot of `vec_uint2`.

## Notes

The result is undefined if the slot is out of range 0-1.

SCE CONFIDENTIAL

# boolInVec

Construct from a slot of `vec_uint4`.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    vec_uint4_arg vec,
                    int slot
                );
            }
        }
    }
}
```

## Arguments

<code>vec</code>	Vector unsigned int value.
<code>slot</code>	Index, expected in the range 0-3.

## Return Values

None

## Description

Construct from a slot of `vec_uint4`.

## Notes

The result is undefined if the slot is out of range 0-3.

SCE CONFIDENTIAL

# boolInVec

Construct from 0 word slot of vec\_uint2.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    vec_uint2_arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec* Vector unsigned int value.

## Return Values

None

## Description

Construct from 0 word slot of vec\_uint2.

SCE CONFIDENTIAL

# boolInVec

Construct from 0 word slot of vec\_uint4.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE boolInVec(
                    vec_uint4_arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                  Vector unsigned int value.

## Return Values

None

## Description

Construct from 0 word slot of vec\_uint4.

# Operator Methods

## operator bool

Implicit cast to bool.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE operator bool() const;
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Implicit cast to bool.

### Notes

This function is disabled if `_SCE_VECTORMATH_NO_SCALAR_CAST` is defined.

SCE CONFIDENTIAL

# operator!

Boolean negation operator.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE const boolInVec operator!() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Boolean negated value

## Description

Boolean negation operator; the result is contained in a vector register.

# **operator&=**

---

Boolean and assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec &operator&=(
                    boolInVec arg vec
                );
            }
        }
    }
}
```

## **Arguments**

---

vec	Scalar value (contained in a vector register).
-----	--

## **Return Values**

---

A reference to the resulting boolean value

## **Description**

---

Boolean and assignment operator; the operand and result are contained in separate vector registers.

SCE CONFIDENTIAL

# **operator=**

---



---

Assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec &operator=(  

                    boolInVec arg vec  

                );
            }
        }
    }
}
```

## **Arguments**

---

*vec*                   Scalar value (contained in a vector register).

## **Return Values**

---

A reference to the resulting boolean value

## **Description**

---

Assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator^=**

---

Boolean exclusive or assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec &operator^=(  

                    boolInVec arg vec  

                );
            }
        }
    }
}
```

## **Arguments**

---

*vec*                   Scalar value (contained in a vector register).

## **Return Values**

---

A reference to the resulting boolean value

## **Description**

---

Boolean exclusive or assignment operator; the operand and result are contained in separate vector registers.

SCE CONFIDENTIAL

# **operator|=**

---



---

Boolean or assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE boolInVec &operator|=(  

                    boolInVec arg vec  

                );
            }
        }
    }
}
```

## **Arguments**

---

*vec*                   Scalar value (contained in a vector register).

## **Return Values**

---

A reference to the resulting boolean value

## **Description**

---

Boolean or assignment operator; the operand and result are contained in separate vector registers.

# Public Instance Methods

## get128

Get vector data; a bool value is in 0 word slot, other slots are undefined.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 get128() const;
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector bool data

### Description

Get vector data; a bool value is in 0 word slot, other slots are undefined.

### Notes

A bool value is 0 (false) or -1 (true). Unused slots are undefined.

SCE CONFIDENTIAL

# get64

Get vector data; a bool value is in 0 word slot, other slots are undefined.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 get64() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Internal vector bool data

## Description

Get vector data; a bool value is in 0 word slot, other slots are undefined.

## Notes

A bool value is 0 (false) or -1 (true). Unused slots are undefined.

SCE CONFIDENTIAL

# getAsBool

Explicit cast to bool.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE bool getAsBool() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Bool value

## Description

Explicit cast to bool.

SCE CONFIDENTIAL

# **getFastVectorType**

Get fast vector data; a bool value is in 0 word slot, other slots are undefined.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE fast_uint_vector_type
                getFastVectorType() const;
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

Internal fast vector bool data

## **Description**

Get fast vector data; a bool value is in 0 word slot, other slots are undefined.

## **Notes**

A bool value is 0 (false) or -1 (true). Unused slots are undefined.

SCE CONFIDENTIAL

# set128

Set vector data; a bool value is extracted from the 0 word slot.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void set128(
                    vec_uint4_arg vu4
                );
            }
        }
    }
}
```

## Arguments

*vu4* Vector data.

## Return Values

None

## Description

Set vector data; a bool value is extracted from the 0 word slot.

SCE CONFIDENTIAL

# set64

Set vector data; a bool value is extracted from the 0 word slot.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void set64(
                    vec_uint2_arg vu2
                );
            }
        }
    }
}
```

## Arguments

vu2                  Vector data.

## Return Values

None

## Description

Set vector data; a bool value is extracted from the 0 word slot.

SCE CONFIDENTIAL

# **setFastVectorType**

Set fast vector data; a bool value is extracted from the 0 word slot.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class boolInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void setFastVectorType(
                    fast_uint_vector_arg vu
                );
            }
        }
    }
}
```

## **Arguments**

*vu* Fast vector data.

## **Return Values**

None

## **Description**

Set fast vector data; a bool value is extracted from the 0 word slot.

**sce::Vectormath::Simd::floatInVec**

000004892117

# Summary

## sce::Vectormath::Simd::floatInVec

A class representing a scalar float value contained in a vector register.

### Definition

```
#include <vectormath.h>
class floatInVec {};
```

### Description

A class representing a scalar float value contained in a vector register. Float value is in 0 word slot and other slots of a vector register are undefined.

### Methods Summary

Methods	Description
<a href="#">floatInVec</a>	Default constructor; does no initialization.
<a href="#">floatInVec</a>	Copy constructor.
<a href="#">floatInVec</a>	Construct from a value converted from bool.
<a href="#">floatInVec</a>	Construct from a slot of vec_float2.
<a href="#">floatInVec</a>	Construct from a slot of vec_float4.
<a href="#">floatInVec</a>	Construct from 0 word slot of vec_float2.
<a href="#">floatInVec</a>	Construct from 0 word slot of vec_float4.
<a href="#">floatInVec</a>	Explicit cast from float.
<a href="#">get128</a>	Get vector data; a float value is in 0 word slot, other slots are undefined.
<a href="#">get64</a>	Get vector data; a float value is in 0 word slot, other slots are undefined.
<a href="#">getAsFloat</a>	Explicit cast to float.
<a href="#">getFastVectorType</a>	Get fast vector data; a float value is in 0 word slot, other slots are undefined.
<a href="#">operator float</a>	Implicit cast to float.
<a href="#">operator*=</a>	Multiplication assignment operator.
<a href="#">operator++</a>	Post increment (add 1.0f).
<a href="#">operator++</a>	Pre increment (add 1.0f).
<a href="#">operator+=</a>	Addition assignment operator.
<a href="#">operator-</a>	Negation operator.
<a href="#">operator--</a>	Post decrement (subtract 1.0f).
<a href="#">operator--</a>	Pre decrement (subtract 1.0f).
<a href="#">operator-=</a>	Subtraction assignment operator.
<a href="#">operator/=</a>	Division assignment operator.
<a href="#">operator=</a>	Assignment operator.
<a href="#">set128</a>	Set vector float data; a float value is extracted from the 0 word slot.
<a href="#">set64</a>	Set vector float data; a float value is extracted from the 0 word slot.
<a href="#">setFastVectorType</a>	Set fast vector float data; a float value is extracted from the 0 word slot.

# Constructors and Destructors

## floatInVec

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec();
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# floatInVec

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    const floatInVec &vec
                );
            };
        }
    }
}
```

## Arguments

`vec` Scalar value (contained in a vector register).

## Return Values

None

## Description

Copy constructor; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# floatInVec

Construct from a value converted from bool.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    boolInVec arg vec
                );
            };
        }
    }
}
```

## Arguments

`vec` Scalar bool value (contained in a vector register).

## Return Values

None

## Description

Construct from a value that is generated from bool by standard type conversion.

SCE CONFIDENTIAL

# floatInVec

Construct from a slot of vec\_float2.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    vec_float2_arg vec,
                    int slot
                );
            }
        }
    }
}
```

## Arguments

vec	Vector float value.
slot	Index, expected in the range 0-1.

## Return Values

None

## Description

Construct from a slot of vec\_float2.

## Notes

The result is undefined if slot is out of range 0-1.

SCE CONFIDENTIAL

# floatInVec

Construct from a slot of vec\_float4.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    vec_float4_arg vec,
                    int slot
                );
            }
        }
    }
}
```

## Arguments

vec	Vector float value.
slot	Index, expected in the range 0-3.

## Return Values

None

## Description

Construct from a slot of vec\_float4.

## Notes

The result is undefined if slot is out of range 0-3.

SCE CONFIDENTIAL

# floatInVec

Construct from 0 word slot of vec\_float2.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    vec_float2_arg vec
                );
            };
        }
    }
}
```

## Arguments

vec                  Vector float value.

## Return Values

None

## Description

Construct from 0 word slot of vec\_float2.

SCE CONFIDENTIAL

# floatInVec

Construct from 0 word slot of vec\_float4.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    vec_float4_arg vec
                );
            };
        }
    }
}
```

## Arguments

vec                  Vector float value.

## Return Values

None

## Description

Construct from 0 word slot of vec\_float4.

SCE CONFIDENTIAL

# floatInVec

Explicit cast from float.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                explicit SCE_VECTORMATH_ALWAYS_INLINE floatInVec(
                    float scalar
                );
            };
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Explicit cast from float.

# Operator Methods

## **operator float**

Implicit cast to float.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE operator float() const;
            }
        }
    }
}
```

### **Arguments**

None

### **Return Values**

None

### **Description**

Implicit cast to float.

### **Notes**

This function is only defined when `SCE_VECTORMATH_NO_SCALAR_CAST` is undefined. (There is the option for the purpose of finding LHS by implicit cast.)

SCE CONFIDENTIAL

# **operator\*=**

Multiplication assignment operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator*=(floatInVec arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*            Scalar value (contained in a vector register).

## **Return Values**

A reference to the resulting value

## **Description**

Multiplication assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# operator++

Post increment (add 1.0f).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator++(int);
            }
        }
    }
}
```

## Arguments

None

## Return Values

Old value

## Description

Post increment (add 1.0f); the result is contained in a vector register.

SCE CONFIDENTIAL

# operator++

Pre increment (add 1.0f).

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator++();
            }
        }
    }
}
```

## Arguments

None

## Return Values

A reference to the updated value

## Description

Pre increment (add 1.0f); the result is contained in a vector register.

SCE CONFIDENTIAL

# operator+=

Addition assignment operator.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator+=(floatInVec arg)
                vec;
            };
        }
    }
}
```

## Arguments

`vec` Scalar value (contained in a vector register).

## Return Values

A reference to the resulting value

## Description

Addition assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator-**

Negation operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator-() const;
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

Negated value

## **Description**

Negation operator; the result is contained in a vector register.

SCE CONFIDENTIAL

## **operator--**

---

Post decrement (subtract 1.0f).

### **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE const floatInVec operator--(int);
            }
        }
    }
}
```

### **Arguments**

---

None

### **Return Values**

---

Old value

### **Description**

---

Post decrement (subtract 1.0f); the result is contained in a vector register.

SCE CONFIDENTIAL

## **operator--**

---

Pre decrement (subtract 1.0f).

### **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator--();
            }
        }
    }
}
```

### **Arguments**

---

None

### **Return Values**

---

A reference to the updated value

### **Description**

---

Pre decrement (subtract 1.0f); the result is contained in a vector register.

SCE CONFIDENTIAL

# **operator-=**

Subtraction assignment operator.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator-=(
                    floatInVec arg
                );
            };
        }
    }
}
```

## **Arguments**

*vec*                   Scalar value (contained in a vector register).

## **Return Values**

A reference to the resulting value

## **Description**

Subtraction assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

SCE CONFIDENTIAL

# **operator/=**

---

Division assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator/=(floatInVec arg)
                {
                    vec = arg;
                    return *this;
                }
            };
        }
    }
}
```

## **Arguments**

---

*vec*                   Scalar value (contained in a vector register).

## **Return Values**

---

A reference to the resulting value

## **Description**

---

Division assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# **operator=**

---



---

Assignment operator.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE floatInVec &operator=(floatInVec arg vec
                );
            }
        }
    }
}
```

## **Arguments**

---

vec	Scalar value (contained in a vector register).
-----	--

## **Return Values**

---

A reference to the resulting value

## **Description**

---

Assignment operator; each operand is contained in its own vector register, and the result is contained in another vector register.

# Public Instance Methods

## get128

Get vector data; a float value is in 0 word slot, other slots are undefined.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE vec_float4 get128() const;
            }
        }
    }
}
```

### Arguments

None

### Return Values

Internal vector float data

### Description

Get vector data; a float value is in 0 word slot, other slots are undefined.

### Notes

Unused slots are undefined.

SCE CONFIDENTIAL

# get64

Get vector data; a float value is in 0 word slot, other slots are undefined.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE vec_float2 get64() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Internal vector float data

## Description

Get vector data; a float value is in 0 word slot, other slots are undefined.

## Notes

Unused slots are undefined.

SCE CONFIDENTIAL

# getAsFloat

Explicit cast to float.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE float getAsFloat() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Float value

## Description

Explicit cast to float.

SCE CONFIDENTIAL

# getFastVectorType

Get fast vector data; a float value is in 0 word slot, other slots are undefined.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE fast_float_vector_type
                getFastVectorType() const;
            }
        }
    }
}
```

## Arguments

None

## Return Values

Internal fast vector float data

## Description

Get fast vector data; a float value is in 0 word slot, other slots are undefined.

## Notes

Unused slots are undefined.

SCE CONFIDENTIAL

# set128

Set vector float data; a float value is extracted from the 0 word slot.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void set128(
                    vec_float4_arg vf4
                );
            }
        }
    }
}
```

## Arguments

*vf4* Vector float data.

## Return Values

None

## Description

Set vector float data; a float value is extracted from the 0 word slot.

SCE CONFIDENTIAL

# set64

Set vector float data; a float value is extracted from the 0 word slot.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void set64(
                    vec_float2_arg vf2
                );
            }
        }
    }
}
```

## Arguments

vf2                  Vector float data.

## Return Values

None

## Description

Set vector float data; a float value is extracted from the 0 word slot.

SCE CONFIDENTIAL

# **setFastVectorType**

Set fast vector float data; a float value is extracted from the 0 word slot.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            class floatInVec {
                SCE_VECTORMATH_ALWAYS_INLINE void setFastVectorType(
                    fast_float_vector_arg vf
                );
            }
        }
    }
}
```

## **Arguments**

*vf* Fast vector float data.

## **Return Values**

None

## **Description**

Set fast vector float data; a float value is extracted from the 0 word slot.

SCE CONFIDENTIAL

---

**sce::Vectormath::Simd::Soa**

000004892117

# Summary

## sce::Vectormath::Simd::Soa

The namespace containing structure-of-arrays (SoA) classes.

### Definition

```
namespace Soa { }
```

### Description

The namespace containing structure-of-arrays (SoA) classes.

### Function Summary

Function	Description
<a href="#">absPerElem</a>	Compute the absolute value of a 2D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3D point per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 2x2 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3x3 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4x4 matrix per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3x4 transformation matrix per element.
<a href="#">affineInverse</a>	Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix.
<a href="#">allElemEqual</a>	All of the elements of the first 2D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 3D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 4D vector are equal to the corresponding element in the second.
<a href="#">allElemEqual</a>	All of the elements of the first 3D point are equal to the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 2D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 3D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThan</a>	All of the elements of the first 4D vector are greater than the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.
<a href="#">allElemGreaterThanOrEqual</a>	All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 2D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 3D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 4D vector are less than the corresponding element in the second.
<a href="#"><u>allElemLessThan</u></a>	All of the elements of the first 3D point are less than the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemLessThanOrEqual</u></a>	All of the elements of the first 3D point are less than or equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 2D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 3D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 4D vector are not equal to the corresponding element in the second.
<a href="#"><u>allElemNotEqual</u></a>	All of the elements of the first 3D point are not equal to the corresponding element in the second.
<a href="#"><u>angle</u></a>	Compute the angle of a 2D vector against the x-axis.
<a href="#"><u>angleBetween</u></a>	Compute the angle between two 2D vectors.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 2x2 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 3x3 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 4x4 matrix.
<a href="#"><u>appendScale</u></a>	Append (post-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 2D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 3D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 4D vector between corresponding elements specifying minimum and maximum values.
<a href="#"><u>clampPerElem</u></a>	Clamp each element of a 3D point between corresponding elements specifying minimum and maximum values.
<a href="#"><u>conj</u></a>	Compute the conjugate of a quaternion.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 2D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 3D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 4D vector to another, per element.
<a href="#"><u>copySignPerElem</u></a>	Copy sign from one 3D point to another, per element.
<a href="#"><u>cross</u></a>	Compute cross product of two 3D vectors.
<a href="#"><u>crossMatrix</u></a>	Cross-product matrix of a 3D vector.
<a href="#"><u>crossMatrixMul</u></a>	Create cross-product matrix and multiply.
<a href="#"><u>determinant</u></a>	Compute the determinant of the matrix created from two 2D vectors columns.
<a href="#"><u>determinant</u></a>	Determinant of a 2x2 matrix.
<a href="#"><u>determinant</u></a>	Determinant of a 3x3 matrix.
<a href="#"><u>determinant</u></a>	Determinant of a 4x4 matrix.
<a href="#"><u>dist</u></a>	Compute the distance between two 3D points.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>distFromOrigin</u></a>	Compute the distance of a 3D point from the coordinate-system origin.
<a href="#"><u>distSqr</u></a>	Compute the square of the distance between two 3D points.
<a href="#"><u>distSqrFromOrigin</u></a>	Compute the square of the distance of a 3D point from the coordinate-system origin.
<a href="#"><u>divPerElem</u></a>	Divide two 2D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 3D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 4D vectors per element.
<a href="#"><u>divPerElem</u></a>	Divide two 3D points per element.
<a href="#"><u>dot</u></a>	Compute the dot product of two 2D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two 3D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two 4D vectors.
<a href="#"><u>dot</u></a>	Compute the dot product of two quaternions.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 2x2 matrix.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 3x3 matrix.
<a href="#"><u>inverse</u></a>	Compute the inverse of a 4x4 matrix.
<a href="#"><u>inverse</u></a>	Inverse of a 3x4 transformation matrix.
<a href="#"><u>length</u></a>	Compute the length of a 2D vector.
<a href="#"><u>length</u></a>	Compute the length of a 3D vector.
<a href="#"><u>length</u></a>	Compute the length of a 4D vector.
<a href="#"><u>length</u></a>	Compute the length of a quaternion.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 2D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 3D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 4D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a quaternion.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 2D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3D points.
<a href="#"><u>lerp</u></a>	Linear interpolation between two quaternions.
<a href="#"><u>loadXYArray</u></a>	Load four two-float 2D vectors, stored in two quadwords.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3D vectors, stored in three quadwords.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3D points, stored in three quadwords.
<a href="#"><u>maxElem</u></a>	Maximum element of a 2D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 4D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3D point.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 2D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 4D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3D points per element.
<a href="#"><u>minElem</u></a>	Minimum element of a 2D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 4D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3D point.
<a href="#"><u>minPerElem</u></a>	Minimum of two 2D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 4D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3D points per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 2D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3D points per element.

SCE CONFIDENTIAL

Function	Description
<a href="#"><u>mulPerElem</u></a>	Multiply two 2x2 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3x3 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4x4 matrices per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3x4 transformation matrices per element.
<a href="#"><u>norm</u></a>	Compute the norm of a quaternion.
<a href="#"><u>normalize</u></a>	Normalize a 2D vector.
<a href="#"><u>normalize</u></a>	Normalize a 3D vector.
<a href="#"><u>normalize</u></a>	Normalize a 4D vector.
<a href="#"><u>normalize</u></a>	Normalize a quaternion.
<a href="#"><u>operator*</u></a>	Multiply a 2D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 3D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 4D vector by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a quaternion by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 2x2 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 3x3 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 4x4 matrix by a scalar.
<a href="#"><u>orthoInverse</u></a>	Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix with an orthogonal upper-left 3x3 submatrix.
<a href="#"><u>orthoInverse</u></a>	Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.
<a href="#"><u>outer</u></a>	Outer product of two 3D vectors.
<a href="#"><u>outer</u></a>	Outer product of two 4D vectors.
<a href="#"><u>perp</u></a>	Compute a 2D vector perpendicular to the 2D vector.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 2x2 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 3x3 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 4x4 matrix.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#"><u>print</u></a>	Print a 2D vector.
<a href="#"><u>print</u></a>	Print a 2D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3D vector.
<a href="#"><u>print</u></a>	Print a 3D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 4D vector.
<a href="#"><u>print</u></a>	Print a 4D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3D point.
<a href="#"><u>print</u></a>	Print a 3D point and an associated string identifier.
<a href="#"><u>print</u></a>	Print a quaternion.
<a href="#"><u>print</u></a>	Print a quaternion and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 2x2 matrix.
<a href="#"><u>print</u></a>	Print a 2x2 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3x3 matrix.
<a href="#"><u>print</u></a>	Print a 3x3 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 4x4 matrix.
<a href="#"><u>print</u></a>	Print a 4x4 matrix and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3x4 transformation matrix.
<a href="#"><u>print</u></a>	Print a 3x4 transformation matrix and an associated string identifier.
<a href="#"><u>projection</u></a>	Scalar projection of a 3D point on a unit-length 3D vector.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 2D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 4D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3D point per element.
<a href="#"><u>rotate</u></a>	Rotate a 2D vector.

Function	Description
<a href="#">rotate</a>	Use a unit-length quaternion to rotate a 3D vector.
<a href="#">rowMul</a>	Pre-multiply a row vector by a 3x3 matrix.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 2D vector per element.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 3D vector per element.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 4D vector per element.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 3D point per element.
<a href="#">scale</a>	Apply uniform scale to a 3D point.
<a href="#">scale</a>	Apply non-uniform scale to a 3D point.
<a href="#">select</a>	Conditionally select between two 2D vectors.
<a href="#">select</a>	Conditionally select between two 3D vectors.
<a href="#">select</a>	Conditionally select between two 4D vectors.
<a href="#">select</a>	Conditionally select between two 3D points.
<a href="#">select</a>	Conditionally select between two quaternions.
<a href="#">select</a>	Conditionally select between two 2x2 matrices.
<a href="#">select</a>	Conditionally select between two 3x3 matrices.
<a href="#">select</a>	Conditionally select between two 4x4 matrices.
<a href="#">select</a>	Conditionally select between two 3x4 transformation matrices.
<a href="#">slerp</a>	Spherical linear interpolation between two 2D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two 3D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two 4D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two quaternions.
<a href="#">sqrtPerElem</a>	Compute the square root of a 2D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 3D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 4D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 3D point per element.
<a href="#">squad</a>	Spherical quadrangle interpolation.
<a href="#">storeHalfFloats</a>	Store eight slots of two SoA 2D vectors as half-floats.
<a href="#">storeHalfFloats</a>	Store eight slots of two SoA 3D vectors as half-floats.
<a href="#">storeHalfFloats</a>	Store four slots of an SoA 4D vector as half-floats.
<a href="#">storeHalfFloats</a>	Store eight slots of two SoA 3D points as half-floats.
<a href="#">storeXYArray</a>	Store four slots of an SoA 2D vector in two quadwords.
<a href="#">storeXYZArray</a>	Store four slots of an SoA 3D vector in three quadwords.
<a href="#">storeXYZArray</a>	Store four slots of an SoA 3D point in three quadwords.
<a href="#">sum</a>	Compute the sum of all elements of a 2D vector.
<a href="#">sum</a>	Compute the sum of all elements of a 3D vector.
<a href="#">sum</a>	Compute the sum of all elements of a 4D vector.
<a href="#">sum</a>	Compute the sum of all elements of a 3D point.
<a href="#">transpose</a>	Transpose of a 2x2 matrix.
<a href="#">transpose</a>	Transpose of a 3x3 matrix.
<a href="#">transpose</a>	Transpose of a 4x4 matrix.

## Inner Classes, Structures, and Namespaces

Item	Description
<a href="#">sce::Vectormath::Simd::Soa::Matrix2</a>	A set of four 2x2 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Matrix3</a>	A set of four 3x3 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Matrix4</a>	A set of four 4x4 matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Point3</a>	A set of four 3D points in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Quat</a>	A set of four quaternions in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Transform3</a>	A set of four 3x4 transformation matrices in structure-of-arrays format.
<a href="#">sce::Vectormath::Simd::Soa::Vector2</a>	A set of four 2D vectors in structure-of-arrays format.

SCE CONFIDENTIAL

Item	Description
<a href="#"><u>sce::Vectormath::Simd::Soa::Vector3</u></a>	A set of four 3D vectors in structure-of-arrays format.
<a href="#"><u>sce::Vectormath::Simd::Soa::Vector4</u></a>	A set of four 4D vectors in structure-of-arrays format.

000004892117

# Type Definitions

## Matrix2\_arg

Type used when passing [Matrix2](#) as a function argument.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Matrix2 SCE_VECTORMATH_SIMD_SOA_MATRIX_ARG
                Matrix2_arg;
            }
        }
    }
}
```

### Description

Type used when passing [Matrix2](#) as a function argument.

# **Matrix3\_arg**

Type used when passing [Matrix3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Matrix3 SCE_VECTORMATH SIMD_SOA_MATRIX_ARG
                Matrix3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Matrix3](#) as a function argument.

# **Matrix4\_arg**

Type used when passing [Matrix4](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Matrix4 SCE_VECTORMATH SIMD_SOA_MATRIX_ARG
                Matrix4_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Matrix4](#) as a function argument.

## **Point3\_arg**

Type used when passing [Point3](#) as a function argument.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Point3 SCE_VECTORMATH SIMD_SOA_VECTOR_ARG
                Point3_arg;
            }
        }
    }
}
```

### **Description**

Type used when passing [Point3](#) as a function argument.

SCE CONFIDENTIAL

## **Quat\_arg**

---

Type used when passing [Quat](#) as a function argument.

### **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Quat SCE_VECTORMATH_SIMD_SOA_VECTOR_ARG Quat_arg;
            }
        }
    }
}
```

### **Description**

---

Type used when passing [Quat](#) as a function argument.

# **Transform3\_arg**

Type used when passing [Transform3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Transform3 SCE_VECTORMATH SIMD_SOA_MATRIX_ARG
                Transform3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Transform3](#) as a function argument.

# **Vector2\_arg**

Type used when passing [Vector2](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Vector2 SCE_VECTORMATH SIMD_SOA_VECTOR_ARG
                Vector2_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector2](#) as a function argument.

# **Vector3\_arg**

Type used when passing [Vector3](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Vector3 SCE_VECTORMATH SIMD_SOA_VECTOR_ARG
                Vector3_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector3](#) as a function argument.

# **Vector4\_arg**

Type used when passing [Vector4](#) as a function argument.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                typedef const Vector4 SCE_VECTORMATH SIMD_SOA_VECTOR_ARG
                Vector4_arg;
            }
        }
    }
}
```

## **Description**

Type used when passing [Vector4](#) as a function argument.

# 2D Vector Functions

## absPerElem

Compute the absolute value of a 2D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 absPerElem(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec*                  2D vector.

### Return Values

2D vector in which each element is the absolute value of the corresponding element of *vec*

### Description

Compute the absolute value of each element of a 2D vector.

# allElemEqual

All of the elements of the first 2D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemEqual(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True (all bits set) if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 2D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false (zero bits per lane) if any element has a NaN value.

SCE CONFIDENTIAL

# allElemGreaterThan

All of the elements of the first 2D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThan(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True (all bits set) if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 2D vector are greater than the corresponding element in the second.

# allElemGreaterThanOrEqual

All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThanOrEqual(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True (all bits set) if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 2D vector are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThan

All of the elements of the first 2D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThan(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True (all bits set) if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 2D vector are less than the corresponding element in the second.

# allElemLessThanOrEqual

All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThanOrEqual(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

True (all bits set) if all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 2D vector are less than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemNotEqual**

All of the elements of the first 2D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemNotEqual(
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 2D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# angle

Compute the angle of a 2D vector against the x-axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 angle(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                    2D vector.

## Return Values

Angle of the specified 2D vector against the x-axis

## Description

Compute the angle of a 2D vector against the x-axis.

# angleBetween

Compute the angle between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 angleBetween(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Angle between two specified 2D vectors

## Description

Compute the angle between two 2D vectors.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 2D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 clampPerElem(
                    Vector2_arg vec,
                    Vector2_arg clampMin,
                    Vector2_arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector to be clamped.
<i>clampMin</i>	2D vector containing the minimum values of the range.
<i>clampMax</i>	2D vector containing the maximum values of the range.

## Return Values

2D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 2D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 copySignPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

# determinant

Compute the determinant of the matrix created from two 2D vectors columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 determinant(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Determinant between two 2D vectors

## Description

Compute the determinant of the matrix created from two 2D vectors columns.

# divPerElem

Divide two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 divPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the quotient of the corresponding elements of the specified 2D vectors

## Description

Divide two 2D vectors element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 dot(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Dot product of the specified 2D vectors

## Description

Compute the dot product of two 2D vectors.

SCE CONFIDENTIAL

# length

Compute the length of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 length(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

Length of the specified 2D vector

## Description

Compute the length of a 2D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 lengthSqr(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

**vec**                  2D vector.

## **Return Values**

Square of the length of the specified 2D vector

## **Description**

Compute the square of the length of a 2D vector.

# lerp

Linear interpolation between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 lerp(
                    vec_float4_arg t,
                    Vector2_arg vec0,
                    Vector2_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

Interpolated 2D vector

## Description

Linearly interpolate between two 2D vectors.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadXYArray

Load four two-float 2D vectors, stored in two quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void loadXYArray(
                    Vector2 &vec,
                    const vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 2D vector.
<i>threeQuads</i>	Array of two quadwords containing 8 floats.

## Return Values

None

## Description

Load four two-float 2D vectors, stored in two quadwords as {x0, y0, x1, y1, x2, y2, x3, y3}, into four slots of an SoA 2D vector.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 maxElem(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    2D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 2D vector.

SCE CONFIDENTIAL

# maxPerElem

Maximum of two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 maxPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the maximum of the corresponding elements of the specified 2D vectors

## Description

Create a 2D vector in which each element is the maximum of the corresponding elements of the specified 2D vectors.

# minElem

Minimum element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 minElem(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      2D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 2D vector.

# minPerElem

Minimum of two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 minPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the minimum of the corresponding elements of the specified 2D vectors

## Description

Create a 2D vector in which each element is the minimum of the corresponding elements of two specified 2D vectors.

# mulPerElem

Multiply two 2D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 mulPerElem(
                    Vector2 arg vec0,
                    Vector2 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.

## Return Values

2D vector in which each element is the product of the corresponding elements of the specified 2D vectors

## Description

Multiply two 2D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 normalize(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      2D vector.

## Return Values

The specified 2D vector scaled to unit length

## Description

Compute a normalized 2D vector.

## Notes

The result is unpredictable when all elements of *vec* are at or near zero.

SCE CONFIDENTIAL

# **operator\***

Multiply a 2D vector by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 operator*(
                    vec_float4 arg scalar,
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

<i>scalar</i>	Scalar value.
<i>vec</i>	2D vector.

## **Return Values**

Scalar product of *vec* and *scalar*

## **Description**

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# perp

Compute a 2D vector perpendicular to the 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 perp(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A 2D vector perpendicular to the specified 2D vector

## Description

Compute a 2D vector perpendicular to the 2D vector.

# print

Print a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      2D vector.

## Return Values

None

## Description

Print a 2D vector. Prints the 2D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 2D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector2 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector.
<i>name</i>	String printed with the 2D vector.

## Return Values

None

## Description

Print a 2D vector and an associated string identifier. Prints the 2D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 2D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 recipPerElem(
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

2D vector in which each element is the reciprocal of the corresponding element of the specified 2D vector

## **Description**

Create a 2D vector in which each element is the reciprocal of the corresponding element of the specified 2D vector.

## **Notes**

Floating-point behavior matches standard library function `recipf4`.

SCE CONFIDENTIAL

# rotate

Rotate a 2D vector .

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 rotate(
                    vec_float4 arg rot,
                    Vector2 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>rot</i>	Angle (radians).
<i>vec</i>	2D vector.

## Return Values

The rotated 2D vector

## Description

Rotate a 2D vector by a specified angle (radians).

# **rsqrtPerElem**

Compute the reciprocal square root of a 2D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 rsqrtPerElem(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

2D vector in which each element is the reciprocal square root of the corresponding element of the specified 2D vector

## **Description**

Create a 2D vector in which each element is the reciprocal square root of the corresponding element of the specified 2D vector.

## **Notes**

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# select

Conditionally select between two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 select(
                    Vector2_arg vec0,
                    Vector2_arg vec1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.
<i>select1</i>	For each of the four word slots, this mask selects either the 2D vector in the corresponding slot of <i>vec0</i> or the 2D vector in the corresponding slot of <i>vec1</i> . A 0 bit selects from <i>vec0</i> whereas a 1 bit selects from <i>vec1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 2D vector at the corresponding slot of *vec0* or *vec1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *vec0*, and a value of 0xFFFFFFFF selects the slot of *vec1*.

## Description

Conditionally select one of the 2D vectors at each of the corresponding slots of *vec0* or *vec1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 2D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 slerp(
                    vec_float4_arg t,
                    Vector2_arg unitVec0,
                    Vector2_arg unitVec1,
                    vec_float4_arg tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_V4
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	2D vector, expected to be unit-length.
<i>unitVec1</i>	2D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 2D vector

## **Description**

Perform spherical linear interpolation between two 2D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# sqrtPerElem

Compute the square root of a 2D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector2 sqrtPerElem(
                    Vector2 arg
                );
            }
        }
    }
}
```

## Arguments

`vec`      2D vector.

## Return Values

2D vector in which each element is the square root of the corresponding element of the specified 2D vector

## Description

Create a 2D vector in which each element is the square root of the corresponding element of the specified 2D vector.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

SCE CONFIDENTIAL

# storeHalfFloats

Store eight slots of two SoA 2D vectors as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector2_arg vec0,
                    Vector2_arg vec1,
                    vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	2D vector.
<i>vec1</i>	2D vector.
<i>twoQuads</i>	An output array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Store eight slots of two SoA 2D vectors in two quadwords of half-float values. If the slots are numbered with *vec0* as 0..2 and *vec1* as 4..7, the output is

{x0, y0, x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7}.

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeXYArray

Store four slots of an SoA 2D vector in two quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeXYArray(
                    Vector2 arg vec,
                    vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	2D vector.
<i>threeQuads</i>	An output array of two quadwords containing 8 floats.

## Return Values

None

## Description

Store four slots of an SoA 2D vector in two quadwords as {x0, y0, x1, y1, x2, y2, x3, y3}.

SCE CONFIDENTIAL

# sum

Compute the sum of all elements of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 sum(
                    Vector2_arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

Sum of all elements of *vec*

## Description

Compute the sum of all elements of a 2D vector.

# 3D Vector Functions

## absPerElem

Compute the absolute value of a 3D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 absPerElem(
                    Vector3_arg vec
                );
            }
        }
    }
}
```

### Arguments

vec                   3D vector.

### Return Values

3D vector in which each element is the absolute value of the corresponding element of vec

### Description

Compute the absolute value of each element of a 3D vector.

# allElemEqual

All of the elements of the first 3D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemEqual(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

*vec0* 3D vector.  
*vec1* 3D vector.

## Return Values

True (all bits set) if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false (zero bits per lane) if any element has a NaN value.

SCE CONFIDENTIAL

# allElemGreaterThan

All of the elements of the first 3D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThan(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True (all bits set) if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D vector are greater than the corresponding element in the second.

# **allElemGreaterThanOrEqual**

All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThanOrEqual(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D vector are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThan

All of the elements of the first 3D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThan(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

True (all bits set) if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D vector are less than the corresponding element in the second.

# **allElemLessThanOrEqual**

All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThanOrEqual(
                    Vector3_arg vec0,
                    Vector3_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D vector are less than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemNotEqual**

All of the elements of the first 3D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemNotEqual(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

*vec0* 3D vector.  
*vec1* 3D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 3D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 clampPerElem(  

                    Vector3_arg vec,  

                    Vector3_arg clampMin,  

                    Vector3_arg clampMax  

                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector to be clamped.
<i>clampMin</i>	3D vector containing the minimum values of the range.
<i>clampMax</i>	3D vector containing the maximum values of the range.

## Return Values

3D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 3D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 copySignPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

SCE CONFIDENTIAL

## cross

Compute cross product of two 3D vectors.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 cross(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

### Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

### Return Values

Cross product of the specified 3D vectors

### Description

Compute cross product of two 3D vectors.

SCE CONFIDENTIAL

# crossMatrix

Cross-product matrix of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 crossMatrix(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Cross-product matrix of *vec*

## Description

Compute a matrix that, when multiplied by a 3D vector, produces the same result as a cross product with that 3D vector.

SCE CONFIDENTIAL

# crossMatrixMul

Create cross-product matrix and multiply.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 crossMatrixMul(
                    Vector3 arg vec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## Return Values

Product of cross-product matrix of *vec* and *mat*

## Description

Multiply a cross-product matrix by another matrix.

## Notes

Faster than separately creating a cross-product matrix and multiplying.

# divPerElem

Divide two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 divPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the quotient of the corresponding elements of the specified 3D vectors

## Description

Divide two 3D vectors element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 dot(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

Dot product of the specified 3D vectors

## Description

Compute the dot product of two 3D vectors.

SCE CONFIDENTIAL

# **length**

Compute the length of a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 length(  

                    Vector3_arg vec  

                );
            }
        }
    }
}
```

## **Arguments**

*vec*                    3D vector.

## **Return Values**

Length of the specified 3D vector

## **Description**

Compute the length of a 3D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 lengthSqr(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                   3D vector.

## **Return Values**

Square of the length of the specified 3D vector

## **Description**

Compute the square of the length of a 3D vector.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 lerp(
                    vec_float4 arg t,
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

Interpolated 3D vector

## Description

Linearly interpolate between two 3D vectors.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# loadXYZArray

Load four three-float 3D vectors, stored in three quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void loadXYZArray(
                    Vector3 &vec,
                    const vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	An output 3D vector.
<i>threeQuads</i>	Array of three quadwords containing 12 floats.

## Return Values

None

## Description

Load four three-float 3D vectors, stored in three quadwords as { $x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3$ }, into four slots of an SoA 3D vector.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 maxElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 3D vector.

# maxPerElem

Maximum of two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 maxPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the maximum of the corresponding elements of the specified 3D vectors

## Description

Create a 3D vector in which each element is the maximum of the corresponding elements of the specified 3D vectors.

SCE CONFIDENTIAL

# minElem

Minimum element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 minElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 3D vector.

# minPerElem

Minimum of two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 minPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the minimum of the corresponding elements of the specified 3D vectors

## Description

Create a 3D vector in which each element is the minimum of the corresponding elements of two specified 3D vectors.

SCE CONFIDENTIAL

# mulPerElem

Multiply two 3D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 mulPerElem(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

3D vector in which each element is the product of the corresponding elements of the specified 3D vectors

## Description

Multiply two 3D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 normalize(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

`vec`      3D vector.

## Return Values

The specified 3D vector scaled to unit length

## Description

Compute a normalized 3D vector.

## Notes

The result is unpredictable when all elements of `vec` are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a 3D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 operator*(
                    vec_float4 arg scalar,
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>vec</i>	3D vector.

## Return Values

Scalar product of *vec* and *scalar*

## Description

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# outer

Outer product of two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 outer(
                    Vector3 arg vec0,
                    Vector3 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.

## Return Values

The 3x3 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

Compute the outer product of two 3D vectors.

## Notes

The term "outer product", which is used in documentation for the PlayStation®2 computer entertainment console, corresponds to "cross product" in this library.

SCE CONFIDENTIAL

# print

Print a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Print a 3D vector. Prints the 3D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when SCE\_VECTORMATH\_DEBUG is defined.

SCE CONFIDENTIAL

# print

Print a 3D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector3 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>name</i>	String printed with the 3D vector.

## Return Values

None

## Description

Print a 3D vector and an associated string identifier. Prints the 3D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# recipPerElem

Compute the reciprocal of a 3D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 recipPerElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

`vec`      3D vector.

## Return Values

3D vector in which each element is the reciprocal of the corresponding element of the specified 3D vector

## Description

Create a 3D vector in which each element is the reciprocal of the corresponding element of the specified 3D vector.

## Notes

Floating-point behavior matches standard library function `recipf4`.

# rowMul

Pre-multiply a row vector by a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 rowMul(
                    Vector3 arg vec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## Return Values

Product of a row-vector and a 3x3 matrix

## Description

Transpose a 3D vector into a row vector and pre-multiply by 3x3 matrix.

SCE CONFIDENTIAL

# **rsqrtPerElem**

Compute the reciprocal square root of a 3D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 rsqrtPerElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                   3D vector.

## **Return Values**

3D vector in which each element is the reciprocal square root of the corresponding element of the specified 3D vector

## **Description**

Create a 3D vector in which each element is the reciprocal square root of the corresponding element of the specified 3D vector.

## **Notes**

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# select

Conditionally select between two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 select(
                    Vector3_arg vec0,
                    Vector3_arg vec1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>select1</i>	For each of the four word slots, this mask selects either the 3D vector in the corresponding slot of <i>vec0</i> or the 3D vector in the corresponding slot of <i>vec1</i> . A 0 bit selects from <i>vec0</i> whereas a 1 bit selects from <i>vec1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 3D vector at the corresponding slot of *vec0* or *vec1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *vec0*, and a value of 0xFFFFFFFF selects the slot of *vec1*.

## Description

Conditionally select one of the 3D vectors at each of the corresponding slots of *vec0* or *vec1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 3D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 slerp(
                    vec_float4_arg t,
                    Vector3_arg unitVec0,
                    Vector3_arg unitVec1,
                    vec_float4_arg tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_V4
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	3D vector, expected to be unit-length.
<i>unitVec1</i>	3D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 3D vector

## **Description**

Perform spherical linear interpolation between two 3D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

# sqrtPerElem

Compute the square root of a 3D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 sqrtPerElem(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

`vec` 3D vector.

## Return Values

3D vector in which each element is the square root of the corresponding element of the specified 3D vector

## Description

Create a 3D vector in which each element is the square root of the corresponding element of the specified 3D vector.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

# storeHalfFloats

Store eight slots of two SoA 3D vectors as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector3_arg vec0,
                    Vector3_arg vec1,
                    vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	3D vector.
<i>vec1</i>	3D vector.
<i>threeQuads</i>	An output array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Store eight slots of two SoA 3D vectors in three quadwords of half-float values. If the slots are numbered with *vec0* as 0..3 and *vec1* as 4..7, the output is

{x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, x5, y5, z5, x6, y6, z6, x7, y7, z7}.

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeXYZArray

Store four slots of an SoA 3D vector in three quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeXYZArray(
                    Vector3 arg vec,
                    vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	3D vector.
<i>threeQuads</i>	An output array of three quadwords containing 12 floats.

## Return Values

None

## Description

Store four slots of an SoA 3D vector in three quadwords as  
 $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3\}$ .

SCE CONFIDENTIAL

# sum

Compute the sum of all elements of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 sum(
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                    3D vector.

## Return Values

Sum of all elements of *vec*

## Description

Compute the sum of all elements of a 3D vector.

# 4D Vector Functions

## absPerElem

Compute the absolute value of a 4D vector per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 absPerElem(
                    Vector4 arg
                );
            }
        }
    }
}
```

### Arguments

*vec*                  4D vector.

### Return Values

4D vector in which each element is the absolute value of the corresponding element of *vec*

### Description

Compute the absolute value of each element of a 4D vector.

# allElemEqual

All of the elements of the first 4D vector are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemEqual(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True (all bits set) if all elements of *vec0* are equal to the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 4D vector are equal to the corresponding element in the second.

## Notes

Comparing a vector against itself will return false (zero bits per lane) if any element has a NaN value.

SCE CONFIDENTIAL

# allElemGreaterThan

All of the elements of the first 4D vector are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThan(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True (all bits set) if all elements of *vec0* are greater than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 4D vector are greater than the corresponding element in the second.

# **allElemGreaterThanOrEqual**

All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThanOrEqual(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are greater than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 4D vector are greater than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# allElemLessThan

All of the elements of the first 4D vector are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThan(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

True (all bits set) if all elements of *vec0* are less than the corresponding element in *vec1*; otherwise false (no bits set).

## Description

All of the elements of the first 4D vector are less than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemLessThanOrEqual**

All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThanOrEqual(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are less than or equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 4D vector are less than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemNotEqual**

All of the elements of the first 4D vector are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemNotEqual(
                    Vector4_arg vec0,
                    Vector4_arg vec1
                );
            }
        }
    }
}
```

## **Arguments**

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## **Return Values**

True (all bits set) if all elements of *vec0* are not equal to the corresponding element in *vec1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 4D vector are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 4D vector between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 clampPerElem(
                    Vector4 arg vec,
                    Vector4 arg clampMin,
                    Vector4 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector to be clamped.
<i>clampMin</i>	4D vector containing the minimum values of the range.
<i>clampMax</i>	4D vector containing the maximum values of the range.

## Return Values

4D vector in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

SCE CONFIDENTIAL

# copySignPerElem

Copy sign from one 4D vector to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 copySignPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element has the magnitude of the corresponding element of *vec0* and the sign of the corresponding element of *vec1*.

## Description

For each element, create a value composed of the magnitude of *vec0* and the sign of *vec1*.

SCE CONFIDENTIAL

# divPerElem

Divide two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 divPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the quotient of the corresponding elements of the specified 4D vectors

## Description

Divide two 4D vectors element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# dot

Compute the dot product of two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 dot(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

Dot product of the specified 4D vectors

## Description

Compute the dot product of two 4D vectors.

SCE CONFIDENTIAL

# length

Compute the length of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 length(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Length of the specified 4D vector

## Description

Compute the length of a 4D vector.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 lengthSqr(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

Square of the length of the specified 4D vector

## **Description**

Compute the square of the length of a 4D vector.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 lerp(
                    vec_float4 arg t,
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

Interpolated 4D vector

## Description

Linearly interpolate between two 4D vectors.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 maxElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  4D vector.

## Return Values

Maximum value of all elements of *vec*

## Description

Compute the maximum value of all elements of a 4D vector.

# maxPerElem

Maximum of two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 maxPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the maximum of the corresponding elements of the specified 4D vectors

## Description

Create a 4D vector in which each element is the maximum of the corresponding elements of the specified 4D vectors.

# minElem

Minimum element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 minElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  4D vector.

## Return Values

Minimum value of all elements of *vec*

## Description

Compute the minimum value of all elements of a 4D vector.

# minPerElem

Minimum of two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 minPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the minimum of the corresponding elements of the specified 4D vectors

## Description

Create a 4D vector in which each element is the minimum of the corresponding elements of two specified 4D vectors.

# mulPerElem

Multiply two 4D vectors per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 mulPerElem(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

4D vector in which each element is the product of the corresponding elements of the specified 4D vectors

## Description

Multiply two 4D vectors element by element.

SCE CONFIDENTIAL

# normalize

Normalize a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 normalize(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*      4D vector.

## Return Values

The specified 4D vector scaled to unit length

## Description

Compute a normalized 4D vector.

## Notes

The result is unpredictable when all elements of *vec* are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a 4D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 operator*(
                    vec_float4 arg scalar,
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>vec</i>	4D vector.

## Return Values

Scalar product of *vec* and *scalar*

## Description

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# outer

Outer product of two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 outer(
                    Vector4 arg vec0,
                    Vector4 arg vec1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.

## Return Values

The 4x4 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

Compute the outer product of two 4D vectors.

## Notes

The term "outer product", which is used in documentation for the PlayStation®2 computer entertainment console, corresponds to "cross product" in this library.

# print

Print a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

None

## Description

Print a 4D vector. Prints the 4D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when SCE\_VECTORMATH\_DEBUG is defined.

SCE CONFIDENTIAL

# print

Print a 4D vector and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Vector4 arg vec,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>name</i>	String printed with the 4D vector.

## Return Values

None

## Description

Print a 4D vector and an associated string identifier. Prints the 4D vector transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# recipPerElem

Compute the reciprocal of a 4D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 recipPerElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## Arguments

*vec*                  4D vector.

## Return Values

4D vector in which each element is the reciprocal of the corresponding element of the specified 4D vector

## Description

Create a 4D vector in which each element is the reciprocal of the corresponding element of the specified 4D vector.

## Notes

Floating-point behavior matches standard library function `recipf4`.

# **rsqrtPerElem**

Compute the reciprocal square root of a 4D vector per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 rsqrtPerElem(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

4D vector in which each element is the reciprocal square root of the corresponding element of the specified 4D vector

## **Description**

Create a 4D vector in which each element is the reciprocal square root of the corresponding element of the specified 4D vector.

## **Notes**

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# select

Conditionally select between two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 select(
                    Vector4_arg vec0,
                    Vector4_arg vec1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>vec0</i>	4D vector.
<i>vec1</i>	4D vector.
<i>select1</i>	For each of the four word slots, this mask selects either the 4D vector in the corresponding slot of <i>vec0</i> or the 4D vector in the corresponding slot of <i>vec1</i> . A 0 bit selects from <i>vec0</i> whereas a 1 bit selects from <i>vec1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 4D vector at the corresponding slot of *vec0* or *vec1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *vec0*, and a value of 0xFFFFFFFF selects the slot of *vec1*.

## Description

Conditionally select one of the 4D vectors at each of the corresponding slots of *vec0* or *vec1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two 4D vectors.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 slerp(
                    vec_float4_arg t,
                    Vector4_arg unitVec0,
                    Vector4_arg unitVec1,
                    vec_float4_arg tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_V4
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitVec0</i>	4D vector, expected to be unit-length.
<i>unitVec1</i>	4D vector, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated 4D vector

## **Description**

Perform spherical linear interpolation between two 4D vectors.

## **Notes**

The result is unpredictable if the vectors point in opposite directions. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# sqrtPerElem

Compute the square root of a 4D vector per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector4 sqrtPerElem(
                    Vector4 arg
                );
            }
        }
    }
}
```

## Arguments

`vec`      4D vector.

## Return Values

4D vector in which each element is the square root of the corresponding element of the specified 4D vector

## Description

Create a 4D vector in which each element is the square root of the corresponding element of the specified 4D vector.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

SCE CONFIDENTIAL

# storeHalfFloats

Store four slots of an SoA 4D vector as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Vector4 arg vec,
                    vec_ushort8 *twoQuads
                );
            }
        }
    }
}
```

## Arguments

<i>vec</i>	4D vector.
<i>twoQuads</i>	An output array of two quadwords containing 16 half-floats.

## Return Values

None

## Description

Store four slots of an SoA 4D vector in two quadwords of half-float values. If the slots are numbered with *vec* as 0..3, the output is {x0, y0, z0, w0, x1, y1, z1, w1, x2, y2, z2, w2, x3, y3, z3, w3}.

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 4D vector.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 sum(
                    Vector4 arg vec
                );
            }
        }
    }
}
```

### Arguments

*vec*                  4D vector.

### Return Values

Sum of all elements of *vec*

### Description

Compute the sum of all elements of a 4D vector.

# 3D Point Functions

## absPerElem

Compute the absolute value of a 3D point per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 absPerElem(
                    Point3_arg pnt
                );
            }
        }
    }
}
```

### Arguments

*pnt*                    3D point.

### Return Values

3D point in which each element is the absolute value of the corresponding element of *pnt*

### Description

Compute the absolute value of each element of a 3D point.

# allElemEqual

All of the elements of the first 3D point are equal to the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemEqual(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

True (all bits set) if all elements of *pnt0* are equal to the corresponding element in *pnt1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D point are equal to the corresponding element in the second.

## Notes

Comparing a 3D point against itself will return false (zero bits per lane) if any element has a NaN value.

SCE CONFIDENTIAL

# allElemGreaterThan

All of the elements of the first 3D point are greater than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThan(
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

True (all bits set) if all elements of *pnt0* are greater than the corresponding element in *pnt1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D point are greater than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemGreaterThanOrEqualTo**

All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemGreaterThanOrEqualTo(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True (all bits set) if all elements of *pnt0* are greater than or equal to the corresponding element in *pnt1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D point are greater than or equal to the corresponding element in the second.

# allElemLessThan

All of the elements of the first 3D point are less than the corresponding element in the second.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThan(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

True (all bits set) if all elements of *pnt0* are less than the corresponding element in *pnt1*; otherwise false (no bits set).

## Description

All of the elements of the first 3D point are less than the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemLessThanOrEqual**

All of the elements of the first 3D point are less than or equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemLessThanOrEqual (
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True (all bits set) if all elements of *pnt0* are less than or equal to the corresponding element in *pnt1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D point are less than or equal to the corresponding element in the second.

SCE CONFIDENTIAL

# **allElemNotEqual**

All of the elements of the first 3D point are not equal to the corresponding element in the second.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_uint4 allElemNotEqual(
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

True (all bits set) if all elements of *pnt0* are not equal to the corresponding element in *pnt1*; otherwise false (no bits set).

## **Description**

All of the elements of the first 3D point are not equal to the corresponding element in the second.

SCE CONFIDENTIAL

# clampPerElem

Clamp each element of a 3D point between corresponding elements specifying minimum and maximum values.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 clampPerElem(
                    Point3 arg pnt,
                    Point3 arg clampMin,
                    Point3 arg clampMax
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point to be clamped.
<i>clampMin</i>	3D point containing the minimum values of the range.
<i>clampMax</i>	3D point containing the maximum values of the range.

## Return Values

3D point in which each element has its value clamped in the range specified by the corresponding element of *clampMin* and the corresponding element of *clampMax*

## Description

For each element, create a value that is clamped between the value of the corresponding elements of *clampMin* and *clampMax*.

## Notes

Result is undefined if any minimum element is greater than the corresponding maximum element.

# copySignPerElem

Copy sign from one 3D point to another, per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 copySignPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element has the magnitude of the corresponding element of *pnt0* and the sign of the corresponding element of *pnt1*

## Description

For each element, create a value composed of the magnitude of *pnt0* and the sign of *pnt1*.

SCE CONFIDENTIAL

# dist

Compute the distance between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 dist(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Distance between two 3D points

## Description

Compute the distance between two 3D points.

SCE CONFIDENTIAL

# **distFromOrigin**

Compute the distance of a 3D point from the coordinate-system origin.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 distFromOrigin(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

Distance of a 3D point from the origin

## **Description**

Compute the distance of a 3D point from the coordinate-system origin.

SCE CONFIDENTIAL

# distSqr

Compute the square of the distance between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 distSqr(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Square of the distance between two 3D points

## Description

Compute the square of the distance between two 3D points.

SCE CONFIDENTIAL

# distSqrFromOrigin

Compute the square of the distance of a 3D point from the coordinate-system origin.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 distSqrFromOrigin(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Square of the distance of a 3D point from the origin

## Description

Compute the square of the distance of a 3D point from the coordinate-system origin.

# divPerElem

Divide two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 divPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the quotient of the corresponding elements of the specified 3D points

## Description

Divide two 3D points element by element.

## Notes

Floating-point behavior matches standard library function `divf4`.

SCE CONFIDENTIAL

# lerp

Linear interpolation between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 lerp(
                    vec_float4_arg t,
                    Point3_arg pnt0,
                    Point3_arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

Interpolated 3D point

## Description

Linearly interpolate between two 3D points.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# **loadXYZArray**

Load four three-float 3D points, stored in three quadwords.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void loadXYZArray(
                    Point3 &pnt,
                    const vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt</i>	An output 3D point.
<i>threeQuads</i>	Array of three quadwords containing 12 floats.

## **Return Values**

None

## **Description**

Load four three-float 3D points, stored in three quadwords as { $x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3$ }, into four slots of an SoA 3D point.

SCE CONFIDENTIAL

# maxElem

Maximum element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 maxElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Maximum value of all elements of *pnt*

## Description

Compute the maximum value of all elements of a 3D point.

# maxPerElem

Maximum of two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 maxPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the maximum of the corresponding elements of the specified 3D points

## Description

Create a 3D point in which each element is the maximum of the corresponding elements of the specified 3D points.

# minElem

Minimum element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 minElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Minimum value of all elements of *pnt*

## Description

Compute the minimum value of all elements of a 3D point.

# minPerElem

Minimum of two 3D points per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 minPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## Return Values

3D point in which each element is the minimum of the corresponding elements of the specified 3D points

## Description

Create a 3D point in which each element is the minimum of the corresponding elements of two specified 3D points.

# **mulPerElem**

Multiply two 3D points per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 mulPerElem(
                    Point3 arg pnt0,
                    Point3 arg pnt1
                );
            }
        }
    }
}
```

## **Arguments**

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.

## **Return Values**

3D point in which each element is the product of the corresponding elements of the specified 3D points

## **Description**

Multiply two 3D points element by element.

SCE CONFIDENTIAL

# print

Print a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Print a 3D point. Prints the 3D point transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 3D point and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Point3 arg pnt,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>name</i>	String printed with the 3D point.

## Return Values

None

## Description

Print a 3D point and an associated string identifier. Prints the 3D point transposed, that is, as a row instead of a column.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# projection

Scalar projection of a 3D point on a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 projection(
                    Point3 arg pnt,
                    Vector3 arg unitVec
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

Scalar projection of the 3D point on the unit-length 3D vector

## Description

Scalar projection of a 3D point on a unit-length 3D vector (dot product).

SCE CONFIDENTIAL

# **recipPerElem**

Compute the reciprocal of a 3D point per element.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 recipPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

3D point in which each element is the reciprocal of the corresponding element of the specified 3D point

## **Description**

Create a 3D point in which each element is the reciprocal of the corresponding element of the specified 3D point.

## **Notes**

Floating-point behavior matches standard library function `recipf4`.

SCE CONFIDENTIAL

# rsqrtPerElem

Compute the reciprocal square root of a 3D point per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 rsqrtPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt* 3D point.

## Return Values

3D point in which each element is the reciprocal square root of the corresponding element of the specified 3D point

## Description

Create a 3D point in which each element is the reciprocal square root of the corresponding element of the specified 3D point.

## Notes

Floating-point behavior matches standard library function `rsqrtf4`.

SCE CONFIDENTIAL

# scale

Apply uniform scale to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 scale(
                    Point3 arg pnt,
                    vec_float4_arg scaleVal
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>scaleVal</i>	Scalar value.

## Return Values

3D point in which every element is multiplied by the scalar value

## Description

Apply uniform scale to a 3D point.

SCE CONFIDENTIAL

# scale

Apply non-uniform scale to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 scale(
                    Point3 arg pnt,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>pnt</i>	3D point.
<i>scaleVec</i>	3D vector.

## Return Values

3D point in which each element is the product of the corresponding elements of the specified 3D point and 3D vector

## Description

Apply non-uniform scale to a 3D point.

SCE CONFIDENTIAL

# select

Conditionally select between two 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 select(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>select1</i>	For each of the four word slots, this mask selects either the 3D point in the corresponding slot of <i>pnt0</i> or the 3D point in the corresponding slot of <i>pnt1</i> . A 0 bit selects from <i>pnt0</i> whereas a 1 bit selects from <i>pnt1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 3D point at the corresponding slot of *pnt0* or *pnt1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *pnt0*, and a value of 0xFFFFFFFF selects the slot of *pnt1*.

## Description

Conditionally select one of the 3D points at each of the corresponding slots of *pnt0* or *pnt1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

# sqrtPerElem

Compute the square root of a 3D point per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Point3 sqrtPerElem(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

3D point in which each element is the square root of the corresponding element of the specified 3D point

## Description

Create a 3D point in which each element is the square root of the corresponding element of the specified 3D point.

## Notes

Floating-point behavior matches standard library function `sqrtf4`.

SCE CONFIDENTIAL

# storeHalfFloats

Store eight slots of two SoA 3D points as half-floats.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeHalfFloats(
                    Point3 arg pnt0,
                    Point3 arg pnt1,
                    vec_ushort8 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	3D point.
<i>pnt1</i>	3D point.
<i>threeQuads</i>	An output array of three quadwords containing 24 half-floats.

## Return Values

None

## Description

Store eight slots of two SoA 3D points in three quadwords of half-float values. If the slots are numbered with *pnt0* as 0..3 and *pnt1* as 4..7, the output is

{x0, y0, z0, x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, x5, y5, z5, x6, y6, z6, x7, y7, z7}.

## Notes

This transformation does not support either denormalized numbers or NaNs.

SCE CONFIDENTIAL

# storeXYZArray

Store four slots of an SoA 3D point in three quadwords.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void storeXYZArray(
                    Point3 arg pnt,
                    vec_float4 *threeQuads
                );
            }
        }
    }
}
```

## Arguments

*pnt*  
*threeQuads*

3D point.  
An output array of three quadwords containing 12 floats.

## Return Values

None

## Description

Store four slots of an SoA 3D point in three quadwords as  
 $\{x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3\}$ .

SCE CONFIDENTIAL

## sum

Compute the sum of all elements of a 3D point.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 sum(
                    Point3 arg pnt
                );
            }
        }
    }
}
```

### Arguments

*pnt*                    3D point.

### Return Values

Sum of all elements of *pnt*

### Description

Compute the sum of all elements of a 3D point.

# Quaternion Functions

## **conj**

Compute the conjugate of a quaternion.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat conj(
                    Quat_arg quat
                );
            }
        }
    }
}
```

### **Arguments**

*quat*      Quaternion.

### **Return Values**

Conjugate of the specified quaternion

### **Description**

Compute the conjugate of a quaternion.

SCE CONFIDENTIAL

# dot

Compute the dot product of two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 dot(
                    Quat arg quat0,
                    Quat arg quat1
                );
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.

## Return Values

Dot product of the specified quaternions

## Description

Compute the dot product of two quaternions.

SCE CONFIDENTIAL

# **length**

Compute the length of a quaternion.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 length(  

                    Quat_arg quat  

                );
            }
        }
    }
}
```

## **Arguments**

*quat*      Quaternion.

## **Return Values**

Length of the specified quaternion

## **Description**

Compute the length of a quaternion.

SCE CONFIDENTIAL

# **lengthSqr**

Compute the square of the length of a quaternion.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 lengthSqr(
                    Quat arg quat
                );
            }
        }
    }
}
```

## **Arguments**

*quat*                   Quaternion.

## **Return Values**

Square of the length of the specified quaternion.

## **Description**

Compute the square of the length of a quaternion.

# lerp

Linear interpolation between two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat lerp(
                    vec_float4_arg t,
                    Quat_arg quat0,
                    Quat_arg quat1
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.

## Return Values

Interpolated quaternion

## Description

Linearly interpolate between two quaternions.

## Notes

Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

## norm

Compute the norm of a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 norm(
                    Quat arg quat
                );
            }
        }
    }
}
```

### Arguments

*quat*      Quaternion.

### Return Values

The norm of the specified quaternion

### Description

Compute the norm, equal to the square of the length, of a quaternion.

SCE CONFIDENTIAL

# normalize

Normalize a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat normalize(
                    Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

The specified quaternion scaled to unit length

## Description

Compute a normalized quaternion.

## Notes

The result is unpredictable when all elements of quat are at or near zero.

SCE CONFIDENTIAL

# operator\*

Multiply a quaternion by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat operator*(  
                    vec_float4_arg scalar,  
                    Quat_arg quat  
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>quat</i>	Quaternion.

## Return Values

Scalar product of *quat* and *scalar*

## Description

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# print

Print a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

None

## Description

Print a quaternion.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a quaternion and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Quat arg quat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>quat</i>	Quaternion.
<i>name</i>	String printed with the quaternion.

## Return Values

None

## Description

Print a quaternion and an associated string identifier.

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# rotate

Use a unit-length quaternion to rotate a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Vector3 rotate(
                    Quat arg unitQuat,
                    Vector3 arg vec
                );
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
<i>vec</i>	3D vector.

## Return Values

The rotated 3D vector, equivalent to `unitQuat*Quat(vec, 0)*conj(unitQuat)`

## Description

Rotate a 3D vector by applying a unit-length quaternion.

SCE CONFIDENTIAL

# select

Conditionally select between two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat select(
                    Quat arg quat0,
                    Quat arg quat1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	Quaternion.
<i>quat1</i>	Quaternion.
<i>select1</i>	For each of the four word slots, this mask selects either the quaternion in the corresponding slot of <i>quat0</i> or the quaternion in the corresponding slot of <i>quat1</i> . A 0 bit selects from <i>quat0</i> whereas a 1 bit selects from <i>quat1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the quaternion at the corresponding slot of *quat0* or *quat1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *quat0*, and a value of 0xFFFFFFFF selects the slot of *quat1*.

## Description

Conditionally select one of the quaternions at each of the corresponding slots of *quat0* or *quat1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# **slerp**

Spherical linear interpolation between two quaternions.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat slerp(
                    vec_float4_arg t,
                    Quat_arg unitQuat0,
                    Quat_arg unitQuat1,
                    vec_float4_arg tol = SCE_VECTORMATH_DEFAULT_SLERP_TOL_V4
                );
            }
        }
    }
}
```

## **Arguments**

<i>t</i>	Interpolation parameter.
<i>unitQuat0</i>	Quaternion, expected to be unit-length.
<i>unitQuat1</i>	Quaternion, expected to be unit-length.
<i>tol</i>	A tolerance value for detection of co-linear vectors.

## **Return Values**

Interpolated quaternion

## **Description**

Perform spherical linear interpolation between two quaternions.

## **Notes**

Interpolates along the shortest path between orientations. Does not clamp *t* between 0 and 1.

SCE CONFIDENTIAL

# squad

Spherical quadrangle interpolation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Quat squad(
                    vec_float4_arg t,
                    Quat_arg unitQuat0,
                    Quat_arg unitQuat1,
                    Quat_arg unitQuat2,
                    Quat_arg unitQuat3
                );
            }
        }
    }
}
```

## Arguments

<i>t</i>	Interpolation parameter.
<i>unitQuat0</i>	Quaternion, expected to be unit-length.
<i>unitQuat1</i>	Quaternion, expected to be unit-length.
<i>unitQuat2</i>	Quaternion, expected to be unit-length.
<i>unitQuat3</i>	Quaternion, expected to be unit-length.

## Return Values

Interpolated quaternion

## Description

Perform spherical quadrangle interpolation between four quaternions.

# 2x2 Matrix Functions

## absPerElem

Compute the absolute value of a 2x2 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 absPerElem(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat*                  2x2 matrix.

### Return Values

2x2 matrix in which each element is the absolute value of the corresponding element of the specified 2x2 matrix

### Description

Compute the absolute value of each element of a 2x2 matrix.

# appendScale

Append (post-multiply) a scale transformation to a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 appendScale(
                    Matrix2 arg mat,
                    Vector2 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	2x2 matrix.
<i>scaleVec</i>	2D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 2x2 matrix by a scale transformation whose diagonal scale factors are contained in the 2D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# determinant

Determinant of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 determinant(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 2x2 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 inverse(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      2x2 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 2x2 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# mulPerElem

Multiply two 2x2 matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 mulPerElem(
                    Matrix2 arg mat0,
                    Matrix2 arg mat1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	2x2 matrix.
<i>mat1</i>	2x2 matrix.

## Return Values

2x2 matrix in which each element is the product of the corresponding elements of the specified 2x2 matrices

## Description

Multiply two 2x2 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 2x2 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 operator*(
                    vec_float4_arg scalar,
                    Matrix2_arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	2x2 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 2x2 matrix by a scalar.

# prependScale

Prepend (pre-multiply) a scale transformation to a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 prependScale(
                    Vector2 arg scaleVec,
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	2D vector.
<i>mat</i>	2x2 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 2x2 matrix by a scale transformation whose diagonal scale factors are contained in the 2D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      2x2 matrix.

## Return Values

None

## Description

Print a 2x2 matrix. Unlike the printing of vectors, the 2x2 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 2x2 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix2 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	2x2 matrix.
<i>name</i>	String printed with the 2x2 matrix.

## Return Values

None

## Description

Print a 2x2 matrix and an associated string identifier. Unlike the printing of vectors, the 2x2 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 select(
                    Matrix2 arg mat0,
                    Matrix2 arg mat1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	2x2 matrix.
<i>mat1</i>	2x2 matrix.
<i>select1</i>	For each of the four word slots, this mask selects either the 2x2 matrix in the corresponding slot of <i>mat0</i> or the 2x2 matrix in the corresponding slot of <i>mat1</i> . A 0 bit selects from <i>mat0</i> whereas a 1 bit selects from <i>mat1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 2x2 matrix at the corresponding slot of *mat0* or *mat1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *mat0* and a value of 0xFFFFFFFF selects the slot of *mat1*.

## Description

Conditionally select one of the 2x2 matrices at each of the corresponding slots of *mat0* or *mat1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix2 transpose(
                    Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 2x2 matrix.

# 3x3 Matrix Functions

## absPerElem

Compute the absolute value of a 3x3 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 absPerElem(
                    Matrix3_arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat*                    3x3 matrix.

### Return Values

3x3 matrix in which each element is the absolute value of the corresponding element of the specified 3x3 matrix

### Description

Compute the absolute value of each element of a 3x3 matrix.

# appendScale

Append (post-multiply) a scale transformation to a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 appendScale(
                    Matrix3 arg mat,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 3x3 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# determinant

Determinant of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 determinant(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 3x3 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 inverse(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 3x3 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# mulPerElem

Multiply two 3x3 matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 mulPerElem(
                    Matrix3 arg mat0,
                    Matrix3 arg mat1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	3x3 matrix.
<i>mat1</i>	3x3 matrix.

## Return Values

3x3 matrix in which each element is the product of the corresponding elements of the specified 3x3 matrices

## Description

Multiply two 3x3 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 3x3 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 operator*(
                    vec_float4 arg scalar,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	3x3 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 3x3 matrix by a scalar.

# prependScale

Prepend (pre-multiply) a scale transformation to a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 prependScale(
                    Vector3 arg scaleVec,
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>mat</i>	3x3 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 3x3 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      3x3 matrix.

## Return Values

None

## Description

Print a 3x3 matrix. Unlike the printing of vectors, the 3x3 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# print

Print a 3x3 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix3 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>name</i>	String printed with the 3x3 matrix.

## Return Values

None

## Description

Print a 3x3 matrix and an associated string identifier. Unlike the printing of vectors, the 3x3 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 select(
                    Matrix3 arg mat0,
                    Matrix3 arg mat1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	3x3 matrix.
<i>mat1</i>	3x3 matrix.
<i>select1</i>	For each of the four word slots, this mask selects either the 3x3 matrix in the corresponding slot of <i>mat0</i> or the 3x3 matrix in the corresponding slot of <i>mat1</i> . A 0 bit selects from <i>mat0</i> whereas a 1 bit selects from <i>mat1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 3x3 matrix at the corresponding slot of *mat0* or *mat1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *mat0* and a value of 0xFFFFFFFF selects the slot of *mat1*.

## Description

Conditionally select one of the 3x3 matrices at each of the corresponding slots of *mat0* or *mat1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix3 transpose(
                    Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 3x3 matrix.

# 4x4 Matrix Functions

## absPerElem

Compute the absolute value of a 4x4 matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 absPerElem(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

### Arguments

*mat*                  4x4 matrix.

### Return Values

4x4 matrix in which each element is the absolute value of the corresponding element of the specified 4x4 matrix

### Description

Compute the absolute value of each element of a 4x4 matrix.

# affineInverse

Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 affineInverse(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Inverse of the specified 4x4 matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 4x4 matrix as *M*, and its translation component as *v*, compute a matrix whose upper-left 3x3 submatrix is `inverse(M)`, whose translation vector is `-inverse(M) * v`, and whose bottom row is (0,0,0,1).

## Notes

This can be used to achieve better performance than a general inverse when the specified 4x4 matrix meets the given restrictions. The result is unpredictable when the determinant of *mat* is equal to or near 0.

# appendScale

Append (post-multiply) a scale transformation to a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 appendScale(
                    Matrix4 arg mat,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	4x4 matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *mat* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 4x4 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# determinant

Determinant of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE vec_float4 determinant(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

The determinant of *mat*

## Description

Compute the determinant of a 4x4 matrix.

SCE CONFIDENTIAL

# inverse

Compute the inverse of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 inverse(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      4x4 matrix.

## Return Values

Inverse of *mat*

## Description

Compute the inverse of a 4x4 matrix.

## Notes

Result is unpredictable when the determinant of *mat* is equal to or near 0.

# mulPerElem

Multiply two 4x4 matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 mulPerElem(
                    Matrix4 arg mat0,
                    Matrix4 arg mat1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	4x4 matrix.
<i>mat1</i>	4x4 matrix.

## Return Values

4x4 matrix in which each element is the product of the corresponding elements of the specified 4x4 matrices

## Description

Multiply two 4x4 matrices element by element.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 operator*(
                    vec_float4 arg scalar,
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
<i>mat</i>	4x4 matrix.

## Return Values

Scalar product of *mat* and *scalar*

## Description

Multiply a 4x4 matrix by a scalar.

# orthoInverse

Compute the inverse of a 4x4 matrix, which is expected to be an affine matrix with an orthogonal upper-left 3x3 submatrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 orthoInverse(
                    Matrix4_arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Inverse of the specified 4x4 matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 4x4 matrix as  $M$ , and its translation component as  $v$ , compute a matrix whose upper-left 3x3 submatrix is  $\text{transpose}(M)$ , whose translation vector is  $-\text{transpose}(M) * v$ , and whose bottom row is  $(0,0,0,1)$ .

## Notes

This can be used to achieve better performance than a general inverse when the specified 4x4 matrix meets the given restrictions.

# prependScale

Prepend (pre-multiply) a scale transformation to a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 prependScale(
                    Vector3 arg scaleVec,
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>mat</i>	4x4 matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *mat*

## Description

Pre-multiply a 4x4 matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*      4x4 matrix.

## Return Values

None

## Description

Print a 4x4 matrix. Unlike the printing of vectors, the 4x4 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# print

Print a 4x4 matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Matrix4 arg mat,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>mat</i>	4x4 matrix.
<i>name</i>	String printed with the 4x4 matrix.

## Return Values

None

## Description

Print a 4x4 matrix and an associated string identifier. Unlike the printing of vectors, the 4x4 matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# select

Conditionally select between two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 select(
                    Matrix4 arg mat0,
                    Matrix4 arg mat1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	4x4 matrix.
<i>mat1</i>	4x4 matrix.
<i>select1</i>	For each of the four word slots, this mask selects either the 4x4 matrix in the corresponding slot of <i>mat0</i> or the 4x4 matrix in the corresponding slot of <i>mat1</i> . A 0 bit selects from <i>mat0</i> whereas a 1 bit selects from <i>mat1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 4x4 matrix at the corresponding slot of *mat0* or *mat1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *mat0* and a value of 0xFFFFFFFF selects the slot of *mat1*.

## Description

Conditionally select one of the 4x4 matrices at each of the corresponding slots of *mat0* or *mat1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

# transpose

Transpose of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Matrix4 transpose(
                    Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

*mat* transposed

## Description

Compute the transpose of a 4x4 matrix.

# 3x4 Transformation Matrix Functions

## absPerElem

Compute the absolute value of a 3x4 transformation matrix per element.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 absPerElem(
                    Transform3 arg
                );
            }
        }
    }
}
```

### Arguments

*tfrm*                  3x4 transformation matrix.

### Return Values

3x4 transformation matrix in which each element is the absolute value of the corresponding element of the specified 3x4 transformation matrix

### Description

Compute the absolute value of each element of a 3x4 transformation matrix.

# appendScale

Append (post-multiply) a scale transformation to a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 appendScale(
                    Transform3 arg tfrm,
                    Vector3 arg scaleVec
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x4 transformation matrix.
<i>scaleVec</i>	3D vector.

## Return Values

The product of *tfrm* and a scale transformation created from *scaleVec*

## Description

Post-multiply a 3x4 transformation matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

# inverse

Inverse of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 inverse(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

Inverse of *tfrm*

## Description

Compute the inverse of a 3x4 transformation matrix.

## Notes

Result is unpredictable when the determinant of the left 3x3 submatrix is equal to or near 0.

# mulPerElem

Multiply two 3x4 transformation matrices per element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 mulPerElem(
                    Transform3 arg tfrm0,
                    Transform3 arg tfrm1
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm0</i>	3x4 transformation matrix.
<i>tfrm1</i>	3x4 transformation matrix.

## Return Values

3x4 transformation matrix in which each element is the product of the corresponding elements of the specified 3x4 transformation matrices

## Description

Multiply two 3x4 transformation matrices element by element.

# ortholInverse

Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 orthoInverse(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*                   3x4 transformation matrix.

## Return Values

Inverse of the specified 3x4 transformation matrix

## Description

Naming the upper-left 3x3 submatrix of the specified 3x4 transformation matrix as  $M$ , and its translation component as  $v$ , compute a matrix whose upper-left 3x3 submatrix is  $\text{transpose}(M)$ , and whose translation vector is  $-\text{transpose}(M) * v$ .

## Notes

This can be used to achieve better performance than a general inverse when the specified 3x4 transformation matrix meets the given restrictions.

# prependScale

Prepend (pre-multiply) a scale transformation to a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 prependScale(
                    Vector3 arg scaleVec,
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

<i>scaleVec</i>	3D vector.
<i>tfrm</i>	3x4 transformation matrix.

## Return Values

The product of a scale transformation created from *scaleVec* and *tfrm*

## Description

Pre-multiply a 3x4 transformation matrix by a scale transformation whose diagonal scale factors are contained in the 3D vector.

## Notes

Faster than creating and multiplying a scale transformation matrix.

SCE CONFIDENTIAL

# print

Print a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Transform3 arg tfrm
                );
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

None

## Description

Print a 3x4 transformation matrix. Unlike the printing of vectors, the 3x4 transformation matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

SCE CONFIDENTIAL

# print

Print a 3x4 transformation matrix and an associated string identifier.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE void print(
                    Transform3 arg tfrm,
                    const char *name
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x4 transformation matrix.
<i>name</i>	String printed with the 3x4 transformation matrix.

## Return Values

None

## Description

Print a 3x4 transformation matrix and an associated string identifier. Unlike the printing of vectors, the 3x4 transformation matrix is printed with the correct orientation (columns appear vertically).

## Notes

Function is only defined when `SCE_VECTORMATH_DEBUG` is defined.

# select

Conditionally select between two 3x4 transformation matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                SCE_VECTORMATH_INLINE const Transform3 select(
                    Transform3 arg tfrm0,
                    Transform3 arg tfrm1,
                    vec_uint4_arg select1
                );
            }
        }
    }
}
```

## Arguments

<i>tfrm0</i>	3x4 transformation matrix.
<i>tfrm1</i>	3x4 transformation matrix.
<i>select1</i>	For each of the four word slots, this mask selects either the 3x4 transformation matrix in the corresponding slot of <i>tfrm0</i> or the 3x4 transformation matrix in the corresponding slot of <i>tfrm1</i> . A 0 bit selects from <i>tfrm0</i> whereas a 1 bit selects from <i>tfrm1</i> . Identical bits should be set for each word of the mask.

## Return Values

Each slot of the result is equal to the 3x4 transformation matrix at the corresponding slot of *tfrm0* or *tfrm1*, depending on the value of *select1* at the corresponding slot. A value of 0 selects the slot of *tfrm0* and a value of 0xFFFFFFFF selects the slot of *tfrm1*.

## Description

Conditionally select one of the 3x4 transformation matrices at each of the corresponding slots of *tfrm0* or *tfrm1*.

## Notes

This function uses a conditional select instruction to avoid a branch.

SCE CONFIDENTIAL

---

**sce::Vectormath::Simd::Soa::Matrix2**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Matrix2

A set of four 2x2 matrices in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Matrix2 {};
```

### Description

A class representing a set of four 2x2 matrices stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 2x2 matrices.
<a href="#">getCol</a>	Get the column of a 2x2 matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 2x2 matrix.
<a href="#">getCol1</a>	Get column 1 of a 2x2 matrix.
<a href="#">getElem</a>	Get the element of a 2x2 matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 2x2 matrix referred to by the specified index.
<a href="#">identity</a>	Construct an identity 2x2 matrix.
<a href="#">Matrix2</a>	Default constructor; does no initialization.
<a href="#">Matrix2</a>	Copy constructor.
<a href="#">Matrix2</a>	Construct a 2x2 matrix containing the specified columns.
<a href="#">Matrix2</a>	Set all elements of a 2x2 matrix to the same scalar value.
<a href="#">Matrix2</a>	Replicate an AoS 2x2 matrix.
<a href="#">Matrix2</a>	Insert four AoS 2x2 matrices.
<a href="#">operator*</a>	Multiply a 2x2 matrix by a scalar.
<a href="#">operator*</a>	Multiply a 2x2 matrix by a 2D vector.
<a href="#">operator*</a>	Multiply two 2x2 matrices.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 2x2 matrix.
<a href="#">operator+</a>	Add two 2x2 matrices.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 2x2 matrix.
<a href="#">operator-</a>	Subtract a 2x2 matrix from another 2x2 matrix.
<a href="#">operator-</a>	Negate all elements of a 2x2 matrix.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 2x2 matrix.
<a href="#">operator=</a>	Assign one 2x2 matrix to another.
<a href="#">operator[]</a>	Subscripting operator to set or get a column.
<a href="#">operator[]</a>	Subscripting operator to get a column.
<a href="#">rotation</a>	Construct a 2x2 matrix to rotate.
<a href="#">scale</a>	Construct a 2x2 matrix to perform scaling.
<a href="#">setCol</a>	Set the column of a 2x2 matrix referred to by the specified index.
<a href="#">setCol0</a>	Set column 0 of a 2x2 matrix.
<a href="#">setCol1</a>	Set column 1 of a 2x2 matrix.
<a href="#">setElem</a>	Set the element of a 2x2 matrix referred to by column and row indices.
<a href="#">setRow</a>	Set the row of a 2x2 matrix referred to by the specified index.
<a href="#">zero</a>	Construct a zero 2x2 matrix.

# Constructors and Destructors

## Matrix2

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix2

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix2(
                        const Matrix2 &mat
                    );
                };
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix2

Construct a 2x2 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2(
                        Vector2 arg col0,
                        Vector2 arg col1
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	2D vector.
<i>col1</i>	2D vector.

## Return Values

None

## Description

Construct a 2x2 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix2

Set all elements of a 2x2 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    explicit SCE_VECTORMATH_INLINE Matrix2(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 2x2 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix2

Replicate an AoS 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2(
                        Aos::Matrix2 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                      AoS 2x2 matrix.

## Return Values

None

## Description

Replicate an AoS 2x2 matrix in all four slots of an SoA 2x2 matrix.

SCE CONFIDENTIAL

# Matrix2

Insert four AoS 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2(
                        Aos::Matrix2 arg mat0,
                        Aos::Matrix2 arg mat1,
                        Aos::Matrix2 arg mat2,
                        Aos::Matrix2 arg mat3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	AoS 2x2 matrix.
<i>mat1</i>	AoS 2x2 matrix.
<i>mat2</i>	AoS 2x2 matrix.
<i>mat3</i>	AoS 2x2 matrix.

## Return Values

None

## Description

Insert four AoS 2x2 matrices into four slots of an SoA 2x2 matrix (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 2x2 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*              Scalar value.

### **Return Values**

Product of the specified 2x2 matrix and scalar

### **Description**

Multiply a 2x2 matrix by a scalar.

SCE CONFIDENTIAL

# operator\*

Multiply a 2x2 matrix by a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator*(
                        Vector2 arg
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

Product of the specified 2x2 matrix and 2D vector

## Description

Multiply a 2x2 matrix by a 2D vector.

SCE CONFIDENTIAL

# operator\*

Multiply two 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator*(
                        Matrix2 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

Product of the specified 2x2 matrices

## Description

Multiply two 2x2 matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator\*= ---

Perform compound assignment and multiplication by a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator*=(  

                        Matrix2 arg mat  

                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Perform compound assignment and multiplication by a 2x2 matrix.

SCE CONFIDENTIAL

# operator+

Add two 2x2 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator+
                        (Matrix2 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

Sum of the specified 2x2 matrices

## Description

Add two 2x2 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator+=(  
                        Matrix2 arg mat  
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Perform compound assignment and addition with a 2x2 matrix.

SCE CONFIDENTIAL

# **operator-**

Subtract a 2x2 matrix from another 2x2 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator-
                        (Matrix2 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

Difference of the specified 2x2 matrices

## **Description**

Subtract a 2x2 matrix from another 2x2 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 2x2 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Matrix2 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

2x2 matrix containing negated elements of the specified 2x2 matrix

## **Description**

Negate all elements of a 2x2 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator==(Matrix2 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                  2x2 matrix.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Perform compound assignment and subtraction by a 2x2 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 2x2 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &operator=(  

                        Matrix2 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  2x2 matrix.

## **Return Values**

A reference to the resulting 2x2 matrix

## **Description**

Assign one 2x2 matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-1.

## **Return Values**

A reference to the indexed column

## **Description**

Subscripting operator invoked when applied to non-const [Matrix2](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-1.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Matrix2](#).

# Public Static Methods

## identity

Construct an identity 2x2 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2x2 matrix

### Description

Construct an identity 2x2 matrix in which non-diagonal elements are zero and diagonal elements are 1.

SCE CONFIDENTIAL

# rotation

Construct a 2x2 matrix to rotate.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 rotation(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 2x2 matrix

## Description

Construct a 2x2 matrix to rotate by the specified radians angle.

SCE CONFIDENTIAL

# scale

Construct a 2x2 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 scale(
                        Vector2 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      2D vector.

## Return Values

The constructed 2x2 matrix

## Description

Construct a 2x2 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

## zero

Construct a zero 2x2 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    static SCE_VECTORMATH_INLINE const Matrix2 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2x2 matrix

### Description

Construct a zero 2x2 matrix in which all elements are zero.

# Public Instance Methods

## get4Aos

Extract four AoS 2x2 matrices.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Matrix2 &result0,
                        Aos::Matrix2 &result1,
                        Aos::Matrix2 &result2,
                        Aos::Matrix2 &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 2x2 matrix.
<i>result1</i>	An output AoS 2x2 matrix.
<i>result2</i>	An output AoS 2x2 matrix.
<i>result3</i>	An output AoS 2x2 matrix.

### Return Values

None

### Description

Extract four AoS 2x2 matrices from four slots of an SoA 2x2 matrix (transpose the data format).

SCE CONFIDENTIAL

# getCol

Get the column of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-1.

## Return Values

The column referred to by the specified index

## Description

Get the column of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 2x2 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 2x2 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 2x2 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>row</i>	Index, expected in the range 0-1.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 2x2 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE const Vector2 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-1.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol

Set the column of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setCol(
                        int col,
                        Vector2 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>vec</i>	2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the column of a 2x2 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setCol0(
                        Vector2 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set column 0 of a 2x2 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 2x2 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setCol1(
                        Vector2 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set column 1 of a 2x2 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 2x2 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setElem(
                        int col,
                        int row,
                        vec_float4_arg val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-1.
<i>row</i>	Index, expected in the range 0-1.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the element of a 2x2 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 2x2 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix2 {
                    SCE_VECTORMATH_INLINE Matrix2 &setRow(
                        int row,
                        Vector2 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-1.
<i>vec</i>	2D vector.

## Return Values

A reference to the resulting 2x2 matrix

## Description

Set the row of a 2x2 matrix referred to by the specified index.

**sce::Vectormath::Simd::Soa::Matrix3**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Matrix3

A set of four 3x3 matrices in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Matrix3 {};
```

### Description

A class representing a set of four 3x3 matrices stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#"><u>get4AoS</u></a>	Extract four AoS 3x3 matrices.
<a href="#"><u>getCol</u></a>	Get the column of a 3x3 matrix referred to by the specified index.
<a href="#"><u>getCol0</u></a>	Get column 0 of a 3x3 matrix.
<a href="#"><u>getCol1</u></a>	Get column 1 of a 3x3 matrix.
<a href="#"><u>getCol2</u></a>	Get column 2 of a 3x3 matrix.
<a href="#"><u>getElem</u></a>	Get the element of a 3x3 matrix referred to by column and row indices.
<a href="#"><u>getRow</u></a>	Get the row of a 3x3 matrix referred to by the specified index.
<a href="#"><u>identity</u></a>	Construct an identity 3x3 matrix.
<a href="#"><u>Matrix3</u></a>	Default constructor; does no initialization.
<a href="#"><u>Matrix3</u></a>	Copy constructor.
<a href="#"><u>Matrix3</u></a>	Construct a 3x3 matrix containing the specified columns.
<a href="#"><u>Matrix3</u></a>	Construct a 3x3 rotation matrix from a unit-length quaternion.
<a href="#"><u>Matrix3</u></a>	Set all elements of a 3x3 matrix to the same scalar value.
<a href="#"><u>Matrix3</u></a>	Replicate an AoS 3x3 matrix.
<a href="#"><u>Matrix3</u></a>	Insert four AoS 3x3 matrices.
<a href="#"><u>operator*</u></a>	Multiply a 3x3 matrix by a scalar.
<a href="#"><u>operator*</u></a>	Multiply a 3x3 matrix by a 3D vector.
<a href="#"><u>operator*</u></a>	Multiply two 3x3 matrices.
<a href="#"><u>operator*=</u></a>	Perform compound assignment and multiplication by a scalar.
<a href="#"><u>operator*=</u></a>	Perform compound assignment and multiplication by a 3x3 matrix.
<a href="#"><u>operator+</u></a>	Add two 3x3 matrices.
<a href="#"><u>operator+=</u></a>	Perform compound assignment and addition with a 3x3 matrix.
<a href="#"><u>operator-</u></a>	Subtract a 3x3 matrix from another 3x3 matrix.
<a href="#"><u>operator-</u></a>	Negate all elements of a 3x3 matrix.
<a href="#"><u>operator-=</u></a>	Perform compound assignment and subtraction by a 3x3 matrix.
<a href="#"><u>operator=</u></a>	Assign one 3x3 matrix to another.
<a href="#"><u>operator[]</u></a>	Subscripting operator to set or get a column.
<a href="#"><u>operator[]</u></a>	Subscripting operator to get a column.
<a href="#"><u>rotation</u></a>	Construct a 3x3 matrix to rotate around a unit-length 3D vector.
<a href="#"><u>rotation</u></a>	Construct a rotation matrix from a unit-length quaternion.
<a href="#"><u>rotationX</u></a>	Construct a 3x3 matrix to rotate around the x axis.
<a href="#"><u>rotationY</u></a>	Construct a 3x3 matrix to rotate around the y axis.
<a href="#"><u>rotationZ</u></a>	Construct a 3x3 matrix to rotate around the z axis.
<a href="#"><u>rotationZYX</u></a>	Construct a 3x3 matrix to rotate around the x, y, and z axes.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>scale</u></a>	Construct a 3x3 matrix to perform scaling.
<a href="#"><u>setCol</u></a>	Set the column of a 3x3 matrix referred to by the specified index.
<a href="#"><u>setCol0</u></a>	Set column 0 of a 3x3 matrix.
<a href="#"><u>setCol1</u></a>	Set column 1 of a 3x3 matrix.
<a href="#"><u>setCol2</u></a>	Set column 2 of a 3x3 matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 3x3 matrix referred to by column and row indices.
<a href="#"><u>setRow</u></a>	Set the row of a 3x3 matrix referred to by the specified index.
<a href="#"><u>zero</u></a>	Construct a zero 3x3 matrix.

000004892117

# Constructors and Destructors

## Matrix3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix3(
                        const Matrix3 &mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix3

Construct a 3x3 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3(
                        Vector3 arg col0,
                        Vector3 arg col1,
                        Vector3 arg col2
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	3D vector.
<i>col1</i>	3D vector.
<i>col2</i>	3D vector.

## Return Values

None

## Description

Construct a 3x3 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix3

Construct a 3x3 rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    explicit SCE_VECTORMATH_INLINE Matrix3(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

None

## Description

Construct a 3x3 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# Matrix3

Set all elements of a 3x3 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    explicit SCE_VECTORMATH_INLINE Matrix3(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 3x3 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix3

## Replicate an AoS 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3(
                        Aos::Matrix3 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat* AoS 3x3 matrix.

## Return Values

None

## Description

Replicate an AoS 3x3 matrix in all four slots of an SoA 3x3 matrix.

SCE CONFIDENTIAL

# Matrix3

Insert four AoS 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3(
                        Aos::Matrix3 arg mat0,
                        Aos::Matrix3 arg mat1,
                        Aos::Matrix3 arg mat2,
                        Aos::Matrix3 arg mat3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	AoS 3x3 matrix.
<i>mat1</i>	AoS 3x3 matrix.
<i>mat2</i>	AoS 3x3 matrix.
<i>mat3</i>	AoS 3x3 matrix.

## Return Values

None

## Description

Insert four AoS 3x3 matrices into four slots of an SoA 3x3 matrix (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 3x3 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 3x3 matrix and scalar

### **Description**

Multiply a 3x3 matrix by a scalar.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x3 matrix by a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        Vector3 arg
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

vec                   3D vector.

## **Return Values**

Product of the specified 3x3 matrix and 3D vector

## **Description**

Multiply a 3x3 matrix by a 3D vector.

SCE CONFIDENTIAL

# operator\*

Multiply two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator*(
                        Matrix3 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Product of the specified 3x3 matrices

## Description

Multiply two 3x3 matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator*=(  

                        Matrix3 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Perform compound assignment and multiplication by a 3x3 matrix.

SCE CONFIDENTIAL

# operator+

Add two 3x3 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator+
                        (Matrix3 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

Sum of the specified 3x3 matrices

## Description

Add two 3x3 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator+=(
                        Matrix3 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Perform compound assignment and addition with a 3x3 matrix.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3x3 matrix from another 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator-
                        (Matrix3 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

Difference of the specified 3x3 matrices

## **Description**

Subtract a 3x3 matrix from another 3x3 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 3x3 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Matrix3 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

3x3 matrix containing negated elements of the specified 3x3 matrix

## **Description**

Negate all elements of a 3x3 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator==(Matrix3 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat*                    3x3 matrix.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Perform compound assignment and subtraction by a 3x3 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 3x3 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &operator=(  

                        Matrix3 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                    3x3 matrix.

## **Return Values**

A reference to the resulting 3x3 matrix

## **Description**

Assign one 3x3 matrix to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-2.

## Return Values

A reference to the indexed column

## Description

Subscripting operator invoked when applied to non-const [Matrix3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-2.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Matrix3](#).

# Public Static Methods

## identity

Construct an identity 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x3 matrix

### Description

Construct an identity 3x3 matrix in which non-diagonal elements are zero and diagonal elements are 1.

# rotation

Construct a 3x3 matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotation(
                        vec_float4_arg radians,
                        Vector3_arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 3x3 matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationX(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 3x3 matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationY(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 3x3 matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationZ(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 3x3 matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                }
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 3x3 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      3D vector.

## Return Values

The constructed 3x3 matrix

## Description

Construct a 3x3 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

## zero

Construct a zero 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    static SCE_VECTORMATH_INLINE const Matrix3 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x3 matrix

### Description

Construct a zero 3x3 matrix in which all elements are zero.

# Public Instance Methods

## get4Aos

Extract four AoS 3x3 matrices.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Matrix3 &result0,
                        Aos::Matrix3 &result1,
                        Aos::Matrix3 &result2,
                        Aos::Matrix3 &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 3x3 matrix.
<i>result1</i>	An output AoS 3x3 matrix.
<i>result2</i>	An output AoS 3x3 matrix.
<i>result3</i>	An output AoS 3x3 matrix.

### Return Values

None

### Description

Extract four AoS 3x3 matrices from four slots of an SoA 3x3 matrix (transpose the data format).

SCE CONFIDENTIAL

# getCol

Get the column of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-2.

## Return Values

The column referred to by the specified index

## Description

Get the column of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 3x3 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 3x3 matrix.

SCE CONFIDENTIAL

## getCol2

Get column 2 of a 3x3 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol2() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Column 2

### Description

Get column 2 of a 3x3 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 3x3 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>row</i>	Index, expected in the range 0-2.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 3x3 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE const Vector3 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-2.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol

Set the column of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setCol(
                        int col,
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the column of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setCol0(
                        Vector3 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 0 of a 3x3 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setCol1(
                        Vector3 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 1 of a 3x3 matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 3x3 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setCol2(
                        Vector3 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set column 2 of a 3x3 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 3x3 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setElem(
                        int col,
                        int row,
                        vec_float4_arg val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-2.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the element of a 3x3 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 3x3 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix3 {
                    SCE_VECTORMATH_INLINE Matrix3 &setRow(
                        int row,
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-2.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x3 matrix

## Description

Set the row of a 3x3 matrix referred to by the specified index.

SCE CONFIDENTIAL

---

**sce::Vectormath::Simd::Soa::Matrix4**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Matrix4

A set of four 4x4 matrices in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Matrix4 {};
```

### Description

A class representing a set of four 4x4 matrices stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">frustum</a>	Construct a perspective projection matrix based on frustum.
<a href="#">get4Aos</a>	Extract four AoS 4x4 matrices.
<a href="#">getCol</a>	Get the column of a 4x4 matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 4x4 matrix.
<a href="#">getCol1</a>	Get column 1 of a 4x4 matrix.
<a href="#">getCol2</a>	Get column 2 of a 4x4 matrix.
<a href="#">getCol3</a>	Get column 3 of a 4x4 matrix.
<a href="#">getElem</a>	Get the element of a 4x4 matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 4x4 matrix referred to by the specified index.
<a href="#">getTranslation</a>	Get the translation component of a 4x4 matrix.
<a href="#">getUpper3x3</a>	Get the upper-left 3x3 submatrix of a 4x4 matrix.
<a href="#">identity</a>	Construct an identity 4x4 matrix.
<a href="#">lookAt</a>	Construct viewing matrix based on eye position, position looked at, and up direction.
<a href="#">Matrix4</a>	Default constructor; does no initialization.
<a href="#">Matrix4</a>	Copy constructor.
<a href="#">Matrix4</a>	Construct a 4x4 matrix containing the specified columns.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a 3x4 transformation matrix.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a 3x3 matrix and a 3D vector.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a unit-length quaternion and a 3D vector.
<a href="#">Matrix4</a>	Set all elements of a 4x4 matrix to the same scalar value.
<a href="#">Matrix4</a>	Replicate an AoS 4x4 matrix.
<a href="#">Matrix4</a>	Insert four AoS 4x4 matrices.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a scalar.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 4D vector.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3D vector.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3D point.
<a href="#">operator*</a>	Multiply two 4x4 matrices.
<a href="#">operator*</a>	Multiply a 4x4 matrix by a 3x4 transformation matrix.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 4x4 matrix.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 3x4 transformation matrix.
<a href="#">operator+</a>	Add two 4x4 matrices.

SCE CONFIDENTIAL

Methods	Description
<code>operator+=</code>	Perform compound assignment and addition with a 4x4 matrix.
<code>operator-</code>	Subtract a 4x4 matrix from another 4x4 matrix.
<code>operator-</code>	Negate all elements of a 4x4 matrix.
<code>operator-=</code>	Perform compound assignment and subtraction by a 4x4 matrix.
<code>operator=</code>	Assign one 4x4 matrix to another.
<code>operator[]</code>	Subscripting operator to set or get a column.
<code>operator[]</code>	Subscripting operator to get a column.
<code>orthographic</code>	Construct an orthographic projection matrix.
<code>perspective</code>	Construct a perspective projection matrix.
<code>rotation</code>	Construct a 4x4 matrix to rotate around a unit-length 3D vector.
<code>rotation</code>	Construct a rotation matrix from a unit-length quaternion.
<code>rotationX</code>	Construct a 4x4 matrix to rotate around the x axis.
<code>rotationY</code>	Construct a 4x4 matrix to rotate around the y axis.
<code>rotationZ</code>	Construct a 4x4 matrix to rotate around the z axis.
<code>rotationZYX</code>	Construct a 4x4 matrix to rotate around the x, y, and z axes.
<code>scale</code>	Construct a 4x4 matrix to perform scaling.
<code>setCol</code>	Set the column of a 4x4 matrix referred to by the specified index.
<code>setCol0</code>	Set column 0 of a 4x4 matrix.
<code>setCol1</code>	Set column 1 of a 4x4 matrix.
<code>setCol2</code>	Set column 2 of a 4x4 matrix.
<code>setCol3</code>	Set column 3 of a 4x4 matrix.
<code>setElem</code>	Set the element of a 4x4 matrix referred to by column and row indices.
<code>setRow</code>	Set the row of a 4x4 matrix referred to by the specified index.
<code>setTranslation</code>	Set translation component.
<code>setUpper3x3</code>	Set the upper-left 3x3 submatrix.
<code>translation</code>	Construct a 4x4 matrix to perform translation.
<code>zero</code>	Construct an identity 4x4 matrix.

# Constructors and Destructors

## Matrix4

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Matrix4

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Matrix4(
                        const Matrix4 &mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Vector4 arg col0,
                        Vector4 arg col1,
                        Vector4 arg col2,
                        Vector4 arg col3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	4D vector.
<i>col1</i>	4D vector.
<i>col2</i>	4D vector.
<i>col3</i>	4D vector.

## Return Values

None

## Description

Construct a 4x4 matrix containing the specified columns.

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    explicit SCE_VECTORMATH_INLINE Matrix4(
                        Transform3 arg mat
                    );
                };
            }
        }
    }
}
```

## Arguments

*mat*                   3x4 transformation matrix.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper 3x4 elements are equal to the 3x4 transformation matrix argument and whose bottom row is equal to (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a 3x3 matrix and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Matrix3 arg mat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>mat</i>	3x3 matrix.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper 3x3 elements are equal to the 3x3 matrix argument, whose translation component is equal to the 3D vector argument, and whose bottom row is (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Construct a 4x4 matrix from a unit-length quaternion and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Quat arg unitQuat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Quaternion, expected to be unit-length.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 4x4 matrix whose upper-left 3x3 submatrix is a rotation matrix converted from the unit-length quaternion argument, whose translation component is equal to the 3D vector argument, and whose bottom row is (0,0,0,1).

SCE CONFIDENTIAL

# Matrix4

Set all elements of a 4x4 matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    explicit SCE_VECTORMATH_INLINE Matrix4(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 4x4 matrix with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Matrix4

## Replicate an AoS 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Aos::Matrix4 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

## Return Values

None

## Description

Replicate an AoS 4x4 matrix in all four slots of an SoA 4x4 matrix.

SCE CONFIDENTIAL

# Matrix4

Insert four AoS 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4(
                        Aos::Matrix4 arg mat0,
                        Aos::Matrix4 arg mat1,
                        Aos::Matrix4 arg mat2,
                        Aos::Matrix4 arg mat3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>mat0</i>	AoS 4x4 matrix.
<i>mat1</i>	AoS 4x4 matrix.
<i>mat2</i>	AoS 4x4 matrix.
<i>mat3</i>	AoS 4x4 matrix.

## Return Values

None

## Description

Insert four AoS 4x4 matrices into four slots of an SoA 4x4 matrix (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 4x4 matrix by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 4x4 matrix and scalar

### **Description**

Multiply a 4x4 matrix by a scalar.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Vector4 arg
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Product of the specified 4x4 matrix and 4D vector

## Description

Multiply a 4x4 matrix by a 4D vector.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Vector3 arg vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Product of the specified 4x4 matrix and 3D vector

## Description

Multiply a 4x4 matrix by a 3D vector treated as if it were a 4D vector with the *w* element equal to 0.

SCE CONFIDENTIAL

# **operator\***

---

Multiply a 4x4 matrix by a 3D point.

## **Definition**

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        Point3 arg pnt
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

---

*pnt*                    3D point.

## **Return Values**

---

Product of the specified 4x4 matrix and 3D point

## **Description**

---

Multiply a 4x4 matrix by a 3D point treated as if it were a 4D vector with the *w* element equal to 1.

SCE CONFIDENTIAL

# operator\*

Multiply two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        Matrix4 arg mat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat*                  4x4 matrix.

## Return Values

Product of the specified 4x4 matrices

## Description

Multiply two 4x4 matrices.

SCE CONFIDENTIAL

# operator\*

Multiply a 4x4 matrix by a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator*(
                        Transform3 arg tfrm
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

Product of the specified 4x4 matrix and 3x4 transformation matrix

## Description

Multiply a 4x4 matrix by a 3x4 transformation matrix treated as if it were a 4x4 matrix with the bottom row equal to (0,0,0,1).

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 4x4 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        Matrix4 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  4x4 matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Perform compound assignment and multiplication by a 4x4 matrix.

SCE CONFIDENTIAL

# operator\*= ---

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator*=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Perform compound assignment and multiplication by a 3x4 transformation matrix.

SCE CONFIDENTIAL

# operator+

Add two 4x4 matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator+
                        (Matrix4 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

Sum of the specified 4x4 matrices

## Description

Add two 4x4 matrices.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator+=(
                        Matrix4 arg mat
                    );
                }
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Perform compound assignment and addition with a 4x4 matrix.

SCE CONFIDENTIAL

# operator-

Subtract a 4x4 matrix from another 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator-
                        (Matrix4 arg mat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

Difference of the specified 4x4 matrices

## Description

Subtract a 4x4 matrix from another 4x4 matrix.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 4x4 matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix4 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

4x4 matrix containing negated elements of the specified 4x4 matrix

## **Description**

Negate all elements of a 4x4 matrix.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator==(Matrix4 arg mat
                );
            }
        }
    }
}
```

## Arguments

*mat* 4x4 matrix.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Perform compound assignment and subtraction by a 4x4 matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 4x4 matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &operator=(  

                        Matrix4 arg mat  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat*                  4x4 matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Assign one 4x4 matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-3.

## **Return Values**

A reference to the indexed column

## **Description**

Subscripting operator invoked when applied to non-const [Matrix4](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Matrix4](#).

# Public Static Methods

## frustum

Construct a perspective projection matrix based on frustum.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 frustum(
                        vec_float4_arg left,
                        vec_float4_arg right,
                        vec_float4_arg bottom,
                        vec_float4_arg top,
                        vec_float4_arg zNear,
                        vec_float4_arg zFar
                    );
                };
            }
        }
    }
}
```

### Arguments

<i>left</i>	Scalar value.
<i>right</i>	Scalar value.
<i>bottom</i>	Scalar value.
<i>top</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

### Return Values

The constructed 4x4 matrix

### Description

Construct a perspective projection matrix based on frustum, equal to:

$2*n/(r-l)$	0	$(r+l)/(r-l)$	0
0	$2*n/(t-b)$	$(t+b)/(t-b)$	0
0	0	$-(f+n)/(f-n)$	$-2*f*n/(f-n)$
0	0	-1	0

*; l = left*  
*; r = right*  
*; b = bottom*  
*; t = top*  
*; n = zNear*  
*; f = zFar*

SCE CONFIDENTIAL

# identity

Construct an identity 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 identity();
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4x4 matrix

## Description

Construct an identity 4x4 matrix in which non-diagonal elements are zero and diagonal elements are 1.

SCE CONFIDENTIAL

# lookAt

Construct viewing matrix based on eye position, position looked at, and up direction.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 lookAt(
                        Point3_arg eyePos,
                        Point3_arg lookAtPos,
                        Vector3_arg upVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>eyePos</i>	3D point.
<i>lookAtPos</i>	3D point.
<i>upVec</i>	3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct the inverse of a coordinate frame that is centered at the eye position, with z axis directed away from *lookAtPos*, and y axis oriented to best match the up direction.

SCE CONFIDENTIAL

# orthographic

Construct an orthographic projection matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 orthographic(
                        vec_float4_arg left,
                        vec_float4_arg right,
                        vec_float4_arg bottom,
                        vec_float4_arg top,
                        vec_float4_arg zNear,
                        vec_float4_arg zFar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>left</i>	Scalar value.
<i>right</i>	Scalar value.
<i>bottom</i>	Scalar value.
<i>top</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct an orthographic projection matrix, equal to:

$$\begin{matrix}
 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\
 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\
 0 & 0 & -2/(f-n) & -(f+n)/(f-n) \\
 0 & 0 & 0 & 1
 \end{matrix}$$

; l = *left*  
 ; r = *right*  
 ; b = *bottom*  
 ; t = *top*  
 ; n = *zNear*  
 ; f = *zFar*

SCE CONFIDENTIAL

# **perspective**

Construct a perspective projection matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 perspective(
                        vec_float4_arg fovyRadians,
                        vec_float4_arg aspect,
                        vec_float4_arg zNear,
                        vec_float4_arg zFar
                    );
                };
            }
        }
    }
}
```

## **Arguments**

<i>fovyRadians</i>	Scalar value.
<i>aspect</i>	Scalar value.
<i>zNear</i>	Scalar value.
<i>zFar</i>	Scalar value.

## **Return Values**

The constructed 4x4 matrix

## **Description**

Construct a perspective projection matrix, equal to:

$$\begin{matrix} \cot(y/2)/a & 0 & 0 & 0 \\ 0 & \cot(y/2) & 0 & 0 \\ 0 & 0 & (f+n)/(n-f) & 2*f*n/(n-f) \\ 0 & 0 & -1 & 0 \end{matrix}$$

*; y = fovyRadians  
 ; a = aspect  
 ; n = zNear  
 ; f = zFar*

# rotation

Construct a 4x4 matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotation(
                        vec_float4_arg radians,
                        Vector3_arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 4x4 matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationX(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 4x4 matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationY(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 4x4 matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationZ(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 4x4 matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                }
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 4x4 matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*scaleVec*      3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of *scaleVec*.

SCE CONFIDENTIAL

# translation

Construct a 4x4 matrix to perform translation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 translation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec* 3D vector.

## Return Values

The constructed 4x4 matrix

## Description

Construct a 4x4 matrix to perform translation, which is an identity matrix except for the translation component, with coordinates equal to those in *translateVec*.

SCE CONFIDENTIAL

## zero

Construct an identity 4x4 matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    static SCE_VECTORMATH_INLINE const Matrix4 zero();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 4x4 matrix

### Description

Construct a zero 4x4 matrix in which all elements are zero.

# Public Instance Methods

## get4Aos

Extract four AoS 4x4 matrices.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Matrix4 &result0,
                        Aos::Matrix4 &result1,
                        Aos::Matrix4 &result2,
                        Aos::Matrix4 &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 4x4 matrix.
<i>result1</i>	An output AoS 4x4 matrix.
<i>result2</i>	An output AoS 4x4 matrix.
<i>result3</i>	An output AoS 4x4 matrix.

### Return Values

None

### Description

Extract four AoS 4x4 matrices from four slots of an SoA 4x4 matrix (transpose the data format).

SCE CONFIDENTIAL

# getCol

Get the column of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

The column referred to by the specified index

## Description

Get the column of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol2

Get column 2 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol2() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 2

## Description

Get column 2 of a 4x4 matrix.

SCE CONFIDENTIAL

# getCol3

Get column 3 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getCol3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 3

## Description

Get column 3 of a 4x4 matrix.

SCE CONFIDENTIAL

# getElem

Get the element of a 4x4 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-3.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 4x4 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector4 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-3.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# getTranslation

Get the translation component of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Vector3 getTranslation() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Translation component

## Description

Get the translation component of a 4x4 matrix.

SCE CONFIDENTIAL

# getUpper3x3

Get the upper-left 3x3 submatrix of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE const Matrix3 getUpper3x3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Upper-left 3x3 submatrix

## Description

Get the upper-left 3x3 submatrix of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol

Set the column of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol(
                        int col,
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the column of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol0(
                        Vector4 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 0 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol1(
                        Vector4 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 1 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol2(
                        Vector4 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 2 of a 4x4 matrix.

SCE CONFIDENTIAL

# setCol3

Set column 3 of a 4x4 matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setCol3(
                        Vector4 arg col3
                    );
                }
            }
        }
    }
}
```

## Arguments

*col3*      4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set column 3 of a 4x4 matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 4x4 matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setElem(
                        int col,
                        int row,
                        vec_float4_arg val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-3.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the element of a 4x4 matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 4x4 matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setRow(
                        int row,
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-3.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the row of a 4x4 matrix referred to by the specified index.

SCE CONFIDENTIAL

# setTranslation

Set translation component.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setTranslation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec*    3D vector.

## Return Values

A reference to the resulting 4x4 matrix

## Description

Set the translation component of a 4x4 matrix equal to the specified 3D vector.

## Notes

This function does not change the bottom row elements.

SCE CONFIDENTIAL

# **setUpper3x3**

Set the upper-left 3x3 submatrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Matrix4 {
                    SCE_VECTORMATH_INLINE Matrix4 &setUpper3x3(
                        Matrix3 arg mat3
                    );
                };
            }
        }
    }
}
```

## **Arguments**

*mat3*      3x3 matrix.

## **Return Values**

A reference to the resulting 4x4 matrix

## **Description**

Set the upper-left 3x3 submatrix elements of a 4x4 matrix equal to the specified 3x3 matrix.

## **Notes**

This function does not change the bottom row elements.

**sce::Vectormath::Simd::Soa::Point3**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Point3

A set of four 3D points in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Point3 {};
```

### Description

A class representing a set of four 3D points stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 3D points.
<a href="#">getElem</a>	Get an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D point by index.
<a href="#">getX</a>	Get the <i>x</i> element of a 3D point.
<a href="#">getXY</a>	Get the <i>x</i> and <i>y</i> elements of a 3D point.
<a href="#">getY</a>	Get the <i>y</i> element of a 3D point.
<a href="#">getZ</a>	Get the <i>z</i> element of a 3D point.
<a href="#">operator+</a>	Add a 3D point to a 3D vector.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3D vector.
<a href="#">operator-</a>	Subtract a 3D point from another 3D point.
<a href="#">operator_-</a>	Subtract a 3D vector from a 3D point.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3D vector.
<a href="#">operator=</a>	Assign one 3D point to another.
<a href="#">operator[]</a>	Subscripting operator to set or get an element.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">origin</a>	Construct a point representing the origin.
<a href="#">Point3</a>	Default constructor; does no initialization.
<a href="#">Point3</a>	Copy constructor.
<a href="#">Point3</a>	Construct a 3D point from <i>x</i> , <i>y</i> , and <i>z</i> elements.
<a href="#">Point3</a>	Construct a 3D point from a 2D vector and a scalar.
<a href="#">Point3</a>	Copy elements from a 3D vector into a 3D point.
<a href="#">Point3</a>	Set all elements of a 3D point to the same scalar value.
<a href="#">Point3</a>	Replicate an AoS 3D point.
<a href="#">Point3</a>	Insert four AoS 3D points.
<a href="#">setElem</a>	Set an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D point by index.
<a href="#">setX</a>	Set the <i>x</i> element of a 3D point.
<a href="#">setXY</a>	Set the <i>x</i> and <i>y</i> elements of a 3D point.
<a href="#">setY</a>	Set the <i>y</i> element of a 3D point.
<a href="#">setZ</a>	Set the <i>z</i> element of a 3D point.

# Constructors and Destructors

## Point3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Point3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Point3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Point3(
                        const Point3 &pnt
                    );
                }
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from  $x$ ,  $y$ , and  $z$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        vec_float4_arg x,
                        vec_float4_arg y,
                        vec_float4_arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.

## Return Values

None

## Description

Construct a 3D point containing the specified  $x$ ,  $y$ , and  $z$  elements.

SCE CONFIDENTIAL

# Point3

Construct a 3D point from a 2D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        Vector2 arg xy,
                        vec_float4_arg z
                    );
                };
            };
        };
    };
}
```

## Arguments

xy	2D vector.
z	Scalar value.

## Return Values

None

## Description

Construct a 3D point. The  $x$  and  $y$  elements are set to those of the specified 2D vector, and the  $z$  element is set to the specified scalar value.

SCE CONFIDENTIAL

# Point3

Copy elements from a 3D vector into a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Construct a 3D point containing the  $x$ ,  $y$ , and  $z$  elements of the specified 3D vector.

SCE CONFIDENTIAL

# Point3

Set all elements of a 3D point to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    explicit SCE_VECTORMATH_INLINE Point3(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a 3D point with all elements set to the scalar value argument.

SCE CONFIDENTIAL

## Point3

## Replicate an AoS 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        Aos::Point3 arg pnt
                    );
                }
            }
        }
    }
}
```

## Arguments

*pnt* AoS 3D point.

## Return Values

None

## Description

Replicate an AoS 3D point in all four slots of an SoA 3D point.

SCE CONFIDENTIAL

# Point3

Insert four AoS 3D points.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3(
                        Aos::Point3 arg pnt0,
                        Aos::Point3 arg pnt1,
                        Aos::Point3 arg pnt2,
                        Aos::Point3 arg pnt3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>pnt0</i>	AoS 3D point.
<i>pnt1</i>	AoS 3D point.
<i>pnt2</i>	AoS 3D point.
<i>pnt3</i>	AoS 3D point.

## Return Values

None

## Description

Insert four AoS 3D points into four slots of an SoA 3D point (transpose the data format).

# Operator Methods

## **operator+**

Add a 3D point to a 3D vector.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Point3 operator+
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*vec*                    3D vector.

### **Return Values**

Sum of the specified 3D point and 3D vector

### **Description**

Add a 3D point to a 3D vector.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator+=(  
                        Vector3 arg vec  
                    );
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D point

## Description

Perform compound assignment and addition with a 3D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3D point from another 3D point.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-
                        (Point3 arg pnt
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

Difference of the specified 3D points

## **Description**

Subtract a 3D point from another 3D point.

SCE CONFIDENTIAL

# operator-

Subtract a 3D vector from a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE const Point3 operator-
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Difference of the specified 3D point and 3D vector

## Description

Subtract a 3D vector from a 3D point.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator==(Vector3 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D point

## Description

Perform compound assignment and subtraction by a 3D vector.

SCE CONFIDENTIAL

# **operator=**

Assign one 3D point to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &operator=(  
                        Point3 arg pnt  
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

A reference to the resulting 3D point

## **Description**

Assign one 3D point to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 &operator[](int idx)
                    ;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

A reference to indexed element

## Description

Subscripting operator invoked when applied to non-const [Point3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 operator[](int idx) const;
                };
            };
        };
    };
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Point3](#).

# Public Static Methods

## origin

Construct a point representing the origin.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    static SCE_VECTORMATH_INLINE const Point3 origin();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3D point

### Description

Construct a 3D point equal to (0,0,0).

# Public Instance Methods

## get4Aos

Extract four AoS 3D points.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Point3 &result0,
                        Aos::Point3 &result1,
                        Aos::Point3 &result2,
                        Aos::Point3 &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 3D point.
<i>result1</i>	An output AoS 3D point.
<i>result2</i>	An output AoS 3D point.
<i>result3</i>	An output AoS 3D point.

### Return Values

None

### Description

Extract four AoS 3D points from four slots of an SoA 3D point (transpose the data format).

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, or *z* element of a 3D point by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, or *z* element of a 3D point by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 getX() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 3D point

## Description

Get the *x* element of a 3D point.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a 3D point's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 3D point

## Description

Get the *y* element of a 3D point.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE vec_float4 getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 3D point

## Description

Get the *z* element of a 3D point.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, or *z* element of a 3D point by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setElem(
                        int idx,
                        vec_float4_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set an *x*, *y*, or *z* element of a 3D point by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setX(
                        vec_float4_arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the *x* element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 3D point

## Description

Set a 3D point's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setY(
                        vec_float4_arg y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the  $y$  element of a 3D point to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Point3 {
                    SCE_VECTORMATH_INLINE Point3 &setZ(
                        vec_float4_arg z
                    );
                }
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 3D point

## Description

Set the *z* element of a 3D point to the specified scalar value.

**sce::Vectormath::Simd::Soa::Quat**

000004892117

# Summary

# sce::Vectormath::Simd::Soa::Quat

A set of four quaternions in structure-of-arrays format.

## Definition

```
#include <vectormath.h>
class Quat {};
```

## Description

A class representing a set of four quaternions stored in structure-of-arrays (SoA) format.

## Methods Summary

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>rotation</u></a>	Construct a quaternion from an Euler rotation.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate between two unit-length 3D vectors.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate around a unit-length 3D vector.
<a href="#"><u>rotationX</u></a>	Construct a quaternion to rotate around the x axis.
<a href="#"><u>rotationY</u></a>	Construct a quaternion to rotate around the y axis.
<a href="#"><u>rotationZ</u></a>	Construct a quaternion to rotate around the z axis.
<a href="#"><u>setElem</u></a>	Set an <i>x</i> , <i>y</i> , <i>z</i> , or <i>w</i> element of a quaternion by index.
<a href="#"><u>setW</u></a>	Set the <i>w</i> element of a quaternion.
<a href="#"><u>setX</u></a>	Set the <i>x</i> element of a quaternion.
<a href="#"><u>setXY</u></a>	Set the <i>x</i> and <i>y</i> elements of a quaternion.
<a href="#"><u>setXYZ</u></a>	Set the <i>x</i> , <i>y</i> , and <i>z</i> elements of a quaternion.
<a href="#"><u>setY</u></a>	Set the <i>y</i> element of a quaternion.
<a href="#"><u>setZ</u></a>	Set the <i>z</i> element of a quaternion.
<a href="#"><u>zero</u></a>	Construct a zero quaternion.

# Constructors and Destructors

## Quat

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE Quat();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Quat

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_ALWAYS_INLINE Quat(
                        const Quat &quat
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      quaternion.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        vec_float4_arg x,
                        vec_float4_arg y,
                        vec_float4_arg z,
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.
$w$	Scalar value.

## Return Values

None

## Description

Construct a quaternion containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Quat

Construct a quaternion from a 3D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        Vector3 arg xyz,
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a quaternion with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Quat

Copy elements from a 4D vector into a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

`vec`      4D vector.

## Return Values

None

## Description

Construct a quaternion containing the  $x$ ,  $y$ ,  $z$ , and  $w$  elements of the specified 4D vector.

SCE CONFIDENTIAL

# Quat

Convert a rotation matrix to a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        Matrix3 arg rotMat
                    );
                }
            }
        }
    }
}
```

## Arguments

*rotMat*      3x3 matrix, expected to be a rotation matrix.

## Return Values

None

## Description

Construct a unit-length quaternion representing the same transformation as a rotation matrix.

SCE CONFIDENTIAL

# Quat

Set all elements of a quaternion to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    explicit SCE_VECTORMATH_INLINE Quat(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

None

## Description

Construct a quaternion with all elements set to the scalar value argument.

SCE CONFIDENTIAL

## Quat

## Replicate an AoS quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        Aos::Quat_arg quat
                    );
                }
            }
        }
    }
}
```

## Arguments

## Return Values

None

## Description

Replicate an AoS quaternion in all four slots of an SoA quaternion.

SCE CONFIDENTIAL

# Quat

Insert four AoS quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat(
                        Aos::Quat_arg quat0,
                        Aos::Quat_arg quat1,
                        Aos::Quat_arg quat2,
                        Aos::Quat_arg quat3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>quat0</i>	AoS quaternion.
<i>quat1</i>	AoS quaternion.
<i>quat2</i>	AoS quaternion.
<i>quat3</i>	AoS quaternion.

## Return Values

None

## Description

Insert four AoS quaternions into four slots of an SoA quaternion (transpose the data format).

# Operator Methods

## **operator\***

Multiply two quaternions.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator*(
                        Quat arg quat
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*quat*                  Quaternion.

### **Return Values**

Product of the specified quaternions

### **Description**

Multiply two quaternions.

SCE CONFIDENTIAL

# operator\*

Multiply a quaternion by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Product of the specified quaternion and scalar

## Description

Multiply a quaternion by a scalar.

SCE CONFIDENTIAL

# operator\*=

Perform compound assignment and multiplication by a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator*=(  

                        Quat arg quat  

                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and multiplication by a quaternion.

SCE CONFIDENTIAL

# operator\*=

Perform compound assignment and multiplication by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two quaternions.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator+
                        (Quat arg quat
                         ) const;
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

Sum of the specified quaternions

## Description

Add two quaternions.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator+=(  
                        Quat arg quat  
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and addition with a quaternion.

SCE CONFIDENTIAL

# operator-

Subtract a quaternion from another quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator-(
                        Quat arg quat
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

Difference of the specified quaternions

## Description

Subtract a quaternion from another quaternion.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a quaternion.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

Quaternion containing negated elements of the specified quaternion

## **Description**

Negate all elements of a quaternion.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator==(Quat arg quat
                );
            }
        }
    }
}
```

## Arguments

*quat*      Quaternion.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and subtraction by a quaternion.

SCE CONFIDENTIAL

# operator/

Divide a quaternion by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Quat operator/(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified quaternion and scalar

## Description

Divide a quaternion by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator/=(
                        vec_float4_arg scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one quaternion to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &operator=(Quat arg quat
                        );
                }
            }
        }
    }
}
```

## **Arguments**

*quat*      Quaternion.

## **Return Values**

A reference to the resulting quaternion

## **Description**

Assign one quaternion to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 &operator[](int idx)
                    ;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

A reference to indexed element

## Description

Subscripting operator invoked when applied to non-const [Quat](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 operator[](int idx) const;
                };
            };
        };
    };
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Quat](#).

# Public Static Methods

## axisAngle

Construct an axis-angle rotation from a quaternion.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Vector4 axisAngle(
                        Quat_arg unitQuat
                    );
                }
            }
        }
    }
}
```

### Arguments

*unitQuat*      Unit rotation quaternion.

### Return Values

A 4D vector containing a 3D xyz vector, representing the unit-length axis, and a w element that specifies the rotation angle in radians.

### Description

Construct an axis-angle rotation from a quaternion.

SCE CONFIDENTIAL

# euler

Construct an Euler rotation from a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Vector3 euler(
                        Quat arg unitQuat,
                        RotationOrder order
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>unitQuat</i>	Unit rotation quaternion.
<i>order</i>	Euler rotation order.

## Return Values

A 3D vector containing values representing Euler angle rotations with an element ordering specified by the value *order*

## Description

Construct an Euler rotation from a quaternion.

## Notes

If your quaternion has been formed from Euler angles that suffer from pitch / pole singularities, then you will not get the same angles back that you put in.

SCE CONFIDENTIAL

# identity

Construct an identity quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat identity();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed quaternion

## Description

Construct an identity quaternion equal to (0,0,0,1).

SCE CONFIDENTIAL

# rotation

Construct a quaternion from an Euler rotation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        Vector3 arg radians,
                        RotationOrder order
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	3D vector of the Euler X, Y, and Z rotations in radians.
<i>order</i>	Euler rotation order.

## Return Values

A quaternion that represents the compounded rotation specified by the elements of *radians*

## Description

Construct a quaternion from an Euler rotation.

SCE CONFIDENTIAL

# rotation

Construct a quaternion to rotate between two unit-length 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        Vector3 arg unitVec0,
                        Vector3 arg unitVec1
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>unitVec0</i>	3D vector, expected to be unit-length.
<i>unitVec1</i>	3D vector, expected to be unit-length.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate between two unit-length 3D vectors.

## Notes

The result is unpredictable if *unitVec0* and *unitVec1* point in opposite directions.

SCE CONFIDENTIAL

# rotation

Construct a quaternion to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotation(
                        vec_float4_arg radians,
                        Vector3_arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotationX

Construct a quaternion to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationX(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a quaternion to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationY(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a quaternion to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat rotationZ(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed quaternion

## Description

Construct a quaternion to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

---

## zero

---

---

Construct a zero quaternion.

### Definition

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    static SCE_VECTORMATH_INLINE const Quat zero();
                }
            }
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed quaternion

### Description

---

Construct an zero quaternion equal to (0,0,0,0).

# Public Instance Methods

## get4Aos

Extract four AoS quaternions.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Quat &result0,
                        Aos::Quat &result1,
                        Aos::Quat &result2,
                        Aos::Quat &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS quaternion.
<i>result1</i>	An output AoS quaternion.
<i>result2</i>	An output AoS quaternion.
<i>result3</i>	An output AoS quaternion.

### Return Values

None

### Description

Extract four AoS quaternions from four slots of an SoA quaternion (transpose the data format).

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, *z*, or *w* element of a quaternion by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, *z*, or *w* element of a quaternion by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# getW

Get the *w* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 getW() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*w* element of a quaternion

## Description

Get the *w* element of a quaternion.

SCE CONFIDENTIAL

# getX

Get the *x* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 getx() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a quaternion

## Description

Get the *x* element of a quaternion.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a quaternion's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getXYZ

Get the  $x$ ,  $y$ , and  $z$  elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE const Vector3 getXYZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

3D vector containing  $x$ ,  $y$ , and  $z$  elements

## Description

Extract a quaternion's  $x$ ,  $y$ , and  $z$  elements into a 3D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a quaternion

## Description

Get the *y* element of a quaternion.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE vec_float4 getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a quaternion

## Description

Get the *z* element of a quaternion.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a quaternion by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setElem(
                        int idx,
                        vec_float4_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set an *x*, *y*, *z*, or *w* element of a quaternion by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setW

Set the *w* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setW(
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

*w*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *w* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setx(
                        vec_float4_arg x
                    );
                };
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *x* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting quaternion

## Description

Set a quaternion's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* and *w* elements.

SCE CONFIDENTIAL

# setXYZ

Set the *x*, *y*, and *z* elements of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setXYZ(
                        Vector3 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                   3D vector.

## Return Values

A reference to the resulting quaternion

## Description

Set a quaternion's *x*, *y*, and *z* elements to those of the specified 3D vector.

## Notes

This function does not change the *w* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setY(
                        vec_float4_arg y
                    );
                };
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the  $y$  element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Quat {
                    SCE_VECTORMATH_INLINE Quat &setZ(
                        vec_float4_arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting quaternion

## Description

Set the *z* element of a quaternion to the specified scalar value.

SCE CONFIDENTIAL

---

**sce::Vectormath::  
Simd::Soa::Transform3**

# Summary

## sce::Vectormath::Simd::Soa::Transform3

A set of four 3x4 transformation matrices in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Transform3 {};
```

### Description

A class representing a set of four 3x4 transformation matrices stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 3x4 transformation matrices.
<a href="#">getCol</a>	Get the column of a 3x4 transformation matrix referred to by the specified index.
<a href="#">getCol0</a>	Get column 0 of a 3x4 transformation matrix.
<a href="#">getCol1</a>	Get column 1 of a 3x4 transformation matrix.
<a href="#">getCol2</a>	Get column 2 of a 3x4 transformation matrix.
<a href="#">getCol3</a>	Get column 3 of a 3x4 transformation matrix.
<a href="#">getElem</a>	Get the element of a 3x4 transformation matrix referred to by column and row indices.
<a href="#">getRow</a>	Get the row of a 3x4 transformation matrix referred to by the specified index.
<a href="#">getTranslation</a>	Get the translation component of a 3x4 transformation matrix.
<a href="#">getUpper3x3</a>	Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.
<a href="#">identity</a>	Construct an identity 3x4 transformation matrix.
<a href="#">operator*</a>	Multiply a 3x4 transformation matrix by a 3D vector.
<a href="#">operator*</a>	Multiply a 3x4 transformation matrix by a 3D point.
<a href="#">operator*</a>	Multiply two 3x4 transformation matrices.
<a href="#">operator*=</a>	Perform compound assignment and multiplication by a 3x4 transformation matrix.
<a href="#">operator=</a>	Assign one 3x4 transformation matrix to another.
<a href="#">operator[]</a>	Subscripting operator to set or get a column.
<a href="#">operator[]</a>	Subscripting operator to get a column.
<a href="#">rotation</a>	Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector.
<a href="#">rotation</a>	Construct a rotation matrix from a unit-length quaternion.
<a href="#">rotationX</a>	Construct a 3x4 transformation matrix to rotate around the x axis.
<a href="#">rotationY</a>	Construct a 3x4 transformation matrix to rotate around the y axis.
<a href="#">rotationZ</a>	Construct a 3x4 transformation matrix to rotate around the z axis.
<a href="#">rotationZYX</a>	Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.
<a href="#">scale</a>	Construct a 3x4 transformation matrix to perform scaling.
<a href="#">setCol</a>	Set the column of a 3x4 transformation matrix referred to by the specified index.
<a href="#">setCol0</a>	Set column 0 of a 3x4 transformation matrix.
<a href="#">setCol1</a>	Set column 1 of a 3x4 transformation matrix.
<a href="#">setCol2</a>	Set column 2 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

Methods	Description
<a href="#"><u>setCol3</u></a>	Set column 3 of a 3x4 transformation matrix.
<a href="#"><u>setElem</u></a>	Set the element of a 3x4 transformation matrix referred to by column and row indices.
<a href="#"><u>setRow</u></a>	Set the row of a 3x4 transformation matrix referred to by the specified index.
<a href="#"><u>setTranslation</u></a>	Set translation component.
<a href="#"><u>setUpper3x3</u></a>	Set the upper-left 3x3 submatrix.
<a href="#"><u>Transform3</u></a>	Default constructor; does no initialization.
<a href="#"><u>Transform3</u></a>	Copy constructor.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix containing the specified columns.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix from a 3x3 matrix and a 3D vector.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix from a unit-length quaternion and a 3D vector.
<a href="#"><u>Transform3</u></a>	Set all elements of a 3x4 transformation matrix to the same scalar value.
<a href="#"><u>Transform3</u></a>	Replicate an AoS 3x4 transformation matrix.
<a href="#"><u>Transform3</u></a>	Insert four AoS 3x4 transformation matrices.
<a href="#"><u>translation</u></a>	Construct a 3x4 transformation matrix to perform translation.

# Constructors and Destructors

## Transform3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Transform3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Transform3(
                        const Transform3 &tfrm
                    );
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix containing the specified columns.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Vector3 arg col0,
                        Vector3 arg col1,
                        Vector3 arg col2,
                        Vector3 arg col3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>col0</i>	3D vector.
<i>col1</i>	3D vector.
<i>col2</i>	3D vector.
<i>col3</i>	3D vector.

## Return Values

None

## Description

Construct a 3x4 transformation matrix containing the specified columns.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix from a 3x3 matrix and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Matrix3 arg tfrm,
                        Vector3 arg translateVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	3x3 matrix.
<i>translateVec</i>	3D vector.

## Return Values

None

## Description

Construct a 3x4 transformation matrix whose upper 3x3 elements are equal to the 3x3 matrix argument and whose translation component is equal to the 3D vector argument.

SCE CONFIDENTIAL

# Transform3

Construct a 3x4 transformation matrix from a unit-length quaternion and a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Quat arg unitQuat,
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

<code>unitQuat</code> <code>translateVec</code>	Quaternion, expected to be unit-length. 3D vector.
--	---

## Return Values

None

## Description

Construct a 3x4 transformation matrix whose upper-left 3x3 submatrix is a rotation matrix converted from the unit-length quaternion argument and whose translation component is equal to the 3D vector argument.

SCE CONFIDENTIAL

# Transform3

Set all elements of a 3x4 transformation matrix to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    explicit SCE_VECTORMATH_INLINE Transform3(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 3x4 transformation matrix with all elements set to the *scalar* value argument.

SCE CONFIDENTIAL

# Transform3

Replicate an AoS 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Aos::Transform3_arg tfrm
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>tfrm</i>	AoS 3x4 transformation matrix.
-------------	--------------------------------

## Return Values

None

## Description

Replicate an AoS 3x4 transformation matrix in all four slots of an SoA 3x4 transformation matrix.

SCE CONFIDENTIAL

# Transform3

Insert four AoS 3x4 transformation matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3(
                        Aos::Transform3_arg tfrm0,
                        Aos::Transform3_arg tfrm1,
                        Aos::Transform3_arg tfrm2,
                        Aos::Transform3_arg tfrm3
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>tfrm0</i>	AoS 3x4 transformation matrix.
<i>tfrm1</i>	AoS 3x4 transformation matrix.
<i>tfrm2</i>	AoS 3x4 transformation matrix.
<i>tfrm3</i>	AoS 3x4 transformation matrix.

## Return Values

None

## Description

Insert four AoS 3x4 transformation matrices into four slots of an SoA 3x4 transformation matrix (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 3x4 transformation matrix by a 3D vector.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        Vector3 arg
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*vec*                    3D vector.

### **Return Values**

Product of the specified 3x4 transformation matrix and 3D vector

### **Description**

Applies the 3x3 upper-left submatrix (but not the translation component) of a 3x4 transformation matrix to a 3D vector.

SCE CONFIDENTIAL

# **operator\***

Multiply a 3x4 transformation matrix by a 3D point.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Point3 operator*(
                        Point3 arg pnt
                    ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*pnt*                    3D point.

## **Return Values**

Product of the specified 3x4 transformation matrix and 3D point

## **Description**

Applies the 3x3 upper-left submatrix and the translation component of a 3x4 transformation matrix to a 3D point.

SCE CONFIDENTIAL

# operator\*

Multiply two 3x4 transformation matrices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Transform3 operator*(
                        Transform3 arg tfrm
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*tfrm*      3x4 transformation matrix.

## Return Values

Product of the specified 3x4 transformation matrices

## Description

Multiply two 3x4 transformation matrices.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &operator*=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*      3x4 transformation matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Perform compound assignment and multiplication by a 3x4 transformation matrix.

SCE CONFIDENTIAL

# **operator=**

Assign one 3x4 transformation matrix to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &operator=(  

                        Transform3 arg tfrm  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*tfrm*      3x4 transformation matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Assign one 3x4 transformation matrix to another.

SCE CONFIDENTIAL

# **operator[]**

Subscripting operator to set or get a column.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator[](int col)
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*col* Index, expected in the range 0-3.

## **Return Values**

A reference to the indexed column

## **Description**

Subscripting operator invoked when applied to non-const [Transform3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get a column.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator[](int col) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

Indexed column

## Description

Subscripting operator invoked when applied to const [Transform3](#).

# Public Static Methods

## identity

Construct an identity 3x4 transformation matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 identity();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3x4 transformation matrix

### Description

Construct an identity 3x4 transformation matrix in which non-diagonal elements are zero and diagonal elements are 1.

SCE CONFIDENTIAL

# rotation

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotation(
                        vec_float4_arg radians,
                        Vector3_arg unitVec
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
<i>unitVec</i>	3D vector, expected to be unit-length.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around a unit-length 3D vector by the specified radians angle.

SCE CONFIDENTIAL

# rotation

Construct a rotation matrix from a unit-length quaternion.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotation(
                        Quat arg unitQuat
                    );
                }
            }
        }
    }
}
```

## Arguments

*unitQuat*      Quaternion, expected to be unit-length.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix that applies the same rotation as the specified unit-length quaternion.

SCE CONFIDENTIAL

# rotationX

Construct a 3x4 transformation matrix to rotate around the x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationX(
                        vec_float4_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the x axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationY

Construct a 3x4 transformation matrix to rotate around the y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationY(
                        vec_float4_arg radians
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>radians</i>	Scalar value.
----------------	---------------

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the y axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZ

Construct a 3x4 transformation matrix to rotate around the z axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationZ(
                        vec_float4_arg radians
                    );
                }
            }
        }
    }
}
```

## Arguments

*radians*      Scalar value.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the z axis by the specified radians angle.

SCE CONFIDENTIAL

# rotationZYX

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 rotationZYX(
                        Vector3 arg radiansXYZ
                    );
                };
            }
        }
    }
}
```

## Arguments

*radiansXYZ*      3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes by the radians angles contained in a 3D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

SCE CONFIDENTIAL

# scale

Construct a 3x4 transformation matrix to perform scaling.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 scale(
                        Vector3 arg scaleVec
                    );
                };
            }
        }
    }
}
```

## Arguments

scaleVec      3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to perform scaling, in which the non-diagonal elements are zero and the diagonal elements are set to the elements of scaleVec.

SCE CONFIDENTIAL

# translation

Construct a 3x4 transformation matrix to perform translation.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    static SCE_VECTORMATH_INLINE const Transform3 translation(
                        Vector3 arg translateVec
                    );
                };
            }
        }
    }
}
```

## Arguments

*translateVec* 3D vector.

## Return Values

The constructed 3x4 transformation matrix

## Description

Construct a 3x4 transformation matrix to perform translation, which is an identity matrix except for the translation component, with coordinates equal to those in *translateVec*.

# Public Instance Methods

## get4Aos

Extract four AoS 3x4 transformation matrices.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Transform3 &result0,
                        Aos::Transform3 &result1,
                        Aos::Transform3 &result2,
                        Aos::Transform3 &result3
                    ) const;
                };
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 3x4 transformation matrix.
<i>result1</i>	An output AoS 3x4 transformation matrix.
<i>result2</i>	An output AoS 3x4 transformation matrix.
<i>result3</i>	An output AoS 3x4 transformation matrix.

### Return Values

None

### Description

Extract four AoS 3x4 transformation matrices from four slots of an SoA 3x4 transformation matrix (transpose the data format).

SCE CONFIDENTIAL

# getCol

Get the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol(
                        int col
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*col* Index, expected in the range 0-3.

## Return Values

The column referred to by the specified index

## Description

Get the column of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# getCol0

Get column 0 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol0() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 0

## Description

Get column 0 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getCol1

Get column 1 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol1() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 1

## Description

Get column 1 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

## getCol2

Get column 2 of a 3x4 transformation matrix.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol2() const;
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

Column 2

### Description

Get column 2 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getCol3

Get column 3 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getCol3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Column 3

## Description

Get column 3 of a 3x4 transformation matrix.

# getElem

Get the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int col,
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-2.

## Return Values

Element selected by *col* and *row*

## Description

Get the element of a 3x4 transformation matrix referred to by column and row indices.

SCE CONFIDENTIAL

# getRow

Get the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector4 getRow(
                        int row
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*row* Index, expected in the range 0-2.

## Return Values

The row referred to by the specified index

## Description

Get the row of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# getTranslation

Get the translation component of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Vector3 getTranslation() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Translation component

## Description

Get the translation component of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# getUpper3x3

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE const Matrix3 getUpper3x3() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

Upper-left 3x3 submatrix

## Description

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol

Set the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol(
                        int col,
                        Vector3 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>vec</i>	3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the column of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# setCol0

Set column 0 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol0(
                        Vector3 arg col0
                    );
                }
            }
        }
    }
}
```

## Arguments

*col0*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 0 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol1

Set column 1 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol1(
                        Vector3 arg col1
                    );
                }
            }
        }
    }
}
```

## Arguments

*col1*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 1 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol2

Set column 2 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol2(
                        Vector3 arg col2
                    );
                }
            }
        }
    }
}
```

## Arguments

*col2*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 2 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setCol3

Set column 3 of a 3x4 transformation matrix.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setCol3(
                        Vector3 arg col3
                    );
                }
            }
        }
    }
}
```

## Arguments

*col3*      3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set column 3 of a 3x4 transformation matrix.

SCE CONFIDENTIAL

# setElem

Set the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setElem(
                        int col,
                        int row,
                        vec_float4_arg val
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>col</i>	Index, expected in the range 0-3.
<i>row</i>	Index, expected in the range 0-2.
<i>val</i>	Scalar value.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the element of a 3x4 transformation matrix referred to by column and row indices.

SCE CONFIDENTIAL

# setRow

Set the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setRow(
                        int row,
                        Vector4 arg vec
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>row</i>	Index, expected in the range 0-2.
<i>vec</i>	4D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the row of a 3x4 transformation matrix referred to by the specified index.

SCE CONFIDENTIAL

# setTranslation

Set translation component.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setTranslation(
                        Vector3 arg translateVec
                    );
                }
            }
        }
    }
}
```

## Arguments

*translateVec*    3D vector.

## Return Values

A reference to the resulting 3x4 transformation matrix

## Description

Set the translation component of a 3x4 transformation matrix equal to the specified 3D vector.

SCE CONFIDENTIAL

# **setUpUpper3x3**

Set the upper-left 3x3 submatrix.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Transform3 {
                    SCE_VECTORMATH_INLINE Transform3 &setUpUpper3x3(
                        Matrix3 arg mat3
                    );
                }
            }
        }
    }
}
```

## **Arguments**

*mat3*      3x3 matrix.

## **Return Values**

A reference to the resulting 3x4 transformation matrix

## **Description**

Set the upper-left 3x3 submatrix elements of a 3x4 transformation matrix equal to the specified 3x3 matrix.

**sce::Vectormath::Simd::Soa::Vector2**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Vector2

A set of four 2D vectors in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Vector2 {};
```

### Description

A class representing a set of four 2D vectors stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4AoS</a>	Extract four AoS 2D vectors.
<a href="#">getElem</a>	Get an <i>x</i> or <i>y</i> element of a 2D vector by index.
<a href="#">getX</a>	Get the <i>x</i> element of a 2D vector.
<a href="#">getY</a>	Get the <i>y</i> element of a 2D vector.
<a href="#">operator*</a>	Multiply a 2D vector by a scalar.
<a href="#">operator*=<sup>=</sup></a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator+<sup>=</sup></a>	Add two 2D vectors.
<a href="#">operator+=<sup>=</sup></a>	Perform compound assignment and addition with a 2D vector.
<a href="#">operator-<sup>=</sup></a>	Subtract a 2D vector from another 2D vector.
<a href="#">operator-<sup>=</sup></a>	Negate all elements of a 2D vector.
<a href="#">operator-=<sup>=</sup></a>	Perform compound assignment and subtraction by a 2D vector.
<a href="#">operator/<sup>=</sup></a>	Divide a 2D vector by a scalar.
<a href="#">operator/=<sup>=</sup></a>	Perform compound assignment and division by a scalar.
<a href="#">operator=</a>	Assign one 2D vector to another.
<a href="#">operator[]</a>	Subscripting operator to set or get an element.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">setElem</a>	Set an <i>x</i> or <i>y</i> element of a 2D vector by index.
<a href="#">setX</a>	Set the <i>x</i> element of a 2D vector.
<a href="#">setY</a>	Set the <i>y</i> element of a 2D vector.
<a href="#">Vector2</a>	Default constructor; does no initialization.
<a href="#">Vector2</a>	Copy constructor.
<a href="#">Vector2</a>	Construct a 2D vector from <i>x</i> and <i>y</i> elements.
<a href="#">Vector2</a>	Set all elements of a 2D vector to the same scalar value.
<a href="#">Vector2</a>	Replicate an AoS 2D vector.
<a href="#">Vector2</a>	Insert four AoS 2D vectors.
<a href="#">xAxis</a>	Construct x axis.
<a href="#">yAxis</a>	Construct y axis.
<a href="#">zero</a>	Construct zero.

# Constructors and Destructors

## Vector2

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector2();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector2

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector2(
                        const Vector2 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector2

Construct a 2D vector from *x* and *y* elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2(
                        vec_float4_arg x,
                        vec_float4_arg y
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>x</i>	Scalar value.
<i>y</i>	Scalar value.

## Return Values

None

## Description

Construct a 2D vector containing the specified *x* and *y* elements.

SCE CONFIDENTIAL

# Vector2

Set all elements of a 2D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    explicit SCE_VECTORMATH_INLINE Vector2(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 2D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector2

Replicate an AoS 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2(
                        Aos::Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

vec                    AoS 2D vector.

## Return Values

None

## Description

Replicate an AoS 2D vector in all four slots of an SoA 2D vector.

SCE CONFIDENTIAL

# Vector2

Insert four AoS 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2(
                        Aos::Vector2 arg vec0,
                        Aos::Vector2 arg vec1,
                        Aos::Vector2 arg vec2,
                        Aos::Vector2 arg vec3
                    );
                };
            }
        }
    }
}
```

## Arguments

vec0	AoS 2D vector.
vec1	AoS 2D vector.
vec2	AoS 2D vector.
vec3	AoS 2D vector.

## Return Values

None

## Description

Insert four AoS 2D vectors into four slots of an SoA 2D vector (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 2D vector by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 2D vector and scalar

### **Description**

Multiply a 2D vector by a scalar.

SCE CONFIDENTIAL

# **operator\*= =====**

Perform compound assignment and multiplication by a scalar.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## **Arguments**

*scalar*              Scalar value.

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator+
                        (Vector2 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

Sum of the specified 2D vectors

## Description

Add two 2D vectors.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator+=(
                        Vector2 arg
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

A reference to the resulting 2D vector

## Description

Perform compound assignment and addition with a 2D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 2D vector from another 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator-
                        (Vector2 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

vec                  2D vector.

## **Return Values**

Difference of the specified 2D vectors

## **Description**

Subtract a 2D vector from another 2D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 2D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

2D vector containing negated elements of the specified 2D vector

## **Description**

Negate all elements of a 2D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator==(Vector2 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                  2D vector.

## Return Values

A reference to the resulting 2D vector

## Description

Perform compound assignment and subtraction by a 2D vector.

SCE CONFIDENTIAL

# operator/

Divide a 2D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE const Vector2 operator/(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified 2D vector and scalar

## Description

Divide a 2D vector by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator/=(
                        vec_float4_arg scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 2D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &operator=(Vector2 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            }
        }
    }
}
```

## **Arguments**

*vec*                  2D vector.

## **Return Values**

A reference to the resulting 2D vector

## **Description**

Assign one 2D vector to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE vec_float4 &operator[](int idx)
                    ;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-1.

## Return Values

A reference to indexed element

## Description

Subscripting operator invoked when applied to non-const [Vector2](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE vec_float4 operator[](int idx) const;
                };
            };
        };
    };
}
```

## Arguments

*idx* Index, expected in the range 0-1.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector2](#).

# Public Static Methods

## xAxis

Construct x axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 xAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 2D vector

### Description

Construct a 2D vector equal to (1,0).

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 yAxis();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 2D vector

## Description

Construct a 2D vector equal to (0,1).

SCE CONFIDENTIAL

---

## zero

---

---

Construct zero.

### Definition

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    static SCE_VECTORMATH_INLINE const Vector2 zero();
                }
            }
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed 2D vector

### Description

---

Construct a 2D vector equal to (0,0).

# Public Instance Methods

## get4Aos

Extract four AoS 2D vectors.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Vector2 &result0,
                        Aos::Vector2 &result1,
                        Aos::Vector2 &result2,
                        Aos::Vector2 &result3
                    ) const;
                }
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 2D vector.
<i>result1</i>	An output AoS 2D vector.
<i>result2</i>	An output AoS 2D vector.
<i>result3</i>	An output AoS 2D vector.

### Return Values

None

### Description

Extract four AoS 2D vectors from four slots of an SoA 2D vector (transpose the data format).

SCE CONFIDENTIAL

# getElem

Get an *x* or *y* element of a 2D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-1.

## Return Values

Element selected by the specified index

## Description

Get an *x* or *y* element of a 2D vector by specifying an index of 0 or 1 respectively.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE vec_float4 getX() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 2D vector

## Description

Get the *x* element of a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE vec_float4 getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 2D vector

## Description

Get the *y* element of a 2D vector.

SCE CONFIDENTIAL

# setElem

Set an *x* or *y* element of a 2D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setElem(
                        int idx,
                        vec_float4_arg value
                    );
                }
            }
        }
    }
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-1.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set an *x* or *y* element of a 2D vector by specifying an index of 0 or 1 respectively.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setX(
                        vec_float4_arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set the *x* element of a 2D vector to the specified scalar value.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 2D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector2 {
                    SCE_VECTORMATH_INLINE Vector2 &setY(
                        vec_float4_arg y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 2D vector

## Description

Set the  $y$  element of a 2D vector to the specified scalar value.

**sce::Vectormath::Simd::Soa::Vector3**

000004892117

# Summary

## sce::Vectormath::Simd::Soa::Vector3

A set of four 3D vectors in structure-of-arrays format.

### Definition

```
#include <vectormath.h>
class Vector3 {};
```

### Description

A class representing a set of four 3D vectors stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 3D vectors.
<a href="#">getElem</a>	Get an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D vector by index.
<a href="#">getX</a>	Get the <i>x</i> element of a 3D vector.
<a href="#">getXY</a>	Get the <i>x</i> and <i>y</i> elements of a 3D vector.
<a href="#">getY</a>	Get the <i>y</i> element of a 3D vector.
<a href="#">getZ</a>	Get the <i>z</i> element of a 3D vector.
<a href="#">operator*</a>	Multiply a 3D vector by a scalar.
<a href="#">operator*=<sup>=</sup></a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator+<sup>=</sup></a>	Add two 3D vectors.
<a href="#">operator+<sup>=</sup></a>	Add a 3D vector to a 3D point.
<a href="#">operator+=<sup>=</sup></a>	Perform compound assignment and addition with a 3D vector.
<a href="#">operator-<sup>=</sup></a>	Subtract a 3D vector from another 3D vector.
<a href="#">operator-<sup>=</sup></a>	Negate all elements of a 3D vector.
<a href="#">operator--<sup>=</sup></a>	Perform compound assignment and subtraction by a 3D vector.
<a href="#">operator/<sup>=</sup></a>	Divide a 3D vector by a scalar.
<a href="#">operator/=<sup>=</sup></a>	Perform compound assignment and division by a scalar.
<a href="#">operator=</a>	Assign one 3D vector to another.
<a href="#">operator[]</a>	Subscripting operator to set or get an element.
<a href="#">operator[]</a>	Subscripting operator to get an element.
<a href="#">setElem</a>	Set an <i>x</i> , <i>y</i> , or <i>z</i> element of a 3D vector by index.
<a href="#">setX</a>	Set the <i>x</i> element of a 3D vector.
<a href="#">setXY</a>	Set the <i>x</i> and <i>y</i> elements of a 3D vector.
<a href="#">setY</a>	Set the <i>y</i> element of a 3D vector.
<a href="#">setZ</a>	Set the <i>z</i> element of a 3D vector.
<a href="#">Vector3</a>	Default constructor; does no initialization.
<a href="#">Vector3</a>	Copy constructor.
<a href="#">Vector3</a>	Construct a 3D vector from <i>x</i> , <i>y</i> , and <i>z</i> elements.
<a href="#">Vector3</a>	Construct a 3D vector from a 2D vector and a scalar.
<a href="#">Vector3</a>	Copy elements from a 3D point into a 3D vector.
<a href="#">Vector3</a>	Set all elements of a 3D vector to the same scalar value.
<a href="#">Vector3</a>	Replicate an AoS 3D vector.
<a href="#">Vector3</a>	Insert four AoS 3D vectors.
<a href="#">xAxis</a>	Construct x axis.
<a href="#">yAxis</a>	Construct y axis.

SCE CONFIDENTIAL

Methods	Description
<u><a href="#">zAxis</a></u>	Construct z axis.
<u><a href="#">zero</a></u>	Construct zero.

000004892117

# Constructors and Destructors

## Vector3

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector3();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector3

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector3(
                        const Vector3 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from  $x$ ,  $y$ , and  $z$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        vec_float4_arg x,
                        vec_float4_arg y,
                        vec_float4_arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.

## Return Values

None

## Description

Construct a 3D vector containing the specified  $x$ ,  $y$ , and  $z$  elements.

SCE CONFIDENTIAL

# Vector3

Construct a 3D vector from a 2D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        Vector2 arg xy,
                        vec_float4_arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value.

## Return Values

None

## Description

Construct a 3D vector. The *x* and *y* elements are set to those of the specified 2D vector, and the *z* element is set to the specified scalar.

SCE CONFIDENTIAL

# Vector3

Copy elements from a 3D point into a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        Point3 arg pnt
                    );
                };
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

None

## Description

Construct a 3D vector containing the *x*, *y*, and *z* elements of the specified 3D point.

SCE CONFIDENTIAL

# Vector3

Set all elements of a 3D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    explicit SCE_VECTORMATH_INLINE Vector3(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 3D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector3

Replicate an AoS 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        Aos::Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                    AoS 3D vector.

## Return Values

None

## Description

Replicate an AoS 3D vector in all four slots of an SoA 3D vector.

SCE CONFIDENTIAL

# Vector3

Insert four AoS 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3(
                        Aos::Vector3 arg vec0,
                        Aos::Vector3 arg vec1,
                        Aos::Vector3 arg vec2,
                        Aos::Vector3 arg vec3
                    );
                };
            }
        }
    }
}
```

## Arguments

vec0	AoS 3D vector.
vec1	AoS 3D vector.
vec2	AoS 3D vector.
vec3	AoS 3D vector.

## Return Values

None

## Description

Insert four AoS 3D vectors into four slots of an SoA 3D vector (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 3D vector by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

<i>scalar</i>	Scalar value.
---------------	---------------

### **Return Values**

Product of the specified 3D vector and scalar

### **Description**

Multiply a 3D vector by a scalar.

SCE CONFIDENTIAL

# operator\*= ---

Perform compound assignment and multiplication by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator*=(vec_float4_arg scalar
                );
            }
        }
    }
}
```

## Arguments

*scalar*              Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 3D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator+
                        (Vector3 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

Sum of the specified 3D vectors

## Description

Add two 3D vectors.

SCE CONFIDENTIAL

# operator+

Add a 3D vector to a 3D point.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Point3 operator+
                        (Point3 arg pnt
                     ) const;
                }
            }
        }
    }
}
```

## Arguments

*pnt*                    3D point.

## Return Values

Sum of the specified 3D vector and 3D point

## Description

Add a 3D vector to a 3D point.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator+=(
                        Vector3 arg
                    );
                };
            };
        };
    };
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and addition with a 3D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 3D vector from another 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-
                        (Vector3 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*vec*                    3D vector.

## **Return Values**

Difference of the specified 3D vectors

## **Description**

Subtract a 3D vector from another 3D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 3D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

3D vector containing negated elements of the specified 3D vector

## **Description**

Negate all elements of a 3D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator==(Vector3 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and subtraction by a 3D vector.

SCE CONFIDENTIAL

# operator/

Divide a 3D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE const Vector3 operator/(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified 3D vector and scalar

## Description

Divide a 3D vector by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator/=(
                        vec_float4_arg scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 3D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &operator=(Vector3 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            };
        };
    };
}
```

## **Arguments**

*vec*                   3D vector.

## **Return Values**

A reference to the resulting 3D vector

## **Description**

Assign one 3D vector to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 &operator[](int idx)
                    ;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

A reference to indexed element

## Description

Subscripting operator invoked when applied to non-const [Vector3](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 operator[](int idx) const;
                };
            };
        };
    };
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector3](#).

# Public Static Methods

## xAxis

Construct x axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 xAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 3D vector

### Description

Construct a 3D vector equal to (1,0,0).

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 yAxis();
                };
            };
        };
    };
}
```

## Arguments

None

## Return Values

The constructed 3D vector

## Description

Construct a 3D vector equal to (0,1,0).

SCE CONFIDENTIAL

# **zAxis**

Construct z axis.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 zAxis();
                };
            };
        };
    };
}
```

## **Arguments**

None

## **Return Values**

The constructed 3D vector

## **Description**

Construct a 3D vector equal to (0,0,1).

SCE CONFIDENTIAL

---

## zero

---

---

Construct zero.

### Definition

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    static SCE_VECTORMATH_INLINE const Vector3 zero();
                };
            };
        };
    };
}
```

### Arguments

---

None

### Return Values

---

The constructed 3D vector

### Description

---

Construct a 3D vector equal to (0,0,0).

# Public Instance Methods

## get4Aos

Extract four AoS 3D vectors.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Vector3 &result0,
                        Aos::Vector3 &result1,
                        Aos::Vector3 &result2,
                        Aos::Vector3 &result3
                    ) const;
                };
            };
        };
    };
}
```

### Arguments

<i>result0</i>	An output AoS 3D vector.
<i>result1</i>	An output AoS 3D vector.
<i>result2</i>	An output AoS 3D vector.
<i>result3</i>	An output AoS 3D vector.

### Return Values

None

### Description

Extract four AoS 3D vectors from four slots of an SoA 3D vector (transpose the data format).

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, or *z* element of a 3D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-2.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, or *z* element of a 3D vector by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 getX() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 3D vector

## Description

Get the *x* element of a 3D vector.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a 3D vector's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 3D vector

## Description

Get the *y* element of a 3D vector.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE vec_float4 getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 3D vector

## Description

Get the *z* element of a 3D vector.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, or *z* element of a 3D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setElem(
                        int idx,
                        vec_float4_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-2.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set an *x*, *y*, or *z* element of a 3D vector by specifying an index of 0, 1, or 2, respectively.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setX(
                        vec_float4_arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the *x* element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 3D vector

## Description

Set a 3D vector's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setY(
                        vec_float4_arg y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the  $y$  element of a 3D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 3D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector3 {
                    SCE_VECTORMATH_INLINE Vector3 &setZ(
                        vec_float4_arg z
                    );
                };
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 3D vector

## Description

Set the *z* element of a 3D vector to the specified scalar value.

**sce::Vectormath::Simd::Soa::Vector4**

000004892117

# Summary

# sce::Vectormath::Simd::Soa::Vector4

A set of four 4D vectors in structure-of-arrays format.

## Definition

```
#include <vectormath.h>
class Vector4 {};
```

## Description

A class representing a set of four 4D vectors stored in structure-of-arrays (SoA) format.

## Methods Summary

SCE CONFIDENTIAL

Methods	Description
<a href="#">Vector4</a>	Copy x, y, and z from a 3D point into a 4D vector, and set w to 1.
<a href="#">Vector4</a>	Copy elements from a quaternion into a 4D vector.
<a href="#">Vector4</a>	Set all elements of a 4D vector to the same scalar value.
<a href="#">Vector4</a>	Replicate an AoS 4D vector.
<a href="#">Vector4</a>	Insert four AoS 4D vectors.
<a href="#">wAxis</a>	Construct w axis.
<a href="#">xAxis</a>	Construct x axis.
<a href="#">yAxis</a>	Construct y axis.
<a href="#">zAxis</a>	Construct z axis.
<a href="#">zero</a>	Construct zero.

# Constructors and Destructors

## Vector4

Default constructor; does no initialization.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector4();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

None

### Description

Default constructor; does no initialization.

SCE CONFIDENTIAL

# Vector4

Copy constructor.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE Vector4(
                        const Vector4 &vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

None

## Description

Copy constructor.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        vec_float4_arg x,
                        vec_float4_arg y,
                        vec_float4_arg z,
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

$x$	Scalar value.
$y$	Scalar value.
$z$	Scalar value.
$w$	Scalar value.

## Return Values

None

## Description

Construct a 4D vector containing the specified  $x$ ,  $y$ ,  $z$ , and  $w$  elements.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from a 2D vector and two scalars.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector2 arg xy,
                        vec_float4_arg z,
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>z</i>	Scalar value.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a 4D vector. The *x* and *y* elements are set to those of the specified 2D vector, and the *z* and *w* elements are set to the specified scalar values.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from two 2D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector2 arg xy,
                        Vector2 arg zw
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xy</i>	2D vector.
<i>zw</i>	2D vector.

## Return Values

None

## Description

Construct a 4D vector. The *x* and *y* elements are set to those of the first specified 2D vector, and the *z* and *w* elements are set to the *x* and *y* elements of the second specified 2D vector.

SCE CONFIDENTIAL

# Vector4

Construct a 4D vector from a 3D vector and a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Vector3_arg xyz,
                        vec_float4_arg w
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>xyz</i>	3D vector.
<i>w</i>	Scalar value.

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to the specified scalar.

SCE CONFIDENTIAL

# Vector4

Copy x, y, and z from a 3D vector into a 4D vector, and set w to 0.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                   3D vector.

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D vector and with the *w* element set to 0.

SCE CONFIDENTIAL

# Vector4

Copy x, y, and z from a 3D point into a 4D vector, and set w to 1.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Point3 arg pnt
                    );
                };
            }
        }
    }
}
```

## Arguments

*pnt*                   3D point.

## Return Values

None

## Description

Construct a 4D vector with the *x*, *y*, and *z* elements of the specified 3D point and with the *w* element set to 1.

SCE CONFIDENTIAL

# Vector4

Copy elements from a quaternion into a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        Quat arg quat
                    );
                }
            }
        }
    }
}
```

## Arguments

*quat*                  Quaternion.

## Return Values

None

## Description

Construct a 4D vector containing the *x*, *y*, *z*, and *w* elements of the specified quaternion.

SCE CONFIDENTIAL

# Vector4

Set all elements of a 4D vector to the same scalar value.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    explicit SCE_VECTORMATH_INLINE Vector4(
                        vec_float4_arg scalar
                    );
                };
            }
        }
    }
}
```

## Arguments

<i>scalar</i>	Scalar value.
---------------	---------------

## Return Values

None
------

## Description

Construct a 4D vector with all elements set to the scalar value argument.

SCE CONFIDENTIAL

# Vector4

Replicate an AoS 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Aos::Vector4 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                    AoS 4D vector.

## Return Values

None

## Description

Replicate an AoS 4D vector in all four slots of an SoA 4D vector.

SCE CONFIDENTIAL

# Vector4

Insert four AoS 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4(
                        Aos::Vector4 arg vec0,
                        Aos::Vector4 arg vec1,
                        Aos::Vector4 arg vec2,
                        Aos::Vector4 arg vec3
                    );
                };
            }
        }
    }
}
```

## Arguments

vec0	AoS 4D vector.
vec1	AoS 4D vector.
vec2	AoS 4D vector.
vec3	AoS 4D vector.

## Return Values

None

## Description

Insert four AoS 4D vectors into four slots of an SoA 4D vector (transpose the data format).

# Operator Methods

## **operator\***

Multiply a 4D vector by a scalar.

### **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator*(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

### **Arguments**

*scalar*      Scalar value.

### **Return Values**

Product of the specified 4D vector and scalar

### **Description**

Multiply a 4D vector by a scalar.

SCE CONFIDENTIAL

# operator\*= ---

Perform compound assignment and multiplication by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator*=(  

                        vec_float4_arg scalar  

                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*              Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and multiplication by a scalar.

SCE CONFIDENTIAL

# operator+

Add two 4D vectors.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator+
                        (Vector4 arg) vec
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

Sum of the specified 4D vectors

## Description

Add two 4D vectors.

SCE CONFIDENTIAL

# operator+=

Perform compound assignment and addition with a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator+=(
                        Vector4 arg
                    );
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and addition with a 4D vector.

SCE CONFIDENTIAL

# **operator-**

Subtract a 4D vector from another 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator-
                        (Vector4 arg vec
                         ) const;
                }
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

Difference of the specified 4D vectors

## **Description**

Subtract a 4D vector from another 4D vector.

SCE CONFIDENTIAL

# **operator-**

Negate all elements of a 4D vector.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator-() const;
                }
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

4D vector containing negated elements of the specified 4D vector

## **Description**

Negate all elements of a 4D vector.

SCE CONFIDENTIAL

# operator==

Perform compound assignment and subtraction by a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator==(Vector4 arg) vec
                };
            }
        }
    }
}
```

## Arguments

vec                  4D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and subtraction by a 4D vector.

SCE CONFIDENTIAL

# operator/

Divide a 4D vector by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector4 operator/(
                        vec_float4_arg scalar
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

Quotient of the specified 4D vector and scalar

## Description

Divide a 4D vector by a scalar.

SCE CONFIDENTIAL

# operator/=

Perform compound assignment and division by a scalar.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator/=(
                        vec_float4_arg scalar
                    );
                }
            }
        }
    }
}
```

## Arguments

*scalar*      Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Perform compound assignment and division by a scalar.

SCE CONFIDENTIAL

# **operator=**

Assign one 4D vector to another.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &operator=(Vector4 arg)
                    {
                        vec = arg;
                        return *this;
                    }
                };
            }
        }
    }
}
```

## **Arguments**

*vec*                  4D vector.

## **Return Values**

A reference to the resulting 4D vector

## **Description**

Assign one 4D vector to another.

SCE CONFIDENTIAL

# operator[]

Subscripting operator to set or get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 &operator[](int idx)
                    ;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

A reference to indexed element

## Description

Subscripting operator invoked when applied to non-const [Vector4](#).

SCE CONFIDENTIAL

# operator[]

Subscripting operator to get an element.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 operator[](int idx) const;
                };
            };
        };
    };
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Indexed element

## Description

Subscripting operator invoked when applied to const [Vector4](#).

# Public Static Methods

## wAxis

Construct w axis.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 wAxis();
                }
            }
        }
    }
}
```

### Arguments

None

### Return Values

The constructed 4D vector

### Description

Construct a 4D vector equal to (0,0,0,1).

SCE CONFIDENTIAL

# xAxis

Construct x axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 xAxis();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4D vector

## Description

Construct a 4D vector equal to (1,0,0,0).

# yAxis

Construct y axis.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 yAxis();
                };
            }
        }
    }
}
```

## Arguments

None

## Return Values

The constructed 4D vector

## Description

Construct a 4D vector equal to (0,1,0,0).

SCE CONFIDENTIAL

# **zAxis**

Construct z axis.

## **Definition**

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 zAxis();
                };
            }
        }
    }
}
```

## **Arguments**

None

## **Return Values**

The constructed 4D vector

## **Description**

Construct a 4D vector equal to (0,0,1,0).

SCE CONFIDENTIAL

---

## zero

---

---

Construct zero.

### Definition

---

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    static SCE_VECTORMATH_INLINE const Vector4 zero();
                };
            };
        };
    };
}
```

### Arguments

---

None

### Return Values

---

The constructed 4D vector

### Description

---

Construct a 4D vector equal to (0,0,0,0).

# Public Instance Methods

## get4Aos

Extract four AoS 4D vectors.

### Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE void get4Aos(
                        Aos::Vector4 &result0,
                        Aos::Vector4 &result1,
                        Aos::Vector4 &result2,
                        Aos::Vector4 &result3
                    ) const;
                };
            }
        }
    }
}
```

### Arguments

<i>result0</i>	An output AoS 4D vector.
<i>result1</i>	An output AoS 4D vector.
<i>result2</i>	An output AoS 4D vector.
<i>result3</i>	An output AoS 4D vector.

### Return Values

None

### Description

Extract four AoS 4D vectors from four slots of an SoA 4D vector (transpose the data format).

SCE CONFIDENTIAL

# getElem

Get an *x*, *y*, *z*, or *w* element of a 4D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 getElem(
                        int idx
                    ) const;
                }
            }
        }
    }
}
```

## Arguments

*idx* Index, expected in the range 0-3.

## Return Values

Element selected by the specified index

## Description

Get an *x*, *y*, *z*, or *w* element of a 4D vector by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# getW

Get the *w* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 getW() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*w* element of a 4D vector

## Description

Get the *w* element of a 4D vector.

SCE CONFIDENTIAL

# getX

Get the *x* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 getX() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*x* element of a 4D vector

## Description

Get the *x* element of a 4D vector.

SCE CONFIDENTIAL

# getXY

Get the *x* and *y* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_ALWAYS_INLINE const Vector2 getXY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

2D vector containing *x* and *y* elements

## Description

Extract a 4D vector's *x* and *y* elements into a 2D vector.

SCE CONFIDENTIAL

# getXYZ

Get the  $x$ ,  $y$ , and  $z$  elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE const Vector3 getXYZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

3D vector containing  $x$ ,  $y$ , and  $z$  elements

## Description

Extract a 4D vector's  $x$ ,  $y$ , and  $z$  elements into a 3D vector.

SCE CONFIDENTIAL

# getY

Get the *y* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 getY() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*y* element of a 4D vector

## Description

Get the *y* element of a 4D vector.

SCE CONFIDENTIAL

# getZ

Get the *z* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE vec_float4 getZ() const;
                }
            }
        }
    }
}
```

## Arguments

None

## Return Values

*z* element of a 4D vector

## Description

Get the *z* element of a 4D vector.

SCE CONFIDENTIAL

# setElem

Set an *x*, *y*, *z*, or *w* element of a 4D vector by index.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setElem(
                        int idx,
                        vec_float4_arg value
                    );
                };
            };
        };
    };
}
```

## Arguments

<i>idx</i>	Index, expected in the range 0-3.
<i>value</i>	Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set an *x*, *y*, *z*, or *w* element of a 4D vector by specifying an index of 0, 1, 2, or 3, respectively.

SCE CONFIDENTIAL

# setW

Set the *w* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setW(
                        vec_float4_arg w
                    );
                }
            }
        }
    }
}
```

## Arguments

*w*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *w* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setX

Set the *x* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setX(
                        vec_float4_arg x
                    );
                }
            }
        }
    }
}
```

## Arguments

*x*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *x* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setXY

Set the *x* and *y* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setXY(
                        Vector2 arg vec
                    );
                }
            }
        }
    }
}
```

## Arguments

*vec*                  2D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Set a 4D vector's *x* and *y* elements to those of the specified 2D vector.

## Notes

This function does not change the *z* and *w* elements.

SCE CONFIDENTIAL

# setXYZ

Set the *x*, *y*, and *z* elements of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setXYZ(
                        Vector3 arg vec
                    );
                };
            }
        }
    }
}
```

## Arguments

*vec*                   3D vector.

## Return Values

A reference to the resulting 4D vector

## Description

Set a 4D vector's *x*, *y*, and *z* elements to those of the specified 3D vector.

## Notes

This function does not change the *w* element.

SCE CONFIDENTIAL

# setY

Set the  $y$  element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setY(
                        vec_float4_arg y
                    );
                }
            }
        }
    }
}
```

## Arguments

$y$  Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the  $y$  element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

# setZ

Set the *z* element of a 4D vector.

## Definition

```
#include <vectormath.h>
namespace sce {
    namespace Vectormath {
        namespace Simd {
            namespace Soa {
                class Vector4 {
                    SCE_VECTORMATH_INLINE Vector4 &setZ(
                        vec_float4_arg z
                    );
                }
            }
        }
    }
}
```

## Arguments

*z*                   Scalar value.

## Return Values

A reference to the resulting 4D vector

## Description

Set the *z* element of a 4D vector to the specified scalar value.

SCE CONFIDENTIAL

---

# **Internal Functions**

# Absolute Value

## Functions

Calculates the absolute value of a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_abs(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_abs(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_abs(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_abs(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_abs(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_abs(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_abs(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_abs(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_abs(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_abs(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_abs(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_abs(
    vec_char16_arg a
);
```

SCE CONFIDENTIAL

---

## Arguments

---

[in]  $\alpha$  The vector to calculate the absolute value for.

---

## Return Values

---

A vector with lanes containing the absolute value of  $\alpha$ .

---

## Notes

---

For floating point values, this may be just a bit manipulation with no INF/NAN handling.

# Addition

## Functions

Adds two vectors together: (a + b).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_add(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_add(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_add(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_add(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_add(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_add(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_add(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_add(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_add(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_add(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_add(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_add(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_add(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_add(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_add(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_add(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_add(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_add(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_add(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_add(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## **Arguments**

- |               |                    |
|---------------|--------------------|
| [in] <i>a</i> | The first vector.  |
| [in] <i>b</i> | The second vector. |

## **Return Values**

The sum of the supplied vectors.

# Bit-Shift Left (Logical per Vector Lane)

## Functions

Performs a logical bit-shift left on a per lane basis.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sll(
    vec_llong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sll(
    vec_llong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sll(
    vec_ullong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sll(
    vec_ullong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sll(
    vec_int2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sll(
    vec_int4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sll(
    vec_uint2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sll(
    vec_uint4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sll(
    vec_short4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sll(
    vec_short8_arg v,
    vec_ushort8_arg s
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sll(
    vec_ushort4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sll(
    vec_ushort8_arg v,
    vec_ushort8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sll(
    vec_char8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sll(
    vec_char16_arg v,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sll(
    vec_uchar8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sll(
    vec_uchar16_arg v,
    vec_uchar16_arg s
);

```

## Arguments

[in] v	The vector to shift left.
[in] s	A vector which specifies how much to shift each lane by.

## Return Values

The shifted vector.

## Description

Performs a logical bit-shift left on a per lane basis.

# Bit-Shift Left (Logical)

## Functions

Performs a logical bit-shift left by a specified scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sll(
    vec_llong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sll(
    vec_llong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sll(
    vec_ullong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sll(
    vec_ullong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sll(
    vec_int2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sll(
    vec_int4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sll(
    vec_uint2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sll(
    vec_uint4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sll(
    vec_short4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sll(
    vec_short8_arg v,
    uint32_t s
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sll(  
    vec_ushort4_arg v,  
    uint32_t s  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sll(  
    vec_ushort8_arg v,  
    uint32_t s  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sll(  
    vec_char8_arg v,  
    uint32_t s  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sll(  
    vec_char16_arg v,  
    uint32_t s  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sll(  
    vec_uchar8_arg v,  
    uint32_t s  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sll(  
    vec_uchar16_arg v,  
    uint32_t s  
) ;
```

## Arguments

- |        |                                    |
|--------|------------------------------------|
| [in] v | The vector to shift left.          |
| [in] s | The scalar value to shift left by. |

## Return Values

The shifted vector.

## Description

Performs a logical bit-shift left by a specified scalar value.

# Bit-Shift Left (Whole Vector Logical)

## Functions

Performs a whole vector logical bit-shift left by a specified scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sllw(
    vec_uint2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sllw(
    vec_uint4_arg v,
    uint32_t s
);
```

### Arguments

[in] <i>v</i>	The vector to shift left.
[in] <i>s</i>	The scalar value to shift left by.

### Return Values

The shifted vector.

### Description

Performs a whole vector logical bit-shift left by a specified scalar value.

# Bit-Shift Right (Arithmetic per Vector Lane)

## Functions

Performs an arithmetic bit-shift right on a per lane basis.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sra(
    vec_llong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sra(
    vec_llong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sra(
    vec_ullong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sra(
    vec_ullong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sra(
    vec_int2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sra(
    vec_int4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sra(
    vec_uint2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sra(
    vec_uint4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sra(
    vec_short4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sra(
    vec_short8_arg v,
    vec_ushort8_arg s
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sra(
    vec_ushort4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sra(
    vec_ushort8_arg v,
    vec_ushort8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sra(
    vec_char8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sra(
    vec_char16_arg v,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sra(
    vec_uchar8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sra(
    vec_uchar16_arg v,
    vec_uchar16_arg s
);

```

## Arguments

[in] v	The vector to shift right.
[in] s	A vector which specifies how much to shift each lane by.

## Return Values

The shifted vector.

## Description

Performs an arithmetic bit-shift right on a per lane basis.

# Bit-Shift Right (Logical per Vector Lane)

## Functions

Performs a logical bit-shift right on a per lane basis.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_srl(
    vec_llong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_srl(
    vec_llong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_srl(
    vec_ullong1_arg v,
    vec_ullong1_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_srl(
    vec_ullong2_arg v,
    vec_ullong2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_srl(
    vec_int2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_srl(
    vec_int4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_srl(
    vec_uint2_arg v,
    vec_uint2_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_srl(
    vec_uint4_arg v,
    vec_uint4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_srl(
    vec_short4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_srl(
    vec_short8_arg v,
    vec_ushort8_arg s
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_srl(
    vec_ushort4_arg v,
    vec_ushort4_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_srl(
    vec_ushort8_arg v,
    vec_ushort8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_srl(
    vec_char8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_srl(
    vec_char16_arg v,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_srl(
    vec_uchar8_arg v,
    vec_uchar8_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_srl(
    vec_uchar16_arg v,
    vec_uchar16_arg s
);

```

## Arguments

- |        |  |
|--------|--|
| [in] v | The vector to shift right.                               |
| [in] s | A vector which specifies how much to shift each lane by. |

## Return Values

The shifted vector.

## Description

Performs a logical bit-shift right on a per lane basis.

# Bit-Shift Right (Logical)

## Functions

Performs a logical bit-shift right by a specified scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_srl(
    vec_llong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_srl(
    vec_llong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_srl(
    vec_ullong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_srl(
    vec_ullong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_srl(
    vec_int2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_srl(
    vec_int4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_srl(
    vec_uint2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_srl(
    vec_uint4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_srl(
    vec_short4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_srl(
    vec_short8_arg v,
    uint32_t s
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_srl(
    vec_ushort4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_srl(
    vec_ushort8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_srl(
    vec_char8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_srl(
    vec_char16_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_srl(
    vec_uchar8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_srl(
    vec_uchar16_arg v,
    uint32_t s
);
```

## Arguments

- |        |                                     |
|--------|-------------------------------------|
| [in] v | The vector to shift right.          |
| [in] s | The scalar value to shift right by. |

## Return Values

The shifted vector.

## Description

Performs a logical bit-shift right by a specified scalar value.

# Bit-Shift Right (Whole Vector Logical)

## Functions

Performs a whole vector logical bit-shift right by a specified scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_srlw(
    vec_uint2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_srlw(
    vec_uint4_arg v,
    uint32_t s
);
```

### Arguments

[in] <i>v</i>	The vector to shift right.
[in] <i>s</i>	The scalar value to shift right by.

### Return Values

The shifted vector.

### Description

Performs a whole vector logical bit-shift right by a specified scalar value.

# Bit-Shifting Right (Arithmetic)

## Functions

Performs an arithmetic bit-shift right by a specified scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sra(
    vec_llong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sra(
    vec_llong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sra(
    vec_ullong1_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sra(
    vec_ullong2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sra(
    vec_int2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sra(
    vec_int4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sra(
    vec_uint2_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sra(
    vec_uint4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sra(
    vec_short4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sra(
    vec_short8_arg v,
    uint32_t s
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sra(
    vec_ushort4_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sra(
    vec_ushort8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sra(
    vec_char8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sra(
    vec_char16_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sra(
    vec_uchar8_arg v,
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sra(
    vec_uchar16_arg v,
    uint32_t s
);
```

## **Arguments**

[in] v	The vector to shift right.
[in] s	The scalar value to shift right by.

## **Return Values**

The shifted vector.

## **Description**

Performs an arithmetic bit-shift right by a specified scalar value.

# Bitwise AND

## Functions

Performs a bitwise AND (a & b) operation on two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_and(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_and(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_and(
    vec_double1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_and(
    vec_double2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_and(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_and(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_and(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_and(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_and(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_and(
    vec_float4_arg a,
    vec_float4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_and(
    vec_float2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_and(
    vec_float4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_and(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_and(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_and(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_and(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_and(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_and(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_and(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_and(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_and(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_and(
    vec_char16_arg a,
    vec_char16_arg b
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_and(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_and(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);
```

## Arguments

- |               |                    |
|---------------|--------------------|
| [in] <i>a</i> | The first vector.  |
| [in] <i>b</i> | The second vector. |

## Return Values

The bitwise AND of the supplied vectors.

## Description

Performs a bitwise AND (*a* & *b*) operation on two vectors.

# Bitwise AND (with scalar value)

## Functions

Performs a bitwise AND ( $a \& b$ ) operation between a vector and a scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_and(
    vec_llong1_arg a,
    int64_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_and(
    vec_llong2_arg a,
    int64_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_and(
    vec_ullong1_arg a,
    uint64_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_and(
    vec_ullong2_arg a,
    uint64_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_and(
    vec_int2_arg a,
    int32_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_and(
    vec_int4_arg a,
    int32_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_and(
    vec_uint2_arg a,
    uint32_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_and(
    vec_uint4_arg a,
    uint32_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_and(
    vec_short4_arg a,
    int16_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_and(
    vec_short8_arg a,
    int16_t b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_and(
    vec_ushort4_arg a,
    uint16_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_and(
    vec_ushort8_arg a,
    uint16_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_and(
    vec_char8_arg a,
    int8_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_and(
    vec_char16_arg a,
    int8_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_and(
    vec_uchar8_arg a,
    uint8_t b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_and(
    vec_uchar16_arg a,
    uint8_t b
);

```

## Arguments

- |               |                          |
|---------------|--------------------------|
| [in] <i>a</i> | The vector to AND.       |
| [in] <i>b</i> | The scalar value to AND. |

## Return Values

The bitwise AND of the supplied vector and scalar value splatted to a vector of the same size.

## Description

Performs a bitwise AND (*a* & *b*) operation between a vector and a scalar value.

# Bitwise COMPLEMENT

## Functions

Performs a bitwise COMPLEMENT ( $\sim a$ ) operation on a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_not(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_not(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_not(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_not(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_not(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_not(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_not(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_not(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_not(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_not(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_not(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_not(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_not(
```

SCE CONFIDENTIAL

```

    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_not(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_not(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_not(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_not(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_not(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_not(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_not(
    vec_uchar16_arg a
);

```

## Arguments

[in] *a* The vector to perform the operation on.

## Return Values

The bitwise COMPLEMENT of the supplied vector.

## Description

Performs a bitwise COMPLEMENT ( $\sim a$ ) operation on a vector.

# Bitwise COMPLEMENT AND

## Functions

Performs a bitwise COMPLEMENT AND ( $a \& \sim b$ ) on two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_andc(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_andc(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_andc(
    vec_double1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_andc(
    vec_double2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_andc(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_andc(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_andc(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_andc(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_andc(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_andc(
    vec_float4_arg a,
    vec_float4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_andc(
    vec_float2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_andc(
    vec_float4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_andc(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_andc(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_andc(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_andc(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_andc(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_andc(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_andc(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_andc(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_andc(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_andc(
    vec_char16_arg a,
    vec_char16_arg b
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_andc(  
    vec_uchar8_arg a,  
    vec_uchar8_arg b  
) ;
```

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_andc(  
    vec_uchar16_arg a,  
    vec_uchar16_arg b  
) ;
```

## Arguments

- |               |                                      |
|---------------|--------------------------------------|
| [in] <i>a</i> | The first vector to COMPLEMENT AND.  |
| [in] <i>b</i> | The second vector to COMPLEMENT AND. |

## Return Values

The bitwise COMPLEMENT AND of the supplied vectors.

## Description

Performs a bitwise COMPLEMENT AND (*a* &  $\sim$ *b*) on two vectors.

# Bitwise COMPLEMENT OR

## Functions

Performs a bitwise COMPLEMENT OR ( $a \mid \sim b$ ) on two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_orc(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_orc(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_orc(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_orc(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_orc(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_orc(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_orc(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_orc(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_orc(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_orc(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_orc(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_orc(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_orc(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_orc(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_orc(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_orc(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_orc(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_orc(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_orc(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_orc(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## **Arguments**

- |               |                                     |
|---------------|-------------------------------------|
| [in] <i>a</i> | The first vector to COMPLIMENT OR.  |
| [in] <i>b</i> | The second vector to COMPLIMENT OR. |

## **Return Values**

The bitwise COMPLEMENT OR of the supplied vectors.

SCE CONFIDENTIAL

---

**Description**

---

Performs a bitwise COMPLEMENT OR ( $a \mid \sim b$ ) on two vectors.

000004892117

# Bitwise OR

## Functions

Performs a bitwise OR ( $a \mid b$ ) on two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_or(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_or(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_or(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_or(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_or(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_or(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_or(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_or(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_or(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_or(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_or(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_or(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_or(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_or(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_or(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_or(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_or(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_or(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_or(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_or(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- |               |                          |
|---------------|--------------------------|
| [in] <i>a</i> | The first vector to OR.  |
| [in] <i>b</i> | The second vector to OR. |

## Return Values

The bitwise OR of the supplied vectors.

SCE CONFIDENTIAL

---

**Description**

---

Performs a bitwise OR (a | b) on two vectors.

000004892117

# Bitwise XOR

## Functions

Performs a bitwise XOR ( $a \wedge b$ ) on two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_xor(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_xor(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_xor(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_xor(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_xor(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_xor(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_xor(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xor(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_xor(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xor(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_xor(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xor(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_xor(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_xor(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_xor(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_xor(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_xor(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_xor(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_xor(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_xor(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- |               |                           |
|---------------|---------------------------|
| [in] <i>a</i> | The first vector to XOR.  |
| [in] <i>b</i> | The second vector to XOR. |

## Return Values

The bitwise XOR of the supplied vectors.

SCE CONFIDENTIAL

---

**Description**

---

Performs a bitwise XOR ( $a \wedge b$ ) on two vectors.

000004892117

# Boolean Check (All Lanes True - Bool Result)

## Functions

Returns true if all lanes are true; otherwise false is returned. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_trueb(
    vec uchar16_arg a
);
```

### Arguments

[in] a      The vector to perform the boolean check on.

### Return Values

A value of true if all lanes are true; otherwise false is returned.

### Description

Returns true if all lanes are true; otherwise false is returned. The function assumes zero/all bits for false/true.

# Boolean Check (All Lanes True)

## Functions

Sets all lanes to true if all lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_true(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_true(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_true(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_true(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_true(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_true(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_true(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_true(
    vec_uchar16_arg a
);
```

### Arguments

[in] a      The vector to perform the boolean check on.

### Return Values

A vector with all lanes set to true if all lanes are true; otherwise all lanes are set to false.

### Description

Sets all lanes to true if all lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

# Boolean Check (Any Lane True)

## Functions

Sets all lanes to true if any lane is true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_anytrue(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_anytrue(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_anytrue(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_anytrue(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_anytrue(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_anytrue(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_anytrue(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_anytrue(
    vec_uchar16_arg a
);
```

### Arguments

[in] a      The vector to perform the boolean check on.

### Return Values

A vector with all lanes set to true if any lane is true; otherwise all lanes are set to false.

### Description

Sets all lanes to true if any lane is true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

# Boolean Check (Any Lanes True - Bool Result)

## Functions

Returns true if any lanes is true; otherwise false is returned. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE bool sce_vectormath_anytrueb(
    vec_uchar16_arg a
);
```

### Arguments

[in] a      The vector to perform the boolean check on.

### Return Values

A value of true if any lane is true; otherwise false is returned.

### Description

Returns true if any lanes is true; otherwise false is returned. The function assumes zero/all bits for false/true.

# Boolean Check (Partial Lanes All True)

## Functions

Sets all lanes to true if all of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_true2(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_true2(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_true3(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_true4(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* The vector to perform the boolean check on.

### Return Values

A vector with all lanes set to true if all of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false.

### Description

Sets all lanes to true if all of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

# Boolean Check (Partial Lanes Any True)

## Functions

Sets all lanes to true if any of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_anytrue2(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_anytrue2(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_anytrue3(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_anytrue4(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* The vector to perform the boolean check on.

### Return Values

A vector with all lanes set to true if any of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false.

### Description

Sets all lanes to true if any of the first 2, 3 or 4 lanes are true; otherwise all lanes are set to false. The function assumes zero/all bits for false/true.

# Byte Packing

## Functions

Packs bytes from one vector into another: (0,1,2,3,4,...) -> (0,4,.....).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_packlu8(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_packlu8(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* A vector containing the bytes to pack.

### Return Values

A vector containing the packed bytes.

### Description

Packs bytes from one vector into another: (0,1,2,3,4,...) -> (0,4,.....).

# Byte Unpacking

## Functions

Unpacks the lowest unsigned bytes from a vector: (0,1,2,3,...) -> ((uint)0,(uint)1,..) and zero extends each to an unsigned integer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_unpack1u8(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_unpack1u8(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* A vector containing the unsigned bytes to unpack.

### Return Values

A vector containing the unpacked zero extended unsigned integers.

### Description

Unpacks the lowest unsigned bytes from a vector: (0,1,2,3,...) -> ((uint)0,(uint)1,..) and zero extends each to an unsigned integer.

# Clearing

## Functions

Clears (zeroes) a specific lane of a vector. The remaining lanes are passed through unchanged.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_clearx(
    vec_float2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_clearx(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_clearx(
    vec_uint2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_clearx(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_clearx(
    vec_int2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_clearx(
    vec_int4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_cleary(
    vec_float2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_cleary(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cleary(
    vec_uint2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cleary(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_cleary(
    vec_int2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_cleary(
    vec_int4_arg v
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_clearz(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_clearz(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_clearz(
    vec_int4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_clearw(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_clearw(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_clearw(
    vec_int4_arg v
);
```

## Arguments

[in] v                   The vector to zero.

## Return Values

The vector with a specific lane cleared to zero.

## Description

Clears (zeroes) a specific lane of a vector. The remaining lanes are passed through unchanged.

# Combination

## Functions

Combines two vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_combine(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_combine(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_combine(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_combine(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_combine(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_combine(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_combine(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_combine(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_combine(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_combine(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);
```

SCE CONFIDENTIAL

---

## Arguments

---

- |          |   |
|----------|---|
| [in] $a$ | The vector to take the lower half from. |
| [in] $b$ | The vector to take the upper half from. |

---

## Return Values

---

The combined vector.

---

## Description

---

Combines two vectors.

# Comparing (Per-Lane Equal)

## Functions

Compares two vectors on a per-lane basis to see if they are equal.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpeq(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpeq(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpeq(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpeq(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpeq(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpeq(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpeq(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpeq(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpeq(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpeq(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpeq(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpeq(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpeq(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpeq(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpeq(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpeq(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpeq(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpeq(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpeq(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpeq(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- [in] *a*                    The first vector to compare.  
 [in] *b*                    The second vector to compare.

## Return Values

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if they are equal.

000004892117

# Comparing (Per-Lane Greater-Than) Functions

## Functions

Compares two vectors on a per-lane basis to see if the first is greater than the other.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmplt(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_cmplt(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmplt(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmplt(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmplt(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmplt(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmplt(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmplt(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- [in] *a*                    The first vector to compare.  
 [in] *b*                    The second vector to compare.

## Return Values

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if the first is greater than the other.

000004892117

# Comparing (Per-Lane Greater-Than-Or-Equal)

## Functions

Compares two vectors on a per-lane basis to see if the first is greater than or equal to the other.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpge(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpge(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpge(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpge(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpge(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpge(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpge(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpge(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpge(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpge(
    vec_int4_arg a,
```

SCE CONFIDENTIAL

```

        vec_int4_arg b
    );

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpge(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpge(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpge(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpge(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpge(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpge(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpge(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpge(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpge(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpge(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- [in] *a*                   The first vector to compare.  
 [in] *b*                   The second vector to compare.

## Return Values

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if the first is greater than or equal to the other.

000004892117

# Comparing (Per-Lane Less-Than-Or-Equal) Functions

## Functions

Compares two vectors on a per-lane basis to see if the first is less than or equal to the other.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmple(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmple(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmple(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmple(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmple(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmple(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmple(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmple(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmple(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmple(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmple(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmple(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmple(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmple(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmple(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmple(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmple(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmple(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmple(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmple(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- [in] *a*                    The first vector to compare.  
 [in] *b*                    The second vector to compare.

## Return Values

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if the first is less than or equal to the other.

000004892117

# Comparing (Per-Lane Not-Equal)

## Functions

Compares two vectors on a per-lane basis to see if they are not equal.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpne(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpne(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpne(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpne(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmpne(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmpne(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpne(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpne(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpne(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpne(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmpne(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmpne(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpne(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpne(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmpne(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmpne(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpne(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpne(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmpne(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmpne(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## **Arguments**

- [in] *a*                    The first vector to compare.  
 [in] *b*                    The second vector to compare.

## **Return Values**

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if they are not equal.

000004892117

# Comparing (Per-lane Less-Than) Functions

## Functions

Compares two vectors on a per-lane basis to see if the first is less than the other.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cmplt(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cmplt(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cmplt(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cmplt(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmplt(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmplt(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cmplt(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cmplt(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmplt(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmplt(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cmplt(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cmplt(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## **Arguments**

- |               |                               |
|---------------|-------------------------------|
| [in] <i>a</i> | The first vector to compare.  |
| [in] <i>b</i> | The second vector to compare. |

## **Return Values**

The result of the comparison. For each lane, zero/all bits are used for false/true.

SCE CONFIDENTIAL

**Description**

---

Compares two vectors on a per-lane basis to see if the first is less than the other.

000004892117

# Conversion (Comparison Result to 0.0/1.0 Single-Precision Floating Point Vector)

## Functions

Converts a zero/all bits comparison result to a 0.0/1.0 single-precision floating point vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_btof(
    vec_uint2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_btof(
    vec_uint4_arg a
) ;
```

### Arguments

[in] *a* An unsigned long integer vector (zero/all bits comparison result).

### Return Values

The converted single-precision floating point vector.

### Description

Converts a zero/all bits comparison result to a 0.0/1.0 single-precision floating point vector.

# Conversion (Comparison Result to Double-Precision Floating Point Vector)

## Functions

Converts a zero/all bits comparison result to a 0.0/1.0 double-precision floating point vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_btod(
    vec_ullong1_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_btod(
    vec_ullong2_arg a
) ;
```

### Arguments

[in] *a* An unsigned long integer vector (zero/all bits comparison result).

### Return Values

The converted double-precision floating point vector.

### Description

Converts a zero/all bits comparison result to a 0.0/1.0 double-precision floating point vector.

# Conversion (Floating Point Vector to 32-bit Signed Integer Vector)

## Functions

Converts a floating point vector to a 32-bit signed integer vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_cts(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_cts(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_cts(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_cts(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_cts(
    vec_double2_arg a,
    vec_double2_arg b
);
```

### Arguments

- [in] *a* A double-precision floating point or single-precision floating point vector.
- [in] *b* A double-precision floating point vector.

### Return Values

The converted 32-bit signed integer vector.

### Description

Converts a floating point vector to a 32-bit signed integer vector.

# Conversion (Floating Point Vector to 32-bit Unsigned Integer Vector)

## Functions

Converts a floating point vector to a 32-bit unsigned integer vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_ctu(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_ctu(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ctu(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_ctu(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ctu(
    vec_double2_arg a,
    vec_double2_arg b
);
```

### Arguments

- [in] *a* A double-precision floating point or single-precision floating point vector.
- [in] *b* A double-precision floating point vector.

### Return Values

The converted 32-bit unsigned integer vector.

### Description

Converts a floating point vector to a 32-bit unsigned integer vector.

# Conversion (Half-Precision Floating Point Vector to Single-Precision Floating Point Vector)

## Functions

Converts a whole, lower half or upper half half-precision floating point vector to a single-precision floating point vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf_low(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf_low(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf_high(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf_high(
    vec_ushort8_arg a
);
```

### Arguments

[in] *a* A half-precision floating point vector.

### Return Values

The converted single-precision floating point vector.

### Description

Converts a whole, lower half or upper half half-precision floating point vector to a single-precision floating point vector.

# Conversion (Single-Precision Floating Point Vector to Half-Precision Floating Point Vector)

## Functions

Converts from a single-precision floating vector to a half-precision floating vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cthf(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cthf(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cthf(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

[in] <i>a</i>	A single-precision floating point vector.
[in] <i>b</i>	A single-precision floating point vector.

### Return Values

The converted half-precision floating point vector.

### Description

Converts from a single-precision floating vector to a half-precision floating vector.

# Conversion (Vector to Double-Precision Floating Point Vector)

## Functions

Converts whole, lower half or upper half vectors to double-precision floating point vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_low(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_low(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_low(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_low(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_low(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_low(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_high(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_high(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_high(
    vec_int2_arg a
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_high(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_ctd_high(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_ctd_high(
    vec_uint4_arg a
);
```

## **Arguments**

[in] *a* A single-precision floating point, signed integer or unsigned integer vector.

## **Return Values**

The converted double-precision floating point vector.

## **Description**

Converts whole, lower half or upper half vectors to double-precision floating point vectors.

# Conversion (Vector to Single-Precision Floating Point Vector)

## Functions

Converts a vector to a single-precision floating point vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ctf(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ctf(
    vec_double2_arg a,
    vec_double2_arg b
);
```

### Arguments

- [in] a A double-precision floating point, signed integer or unsigned integer vector.
- [in] b A double-precision floating point vector.

### Return Values

The converted single-precision floating point vector.

### Description

Converts a vector to a single-precision floating point vector.

# Copy Sign

## Functions

Copies the sign of one vector to another.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_copysign(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_copysign(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_copysign(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_copysign(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

- [in] *a* The magnitude vector.
- [in] *b* The sign vector.

### Return Values

A vector with lanes containing the absolute magnitude of *a* and the sign of input *b*.

### Description

Copies the sign of one vector to another.

### Notes

This may be just a bit manipulation with no INF/NAN handling.

# Counting (Leading Zero Bits)

## Functions

Counts the number of leading zeros per-lane. Zero inputs are correctly managed.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_ctlz(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_ctlz(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_ctlz(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ctlz(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_ctlz(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_ctlz(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_ctlz(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_ctlz(
    vec_uchar16_arg a
);
```

### Arguments

[in] *a* The vector to count.

### Return Values

A vector containing the number of leading zero bits in each lane of the supplied vector.

### Description

Counts the number of leading zeros per-lane. Zero inputs are correctly managed.

# Counting (Set Bits)

## Functions

Counts the number of set bits (population count) per-lane. Zero inputs are correctly managed.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_popcnt(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_popcnt(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_popcnt(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_popcnt(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_popcnt(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_popcnt(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_popcnt(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_popcnt(
    vec_uchar16_arg a
);
```

### Arguments

[in] *a* The vector to count.

### Return Values

A vector containing the number of set bits in each lane of the supplied vector.

### Description

Counts the number of set bits (population count) per-lane. Zero inputs are correctly managed.

# Counting (Trailing Zero Bits)

## Functions

Counts the number of trailing zeros per-lane. Zero inputs are correctly managed.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_cnttz(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_cnttz(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_cnttz(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cnttz(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_cnttz(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_cnttz(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_cnttz(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_cnttz(
    vec_uchar16_arg a
);
```

### Arguments

[in] *a* The vector to count.

### Return Values

A vector containing the number of trailing zero bits in each lane of the supplied vector.

### Description

Counts the number of trailing zeros per-lane. Zero inputs are correctly managed.

# Creation (From Transposed Lanes)

## Functions

Creates a vector from the transposed lanes of up to four vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_transpose2x(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_transpose2x(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_transpose2x(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_transpose2y(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_transpose2y(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_transpose2y(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose3x(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose3x(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose3x(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose3y(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose3y(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose3y(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose3w(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose3w(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose3w(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose4x(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c,
    vec_float4_arg d
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose4x(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c,
    vec_int4_arg d
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose4x(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c,
    vec_uint4_arg d
);

#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose4y(
    vec_float4_arg a,
    vec_float4_arg b,

```

SCE CONFIDENTIAL

```

    vec_float4_arg c,
    vec_float4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose4y(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c,
    vec_int4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose4y(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c,
    vec_uint4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose4z(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c,
    vec_float4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose4z(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c,
    vec_int4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose4z(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c,
    vec_uint4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_transpose4w(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c,
    vec_float4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_transpose4w(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c,
    vec_int4_arg d
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_transpose4w(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c,
    vec_uint4_arg d
) ;

```

SCE CONFIDENTIAL

---

**Arguments**

---

[in] <i>a</i>	The first vector.
[in] <i>b</i>	The second vector.
[in] <i>c</i>	The third vector.
[in] <i>d</i>	The fourth vector.

**Return Values**

---

A vector containing the specified transposed lanes from the supplied vectors.

**Description**

---

Creates a vector from the transposed lanes of up to four vectors.

# Creation (From Vector Lower Half)

## Functions

Creates a vector from the lower half of a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_low(
    vec_double2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_low(
    vec_llong2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_low(
    vec_ullong2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_low(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_low(
    vec_int4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_low(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_low(
    vec_short8_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_low(
    vec_ushort8_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_low(
    vec_char16_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_low(
    vec_uchar16_arg v
);
```

### Arguments

[in] v                  The whole vector to create the vector from.

### Return Values

A vector created from the lower half of the supplied whole vector.

SCE CONFIDENTIAL

**Description**

---

Creates a vector from the lower half of a vector.

000004892117

# Creation (From Vector Upper Half)

## Functions

Creates a vector from the upper half of a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_high(
    vec_double2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_high(
    vec_llong2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_high(
    vec_ullong2_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_high(
    vec_float4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_high(
    vec_int4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_high(
    vec_uint4_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_high(
    vec_short8_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_high(
    vec_ushort8_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_high(
    vec_char16_arg v
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_high(
    vec_uchar16_arg v
);
```

### Arguments

[in] v                  The whole vector to create the vector from.

### Return Values

A vector created from the upper half of the supplied whole vector.

SCE CONFIDENTIAL

---

**Description**

Creates a vector from the upper half of a vector.

000004892117

# Cross Product

## Functions

Calculates the vector cross product.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_cross(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

[in] <i>a</i>	The first vector.
[in] <i>b</i>	The second vector.

### Return Values

The vector cross product (with w lane undefined).

### Description

Calculates the vector cross product.

# Division

## Functions

Divides one vector by another: (a / b).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_div(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_div(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_div(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_div(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

[in] <i>a</i>	The first vector.
[in] <i>b</i>	The second vector.

### Return Values

The result of the supplied vectors.

### Description

Divides one vector by another: (a / b).

### Notes

The result may be a Newton-Raphson approximation.

# Dot Product

## Functions

Calculates the dot product of the vector. The functions are fast but only lane 0 is guaranteed.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_dot2(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_dot2(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot2(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot3(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot4(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

- [in] *a* The first vector.
- [in] *b* The second vector.

### Return Values

The dot product of the first 2, 3 or 4 lanes of the supplied vectors. Only lane 0 is guaranteed to contain the result; the rest of the lanes will be undefined.

### Description

Calculates the dot product of the vector. The functions are fast but only lane 0 is guaranteed.

# Dot Product (All Lane Splatting)

## Functions

Calculates the dot product of the vector and splats the results to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_dot2splat(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_dot2splat(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot2splat(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot3splat(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_dot4splat(
    vec_float4_arg a,
    vec_float4_arg b
);
```

### Arguments

- [in] *a* The first vector.
- [in] *b* The second vector.

### Return Values

The dot product of the first 2, 3 or 4 lanes of the supplied vectors splatted to all lanes.

### Description

Calculates the dot product of the vector and splats the results to all lanes.

# Endian Swapping

## Functions

Swaps each lane of a vector from big endian to small endian or vice versa.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_endianswap(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_endianswap(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_endianswap(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_endianswap(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_endianswap(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_endianswap(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_endianswap(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_endianswap(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_endianswap(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_endianswap(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_endianswap(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_endianswap(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_endianswap(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_endianswap(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_endianswap(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_endianswap(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_endianswap(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_endianswap(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_endianswap(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_endianswap(
    vec_uchar16_arg a
);
```

## Arguments

[in] *a* A vector to swap.

## Return Values

The endian swapped vector.

## Description

Swaps each lane of a vector from big endian to small endian or vice versa.

# Endian Swapping (Big Endian to Native Endian)

## Functions

Swaps each lane of a vector from big endian to native endian.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_frombe(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_frombe(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_frombe(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_frombe(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_frombe(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_frombe(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_frombe(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_frombe(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_frombe(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_frombe(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_frombe(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_frombe(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_frombe(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_frombe(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_frombe(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_frombe(
    vec_ushort8_arg a
);
```

## Arguments

[in] *a* A big endian vector.

## Return Values

The native endian vector.

## Description

Swaps each lane of a vector from big endian to native endian.

# Endian Swapping (Little Endian to Native Endian)

## Functions

Swaps each lane of a vector from little endian to native endian.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_fromle(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_fromle(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_fromle(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_fromle(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_fromle(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_fromle(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_fromle(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_fromle(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_fromle(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_fromle(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_fromle(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_fromle(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_fromle(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_fromle(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_fromle(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_fromle(
    vec_ushort8_arg a
);
```

## Arguments

[in] *a* A little endian vector.

## Return Values

The native endian vector.

## Description

Swaps each lane of a vector from little endian to native endian.

# Endian Swapping (Native Endian to BigEndian)

## Functions

Swaps each lane of a vector from native endian to big endian.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_tobe(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_tobe(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_tobe(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_tobe(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tobe(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tobe(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_tobe(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_tobe(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_tobe(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_tobe(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tobe(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tobe(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_tobe(  
    vec_short4_arg a  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_tobe(  
    vec_short8_arg a  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tobe(  
    vec_ushort4_arg a  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tobe(  
    vec_ushort8_arg a  
) ;
```

## Arguments

[in] *a* A native endian vector.

## Return Values

The big endian vector.

## Description

Swaps each lane of a vector from native endian to big endian.

# Endian Swapping (Native Endian to LittleEndian)

## Functions

Swaps each lane of a vector from native endian to little endian.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_tole(
    vec_double1_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_tole(
    vec_double2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_tole(
    vec_llong1_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_tole(
    vec_llong2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tole(
    vec_ullong1_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tole(
    vec_ullong2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_tole(
    vec_float2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_tole(
    vec_float4_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_tole(
    vec_int2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_tole(
    vec_int4_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tole(
    vec_uint2_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tole(
    vec_uint4_arg a
) ;
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_tole(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_tole(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tole(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tole(
    vec_ushort8_arg a
);
```

## Arguments

[in] *a* A native endian vector.

## Return Values

The little endian vector.

## Description

Swaps each lane of a vector from native endian to little endian.

# Extraction

## Functions

Extracts a value from a vector lane to a scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE double sce_vectormath_extract(
    vec_double1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE double sce_vectormath_extract(
    vec_double2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int64_t sce_vectormath_extract(
    vec_llong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int64_t sce_vectormath_extract(
    vec_llong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint64_t sce_vectormath_extract(
    vec_ullong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint64_t sce_vectormath_extract(
    vec_ullong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE float sce_vectormath_extract(
    vec_float2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE float sce_vectormath_extract(
    vec_float4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int32_t sce_vectormath_extract(
    vec_int2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int32_t sce_vectormath_extract(
    vec_int4_arg v,
    uint32_t slot
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE uint32_t sce_vectormath_extract(
    vec_uint2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint32_t sce_vectormath_extract(
    vec_uint4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int16_t sce_vectormath_extract(
    vec_short4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int16_t sce_vectormath_extract(
    vec_short8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint16_t sce_vectormath_extract(
    vec_ushort4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint16_t sce_vectormath_extract(
    vec_ushort8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int8_t sce_vectormath_extract(
    vec_char8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE int8_t sce_vectormath_extract(
    vec_char16_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint8_t sce_vectormath_extract(
    vec_uchar8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE uint8_t sce_vectormath_extract(
    vec_uchar16_arg v,
    uint32_t slot
);

```

## Arguments

- [in] *v* The vector to extract from.  
 [in] *slot* The index of the specific lane to extract from.

## Return Values

A scalar value from the specified lane of the supplied vector.

SCE CONFIDENTIAL

---

**Description**

---

Extracts a value from a vector lane to a scalar value.

000004892117

# Floating Point Exponent Extraction

## Functions

Extracts a floating point exponent vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_exponent(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_exponent(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_exponent(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_exponent(
    vec_float4_arg a
);
```

### Arguments

[in] *a* A floating point vector.

### Return Values

The floating point exponent vector.

### Description

Extracts a floating point exponent vector.

# Floating Point Sign Bit Splatting

## Functions

Splats a floating point sign bit vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sign(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sign(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sign(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sign(
    vec_float4_arg a
);
```

### Arguments

[in] *a* A floating point vector.

### Return Values

The splatted floating point sign bit vector (if negative all bits; otherwise no bits).

### Description

Splats a floating point sign bit vector.

# Geometric Length

## Functions

Calculates the geometric length of a vector:  $\sqrt{a[0]^2 + a[1]^2 + \dots}$ .

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_length(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_length(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_length(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_length(
    vec_float4_arg a
);
```

### Arguments

[in] *a* The first vector.

### Return Values

The geometric length of the vector.

### Description

Calculates the geometric length of a vector:  $\sqrt{a[0]^2 + a[1]^2 + \dots}$ .

### Notes

The result may be a Newton-Raphson approximation.

# Geometric Partial Length

## Functions

Calculates the partial geometric length of the first 2, 3 or 4 lanes of a vector:

$\text{sqrt}(a[0]*a[0]+a[1]*a[1]+\dots)$ .

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_length2(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_length2(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_length2(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_length3(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_length4(
    vec_float4_arg a
);
```

### Arguments

[in] *a* The first vector.

### Return Values

The partial geometric length of the first 2, 3 or 4 lanes of a vector.

### Description

Calculates the partial geometric length of the first 2, 3 or 4 lanes of a vector:  
 $\text{sqrt}(a[0]*a[0]+a[1]*a[1]+\dots)$ .

### Notes

The result may be a Newton-Raphson approximation.

# Horizontal Maximum Value

## Functions

Obtains the horizontal maximum value of the vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_max(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_max(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_max(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_max(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_max(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_max(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_max(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_max(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_max(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_max(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_max(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_max(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_max(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_max(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_max(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_max(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_max(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_max(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_max(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_max(
    vec_uchar16_arg a
);

```

## Arguments

[in] *a* The vector to obtain the horizontal maximum value for.

## Return Values

A vector in which every lane has the maximum horizontal value of the supplied vector.

## Description

Obtains the horizontal maximum value of the vector. This is then splatted to all lanes.

# Horizontal Minimum Value

## Functions

Obtains the horizontal minimum value of the vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_min(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_min(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_min(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_min(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_min(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_min(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_min(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_min(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_min(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_min(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_min(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_min(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_min(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_min(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_min(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_min(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_min(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_min(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_min(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_min(
    vec_uchar16_arg a
);

```

## Arguments

[in] *a* The vector to obtain the horizontal minimum value for.

## Return Values

A vector in which every lane has the minimum horizontal value of the supplied vector.

## Description

Obtains the horizontal minimum value of the vector. This is then splatted to all lanes.

# Horizontal Partial Maximum Value

## Functions

Obtains the horizontal maximum value from the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_max2(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_max2(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_max2(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_max2(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_max2(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_max2(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_max3(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_max3(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_max3(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_max4(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_max4(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_max4(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

---

## Arguments

---

[in]  $\alpha$  The vector to obtain the partial horizontal maximum value for.

---

## Return Values

---

A vector with the horizontal maximum value from the first 2, 3 or 4 lanes of the supplied vector splatted to all lanes.

---

## Description

---

Obtains the horizontal maximum value from the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

# Horizontal Partial Minimum Value

## Functions

Obtains the horizontal minimum value from the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_min2(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_min2(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_min2(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_min2(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_min2(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_min2(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_min3(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_min3(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_min3(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_min4(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_min4(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_min4(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

## **Arguments**

[in]  $\alpha$  The vector to obtain the partial horizontal minimum value for.

## **Return Values**

A vector with the horizontal minimum value from the first 2, 3 or 4 lanes of the supplied vector splatted to all lanes.

## **Description**

Obtains the horizontal minimum value from the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

# Horizontal Partial Sum

## Functions

Calculates the horizontal sum of the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_sum2(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sum2(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sum2(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sum2(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sum2(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sum2(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sum3(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sum3(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sum3(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sum4(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sum4(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sum4(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

---

## Arguments

---

[in]  $\alpha$

The vector to calculate the partial horizontal sum for.

---

## Return Values

---

The sum of the first 2, 3 or 4 lanes of the supplied vector splatted to all lanes.

---

## Description

---

Calculates the horizontal sum of the first 2, 3 or 4 lanes of a vector. This is then splatted to all lanes.

# Horizontal Sum

## Functions

Calculates the horizontal sum of a vector. This is then splatted to all lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_sum(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_sum(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sum(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sum(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sum(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sum(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_sum(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sum(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sum(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sum(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sum(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sum(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sum(
    vec_short4_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sum(
    vec_short8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sum(
    vec_ushort4_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sum(
    vec_ushort8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sum(
    vec_char8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sum(
    vec_char16_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sum(
    vec_uchar8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sum(
    vec_uchar16_arg a
) ;

```

## Arguments

[in] *a* The vector to calculate the horizontal sum for.

## Return Values

The horizontal sum of the supplied vector splatted to all lanes.

## Description

Calculates the horizontal sum of a vector. This is then splatted to all lanes.

# Insertion

## Functions

Inserts a scalar value into a specified vector lane.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_insert(
    double s,
    vec_double1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_insert(
    double s,
    vec_double2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_insert(
    int64_t s,
    vec_llong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_insert(
    int64_t s,
    vec_llong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_insert(
    uint64_t s,
    vec_ullong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_insert(
    uint64_t s,
    vec_ullong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_insert(
    float s,
    vec_float2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_insert(
    float s,
    vec_float4_arg v,
    uint32_t slot
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_insert(
    int32_t s,
    vec_int2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_insert(
    int32_t s,
    vec_int4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_insert(
    uint32_t s,
    vec_uint2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_insert(
    uint32_t s,
    vec_uint4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_insert(
    int16_t s,
    vec_short4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_insert(
    int16_t s,
    vec_short8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_insert(
    uint16_t s,
    vec_ushort4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_insert(
    uint16_t s,
    vec_ushort8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_insert(
    int8_t s,
    vec_char8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_insert(
    int8_t s,
    vec_char16_arg v,
    uint32_t slot
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_insert(
    uint8_t s,
    vec_uchar8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_insert(
    uint8_t s,
    vec_uchar16_arg v,
    uint32_t slot
);
```

## Arguments

[in] <i>s</i>	The scalar value to insert.
[in] <i>v</i>	The vector to insert the scalar value into.
[in] <i>slot</i>	The index of the specific lane to insert the scalar value into.

## Return Values

A vector with the supplied scalar value inserted into the specified lane.

## Description

Inserts a scalar value into a specified vector lane.

# Insertion (Take From First Lane)

## Functions

Inserts the first lane from a vector into the specified lane of another vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_insertv(
    vec_double1_arg s,
    vec_double1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_insertv(
    vec_double2_arg s,
    vec_double2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_insertv(
    vec_llong1_arg s,
    vec_llong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_insertv(
    vec_llong2_arg s,
    vec_llong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_insertv(
    vec_ullong1_arg s,
    vec_ullong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_insertv(
    vec_ullong2_arg s,
    vec_ullong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_insertv(
    vec_float2_arg s,
    vec_float2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_insertv(
    vec_float4_arg s,
    vec_float4_arg v,
    uint32_t slot
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_insertv(
    vec_int2_arg s,
    vec_int2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_insertv(
    vec_int4_arg s,
    vec_int4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_insertv(
    vec_uint2_arg s,
    vec_uint2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_insertv(
    vec_uint4_arg s,
    vec_uint4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_insertv(
    vec_short4_arg s,
    vec_short4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_insertv(
    vec_short8_arg s,
    vec_short8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_insertv(
    vec_ushort4_arg s,
    vec_ushort4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_insertv(
    vec_ushort8_arg s,
    vec_ushort8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_insertv(
    vec_char8_arg s,
    vec_char8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_insertv(
    vec_char16_arg s,
    vec_char16_arg v,
    uint32_t slot
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_insertv(
    vec_uchar8_arg s,
    vec_uchar8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_insertv(
    vec_uchar16_arg s,
    vec_uchar16_arg v,
    uint32_t slot
);
```

## Arguments

[in] <i>s</i>	The source vector.
[in] <i>v</i>	The destination vector.
[in] <i>slot</i>	The index of the specific lane in the destination vector to insert the source vector's first lane into.

## Return Values

The destination vector with the first lane of the source vector inserted into the specified lane.

## Description

Inserts the first lane from a vector into the specified lane of another vector.

# Integer Narrowing by Discarding Upper Bits

## Functions

Narrows an integer by discarding the upper bits.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_narrow(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_narrow(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_narrow(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_narrow(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_narrow(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_narrow(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_narrow(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_narrow(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_narrow(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_narrow(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_narrow(
    vec_int2_arg a,
    vec_int2_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_narrow(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_narrow(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_narrow(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_narrow(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_narrow(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_narrow(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_narrow(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

```

## Arguments

- [in] *a* A signed/unsigned integer vector.
- [in] *b* A signed/unsigned integer vector.

## Return Values

The narrowed signed/unsigned integer vector.

## Description

Narrows an integer by discarding the upper bits.

# Integer Narrowing by Value Saturation

## Functions

Narrows integers by saturating the vector to the target integer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_narrow_sat(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_narrow_sat(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_narrow_sat(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_narrow_sat(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_narrow_sat(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_narrow_sat(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_narrow_sat(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_narrow_sat(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_narrow_sat(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_narrow_sat(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_narrow_sat(
    vec_int2_arg a,
    vec_int2_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_narrow_sat(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_narrow_sat(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_narrow_sat(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_narrow_sat(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_narrow_sat(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_narrow_sat(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_narrow_sat(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

```

## Arguments

- [in] *a* A signed/unsigned integer vector.
- [in] *b* A signed/unsigned integer vector.

## Return Values

The saturated, narrowed signed/unsigned integer vector.

## Description

Narrows integers by saturating the vector to the target integer.

# Lane Order Reversal

## Functions

Reverses the order of a vector's lanes.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_orderswap(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_orderswap(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_orderswap(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_orderswap(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_orderswap(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_orderswap(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_orderswap(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_orderswap(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_orderswap(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_orderswap(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_orderswap(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_orderswap(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_orderswap(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_orderswap(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_orderswap(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_orderswap(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_orderswap(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_orderswap(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_orderswap(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_orderswap(
    vec_uchar16_arg a
);

```

## Arguments

[in] *a* A vector to reverse the lanes for.

## Return Values

A vector with a reversed (shuffled) order to the supplied vector.

## Description

Reverses the order of a vector's lanes.

# Lane Order Reversal (Big to Native)

## Functions

Reverses a vector from big order to native order.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_frombo(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_frombo(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_frombo(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_frombo(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_frombo(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_frombo(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_frombo(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_frombo(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_frombo(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_frombo(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_frombo(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_frombo(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_frombo(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_frombo(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_frombo(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_frombo(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_frombo(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_frombo(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_frombo(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_frombo(
    vec_uchar16_arg a
);
```

## Arguments

[in] *a* A vector with big lane order.

## Return Values

A vector with native lane order.

## Description

Reverses a vector from big order to native order.

# Lane Order Reversal (Little to Native)

## Functions

Reverses a vector from little order to native order.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_fromlo(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_fromlo(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_fromlo(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_fromlo(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_fromlo(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_fromlo(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_fromlo(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_fromlo(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_fromlo(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_fromlo(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_fromlo(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_fromlo(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_fromlo(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_fromlo(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_fromlo(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_fromlo(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_fromlo(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_fromlo(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_fromlo(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_fromlo(
    vec_uchar16_arg a
);

```

## Arguments

[in] *a* A vector with little lane order.

## Return Values

A vector with native lane order.

## Description

Reverses a vector from little order to native order.

# Lane Order Reversal (Native to Big)

## Functions

Reverses a vector from native order to big order.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_tobo(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_tobo(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_tobo(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_tobo(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tobo(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tobo(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_tobo(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_tobo(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_tobo(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_tobo(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tobo(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tobo(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_tobo(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_tobo(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tobo(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tobo(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_tobo(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_tobo(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_tobo(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_tobo(
    vec_uchar16_arg a
);
```

## Arguments

[in] *a* A vector with native lane order.

## Return Values

A vector with big lane order.

## Description

Reverses a vector from native order to big order.

# Lane Order Reversal (Native to Little)

## Functions

Reverses a vector from native order to little order.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_tolo(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_tolo(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_tolo(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_tolo(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tolo(
    vec_ullong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tolo(
    vec_ullong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_tolo(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_tolo(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_tolo(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_tolo(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tolo(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tolo(
    vec_uint4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_tolo(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_tolo(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tolo(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tolo(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_tolo(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_tolo(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_tolo(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_tolo(
    vec_uchar16_arg a
);
```

## Arguments

[in] *a* A vector with native lane order.

## Return Values

A vector with little lane order.

## Description

Reverses a vector from native order to little order.

# Lane Shifting (Create from Appended)

## Functions

Creates a vector by appending the supplied vectors starting at lane *i*.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_shift(
    vec_double1_arg a,
    vec_double1_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_shift(
    vec_double2_arg a,
    vec_double2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_shift(
    vec_llong1_arg a,
    vec_llong1_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_shift(
    vec_llong2_arg a,
    vec_llong2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_shift(
    vec_ullong1_arg a,
    vec_ullong1_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_shift(
    vec_ullong2_arg a,
    vec_ullong2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_shift(
    vec_float2_arg a,
    vec_float2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_shift(
    vec_float4_arg a,
    vec_float4_arg b,
    uint32_t i
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_shift(
    vec_int2_arg a,
    vec_int2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_shift(
    vec_int4_arg a,
    vec_int4_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_shift(
    vec_uint2_arg a,
    vec_uint2_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_shift(
    vec_uint4_arg a,
    vec_uint4_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_shift(
    vec_short4_arg a,
    vec_short4_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_shift(
    vec_short8_arg a,
    vec_short8_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_shift(
    vec_ushort4_arg a,
    vec_ushort4_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_shift(
    vec_ushort8_arg a,
    vec_ushort8_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_shift(
    vec_char8_arg a,
    vec_char8_arg b,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_shift(
    vec_char16_arg a,
    vec_char16_arg b,
    uint32_t i
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_shift(  
    vec_uchar8_arg a,  
    vec_uchar8_arg b,  
    uint32_t i  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_shift(  
    vec_uchar16_arg a,  
    vec_uchar16_arg b,  
    uint32_t i  
) ;
```

## **Arguments**

[in] <i>a</i>	The first vector
[in] <i>b</i>	The second vector.
[in] <i>i</i>	The lane to start shifting from.

## **Return Values**

A lane shifted vector.

## **Description**

Creates a vector by appending the supplied vectors starting at lane *i*.

# Lane Shifting (Create from Rotated)

## Functions

Creates a vector from a rotated vector starting at lane *i*.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_rotate(
    vec_double1_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_rotate(
    vec_double2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_rotate(
    vec_llong1_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_rotate(
    vec_llong2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_rotate(
    vec_ullong1_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_rotate(
    vec_ullong2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_rotate(
    vec_float2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_rotate(
    vec_float4_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_rotate(
    vec_int2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_rotate(
    vec_int4_arg a,
    uint32_t i
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_rotate(
    vec_uint2_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_rotate(
    vec_uint4_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_rotate(
    vec_short4_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_rotate(
    vec_short8_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_rotate(
    vec_ushort4_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_rotate(
    vec_ushort8_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_rotate(
    vec_char8_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_rotate(
    vec_char16_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_rotate(
    vec_uchar8_arg a,
    uint32_t i
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_rotate(
    vec_uchar16_arg a,
    uint32_t i
);

```

## Arguments

- |               |                         |
|---------------|-------------------------|
| [in] <i>a</i> | The input vector.       |
| [in] <i>i</i> | The lane to start from. |

## Return Values

A rotated vector.

SCE CONFIDENTIAL

**Description**

---

Creates a vector from a rotated vector starting at lane  $i$ .

000004892117

# Loading (Scalar Aligned)

## Functions

Loads a whole vector, which is assumed to be aligned to the scalar type.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_load1(
    const double *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_load1(
    const int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_load1(
    const uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_load2(
    const double *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_load2(
    const int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_load2(
    const uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_load2(
    const float *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_load2(
    const int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_load2(
    const uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_load4(
    const float *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_load4(
    const int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_load4(
    const uint32_t *p
);
```

SCE CONFIDENTIAL

---

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_load4(  

    const int16_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_load4(  

    const uint16_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_load8(  

    const int16_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_load8(  

    const uint16_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_load8(  

    const int8_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_load8(  

    const uint8_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_load16(  

    const int8_t *p  

) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_load16(  

    const uint8_t *p  

) ;

```

## Arguments

[in] *p* A pointer to a vector, which is only aligned to the scalar type.

## Return Values

The loaded vector.

## Description

Loads a whole vector, which is assumed to be aligned to the scalar type.

# Loading (Aligned with Lane Counts)

## Functions

Loads an aligned whole vector with specific lane counts.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_load1(
    const vec_double1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_load1(
    const vec_llong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_load1(
    const vec_ullong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_load2(
    const vec_double2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_load2(
    const vec_llong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_load2(
    const vec_ullong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_load2(
    const vec_float2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_load2(
    const vec_int2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_load2(
    const vec_uint2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_load4(
    const vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_load4(
    const vec_int4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_load4(
    const vec_uint4 *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_load4(
    const vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_load4(
    const vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_load8(
    const vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_load8(
    const vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_load8(
    const vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_load8(
    const vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_load16(
    const vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_load16(
    const vec_uchar16 *p
);

```

## Arguments

[in] *p*

A pointer to a vector, which is assumed to be aligned.

## Return Values

The loaded vector.

## Description

Loads an aligned whole vector with specific lane counts.

# Loading (Aligned)

## Functions

Loads an aligned whole vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_load(
    const vec_double1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_load(
    const vec_double2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_load(
    const vec_llong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_load(
    const vec_llong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_load(
    const vec_ullong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_load(
    const vec_ullong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_load(
    const vec_float2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_load(
    const vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_load(
    const vec_int2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_load(
    const vec_int4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_load(
    const vec_uint2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_load(
    const vec_uint4 *p
);
```

SCE CONFIDENTIAL

---

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_load(
    const vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_load(
    const vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_load(
    const vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_load(
    const vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_load(
    const vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_load(
    const vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_load(
    const vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_load(
    const vec_uchar16 *p
);

```

**Arguments**

[in] *p* A pointer to a vector, which is assumed to be aligned.

**Return Values**

The loaded vector.

**Description**

Loads an aligned whole vector.

# Loading (Four Sequential Aligned Vectors)

## Functions

Loads four sequential aligned vectors and transposes the results.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_load4x4(
    const vec_float4 *p,
    vec_float4 &a,
    vec_float4 &b,
    vec_float4 &c,
    vec_float4 &d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_load4x4(
    const vec_uint4 *p,
    vec_uint4 &a,
    vec_uint4 &b,
    vec_uint4 &c,
    vec_uint4 &d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_load4x4(
    const vec_int4 *p,
    vec_int4 &a,
    vec_int4 &b,
    vec_int4 &c,
    vec_int4 &d
);
```

### Arguments

[in] <i>p</i>	A pointer to four source vectors, which are assumed to be aligned.
[out] <i>a</i>	Receives the transposed first lane of each source vector (xxxx).
[out] <i>b</i>	Receives the transposed second lane of each source vector (yyyy).
[out] <i>c</i>	Receives the transposed third lane of each source vector (zzzz).
[out] <i>d</i>	Receives the transposed fourth lane of each source vector (wwww).

### Return Values

None

### Description

Loads four sequential aligned vectors and transposes the results.

# Loading (No Alignment)

## Functions

Loads an unaligned whole vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_load1u(
    const double *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_load1u(
    const int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_load1u(
    const uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_load2u(
    const double *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_load2u(
    const int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_load2u(
    const uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_load2u(
    const float *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_load2u(
    const int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_load2u(
    const uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_load4u(
    const float *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_load4u(
    const int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_load4u(
    const uint32_t *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_load4u(
    const int16_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_load4u(
    const uint16_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_load8u(
    const int16_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_load8u(
    const uint16_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_load8u(
    const int8_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_load8u(
    const uint8_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_load16u(
    const int8_t *p
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_load16u(
    const uint8_t *p
) ;

```

## Arguments

[in] *p* A pointer to a vector, which is assumed to have no alignment.

## Return Values

The loaded vector.

## Description

Loads an unaligned whole vector.

# Multiplication

## Functions

Multiplies two vector together: (a \* b).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_mul(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_mul(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_mul(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_mul(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_mul(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_mul(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_mul(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_mul(
    vec_uint4_arg a,
    vec_uint4_arg b
);
```

### Arguments

- [in] *a* The first vector.
- [in] *b* The second vector.

### Return Values

The product of the supplied vectors.

SCE CONFIDENTIAL

**Description**

---

Multiplies two vector together: (a \* b).

000004892117

# Multiplication (Followed by Addition)

## Functions

Multiplies two vectors together and then adds a third: ((a\*b)+c).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_madd(
    vec_double1_arg a,
    vec_double1_arg b,
    vec_double1_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_madd(
    vec_double2_arg a,
    vec_double2_arg b,
    vec_double2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_madd(
    vec_float2_arg a,
    vec_float2_arg b,
    vec_float2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_madd(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_madd(
    vec_int2_arg a,
    vec_int2_arg b,
    vec_int2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_madd(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_madd(
    vec_uint2_arg a,
    vec_uint2_arg b,
    vec_uint2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_madd(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);
```

SCE CONFIDENTIAL

---

## Arguments

---

- [in]  $a$  The first vector.
- [in]  $b$  The second vector.
- [in]  $c$  The third vector. This is added to the product of the first two inputs.

---

## Return Values

---

The result of the supplied vectors:  $((a \cdot b) + c)$ .

---

## Description

---

Multiplies two vectors together and then adds a third:  $((a \cdot b) + c)$ .

---

## Notes

---

Depending on the compiler and target CPU, this function may or may not be fused into a single instruction. This will affect the precision of the result.

# Multiplication (Followed by Subtraction)

## Functions

Multiplies two vectors together and then subtracts a third: ((a\*b)-c).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_msub(
    vec_double1_arg a,
    vec_double1_arg b,
    vec_double1_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_msub(
    vec_double2_arg a,
    vec_double2_arg b,
    vec_double2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_msub(
    vec_float2_arg a,
    vec_float2_arg b,
    vec_float2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_msub(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_msub(
    vec_int2_arg a,
    vec_int2_arg b,
    vec_int2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_msub(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_msub(
    vec_uint2_arg a,
    vec_uint2_arg b,
    vec_uint2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_msub(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);
```

SCE CONFIDENTIAL

---

## Arguments

---

- [in]  $a$  The first vector.
- [in]  $b$  The second vector.
- [in]  $c$  The third vector. This is subtracted from the product of the first two inputs.

---

## Return Values

---

The result of the supplied vectors:  $((a*b)-c)$ .

---

## Description

---

Multiplies two vectors together and then subtracts a third:  $((a*b)-c)$ .

---

## Notes

---

Depending on the compiler and target CPU, this function may or may not be fused into a single instruction. This will affect the precision of the result.

# Multiplication (Product Subtracted)

## Functions

Subtracts the product of a vector multiplication from a third vector: (c-(a\*b)).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_nmsub(
    vec_double1_arg a,
    vec_double1_arg b,
    vec_double1_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_nmsub(
    vec_double2_arg a,
    vec_double2_arg b,
    vec_double2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_nmsub(
    vec_float2_arg a,
    vec_float2_arg b,
    vec_float2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_nmsub(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_nmsub(
    vec_int2_arg a,
    vec_int2_arg b,
    vec_int2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_nmsub(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_nmsub(
    vec_uint2_arg a,
    vec_uint2_arg b,
    vec_uint2_arg c
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_nmsub(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c
);
```

SCE CONFIDENTIAL

---

**Arguments**

---

- [in]  $a$  The first vector.
- [in]  $b$  The second vector.
- [in]  $c$  The third vector. The product of the first two inputs are subtracted from this.

**Return Values**

---

The result of the supplied vectors:  $(c - (a * b))$ .

**Description**

---

Subtracts the product of a vector multiplication from a third vector:  $(c - (a * b))$ .

**Notes**

---

Depending on the compiler and target CPU, this function may or may not be fused into a single instruction. This will affect the precision of the result.

# Negation

## Functions

Negates a vector. This may just involve flipping the sign bit in the case of floating point input.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_negate(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_negate(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_negate(
    vec_llong1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_negate(
    vec_llong2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_negate(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_negate(
    vec_float4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_negate(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_negate(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_negate(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_negate(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_negate(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_negate(
    vec_char16_arg a
);
```

SCE CONFIDENTIAL

## **Arguments**

[in]  $a$  The vector to negate.

## **Return Values**

The negative of the supplied vector.

## **Description**

Negates a vector. This may just involve flipping the sign bit in the case of floating point input.

## **Notes**

For floating point values, this may be just a bit manipulation with no INF/NAN handling.

# Per-Lane Bit Test

## Functions

Performs a per-lane bit test. This involves a bitwise AND between matching lanes of the two supplied vectors. If any bits are set after the operation, all the bits in the lane are then set.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tst(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tst(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_tst(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_tst(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tst(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tst(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_tst(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_tst(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tst(
    vec_short4_arg a,
    vec_short4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tst(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_tst(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_tst(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_tst(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_tst(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_tst(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_tst(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- [in] *a*              The first vector to compare.  
 [in] *b*              The second vector to compare.

## Return Values

The result of the comparison. For each lane, zero/all bits are used for false/true.

## Description

Performs a per-lane bit test. This involves a bitwise AND between matching lanes of the two supplied vectors. If any bits are set after the operation, all the bits in the lane are then set.

# Per-Lane Maximum Value

## Functions

Chooses the maximum value for each lane from the corresponding lanes of the supplied vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_max(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_max(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_max(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_max(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_max(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_max(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_max(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_max(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_max(
    vec_int2_arg a,
    vec_int2_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_max(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_max(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_max(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_max(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_max(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_max(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_max(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_max(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_max(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_max(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_max(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- |               |                    |
|---------------|--------------------|
| [in] <i>a</i> | The first vector.  |
| [in] <i>b</i> | The second vector. |

SCE CONFIDENTIAL

---

### **Return Values**

A vector in which each lane is the maximum of the corresponding lanes of the supplied vectors.

---

### **Description**

Chooses the maximum value for each lane from the corresponding lanes of the supplied vectors.

# Per-Lane Minimum Value

## Functions

Chooses the minimum value for each lane from the corresponding lanes of the supplied vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_min(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_min(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_min(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_min(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_min(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_min(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_min(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_min(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_min(
    vec_int2_arg a,
    vec_int2_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_min(
    vec_int4_arg a,
    vec_int4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_min(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_min(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_min(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_min(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_min(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_min(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_min(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_min(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_min(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_min(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## Arguments

- |               |                    |
|---------------|--------------------|
| [in] <i>a</i> | The first vector.  |
| [in] <i>b</i> | The second vector. |

SCE CONFIDENTIAL

---

**Return Values**

A vector in which each lane is the minimum of the corresponding lanes of the specified vectors.

---

**Description**

Chooses the minimum value for each lane from the corresponding lanes of the supplied vectors.

# Point Mantissa Extraction

## Functions

Extracts a floating point mantissa vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_mantissa(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_mantissa(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_mantissa(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_mantissa(
    vec_float4_arg a
);
```

### Arguments

[in] *a* A floating point vector.

### Return Values

The floating point mantissa vector.

### Description

Extracts a floating point mantissa vector.

# Reciprocal

## Functions

Calculates the reciprocal for a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_recip(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_recip(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_recip(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_recip(
    vec_float4_arg a
);
```

### Arguments

[in] *a* The vector to calculate the reciprocal for.

### Return Values

The reciprocal of the supplied vector.

### Description

Calculates the reciprocal for a vector.

### Notes

The result may be a Newton-Raphson approximation.

# Reciprocal Square Root

## Functions

Calculates the reciprocal square root for a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_rsqrt(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_rsqrt(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_rsqrt(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_rsqrt(
    vec_float4_arg a
);
```

### Arguments

[in] *a* The vector to calculate the reciprocal square root for.

### Return Values

The reciprocal square root of the supplied vector.

### Description

Calculates the reciprocal square root for a vector.

### Notes

The result may be a Newton-Raphson approximation.

# Selection (Per-Bit)

## Functions

Performs per-bit vector selection:  $( (a \& \sim\text{mask}) | (b \& \text{mask}) )$ . For each bit,  $b$  is selected if masked; otherwise  $a$  is selected.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_sel(
    vec_double1_arg a,
    vec_double1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_sel(
    vec_double2_arg a,
    vec_double2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sel(
    vec_ullong1_arg a,
    vec_ullong1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sel(
    vec_ullong2_arg a,
    vec_ullong2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sel(
    vec_llong1_arg a,
    vec_llong1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sel(
    vec_llong2_arg a,
    vec_llong2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_sel(
    vec_float2_arg a,
    vec_float2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sel(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_uint4_arg mask
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sel(
    vec_uint2_arg a,
    vec_uint2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sel(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sel(
    vec_int2_arg a,
    vec_int2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sel(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_uint4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sel(
    vec_ushort4_arg a,
    vec_ushort4_arg b,
    vec_ushort4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sel(
    vec_ushort8_arg a,
    vec_ushort8_arg b,
    vec_ushort8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sel(
    vec_short4_arg a,
    vec_short4_arg b,
    vec_ushort4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sel(
    vec_short8_arg a,
    vec_short8_arg b,
    vec_ushort8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sel(
    vec_uchar8_arg a,
    vec_uchar8_arg b,
    vec_uchar8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sel(
    vec_uchar16_arg a,
    vec_uchar16_arg b,
    vec_uchar16_arg mask
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sel(
    vec_char8_arg a,
    vec_char8_arg b,
    vec_uchar8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sel(
    vec_char16_arg a,
    vec_char16_arg b,
    vec_uchar16_arg mask
);
```

## **Arguments**

- |                  |                            |
|------------------|----------------------------|
| [in] <i>a</i>    | The first vector.          |
| [in] <i>b</i>    | The second vector.         |
| [in] <i>mask</i> | The selection mask vector. |

## **Return Values**

The selected result of the supplied vectors.

## **Description**

Performs per-bit vector selection:  $( (a \& \sim\text{mask}) | (b \& \text{mask}) )$ . For each bit, *b* is selected if masked; otherwise *a* is selected.

# Selection (Per-Lane)

## Functions

Performs per-lane vector selection. This is for fast lane selection, and must only be used with a mask that either has zero or all bits set in each lane (e.g. from comparison results).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_csel(
    vec_double1_arg a,
    vec_double1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_csel(
    vec_double2_arg a,
    vec_double2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_csel(
    vec_ullong1_arg a,
    vec_ullong1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_csel(
    vec_ullong2_arg a,
    vec_ullong2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_csel(
    vec_llong1_arg a,
    vec_llong1_arg b,
    vec_ullong1_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_csel(
    vec_llong2_arg a,
    vec_llong2_arg b,
    vec_ullong2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_csel(
    vec_float2_arg a,
    vec_float2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_csel(
    vec_float4_arg a,
    vec_float4_arg b,
    vec_uint4_arg mask
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_csel(
    vec_uint2_arg a,
    vec_uint2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_csel(
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_csel(
    vec_int2_arg a,
    vec_int2_arg b,
    vec_uint2_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_csel(
    vec_int4_arg a,
    vec_int4_arg b,
    vec_uint4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_csel(
    vec_ushort4_arg a,
    vec_ushort4_arg b,
    vec_ushort4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_csel(
    vec_ushort8_arg a,
    vec_ushort8_arg b,
    vec_ushort8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_csel(
    vec_short4_arg a,
    vec_short4_arg b,
    vec_ushort4_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_csel(
    vec_short8_arg a,
    vec_short8_arg b,
    vec_ushort8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_csel(
    vec_uchar8_arg a,
    vec_uchar8_arg b,
    vec_uchar8_arg mask
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_csel(
    vec_uchar16_arg a,
    vec_uchar16_arg b,
    vec_uchar16_arg mask
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_csel(  
    vec_char8_arg a,  
    vec_char8_arg b,  
    vec_uchar8_arg mask  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_csel(  
    vec_char16_arg a,  
    vec_char16_arg b,  
    vec_uchar16_arg mask  
) ;
```

## Arguments

- |                  |   |
|------------------|---|
| [in] <i>a</i>    | The first vector. A lane is chosen from this vector if the corresponding selection mask lane has zero bits set. |
| [in] <i>b</i>    | The second vector. A lane is chosen from this vector if the corresponding selection mask lane has all bits set. |
| [in] <i>mask</i> | The selection mask vector.  |

## Return Values

The selected result from the two specified vectors.

## Description

Performs per-lane vector selection. This is for fast lane selection, and must only be used with a mask that either has zero or all bits set in each lane (e.g. from comparison results).

# Short Packing

## Functions

Packs shorts from one vector into another: (0,1,2,3,...) -> (0,2,.....).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_packlu16(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_packlu16(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* A vector containing the shorts to pack.

### Return Values

A vector containing the packed shorts.

### Description

Packs shorts from one vector into another: (0,1,2,3,...) -> (0,2,.....).

# Short Unpacking

## Functions

Unpacks the lowest unsigned shorts from a vector: (0,1,...) -> ((uint)0,(uint)1,..) and zero extends each to an unsigned integer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_unpack1u16(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_unpack1u16(
    vec_uint4_arg a
);
```

### Arguments

[in] *a* A vector containing the unsigned shorts to unpack.

### Return Values

A vector containing the unpacked zero extended unsigned integers.

### Description

Unpacks the lowest unsigned shorts from a vector: (0,1,...) -> ((uint)0,(uint)1,..) and zero extends each to an unsigned integer.

# Shuffle (128-bit)

## Functions

Shuffles two 128-bit vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xayb(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xayb(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xayb(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_z cwd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_z cwd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_z cwd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyza(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyza(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyza(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yzab(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yzab(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yzab(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zabc(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zabc(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zabc(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wabc(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wabc(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wabc(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zwab(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zwab(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zwab(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xycox(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xycx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xycx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ycxy(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_ycxy(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ycxy(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_cxyc(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_cxyc(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_cxyc(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zayx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zayx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zayx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_bzxx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_bzxx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_bzxx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xzya(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xzya(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xzya(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zxxb(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zxxb(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zxxb(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yxac(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yxac(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yxac(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_bbyx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_bbyx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_bbyx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xzbx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xzbx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xzbx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_caxx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_caxx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_caxx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yaxx(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yaxx(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yaxx(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xxaa(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xxaa(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xxaa(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yybb(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yybb(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yybb(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wwdd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wwdd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wwdd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyab(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyab(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyab(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yxba(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yxba(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yxba(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zwcd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zwcd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zwcd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wzdc(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wzdc(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wzdc(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zwba(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zwba(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zwba(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zbwa(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zbwa(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zbwa(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xzac(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xzac(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xzac(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ywbd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_ywbd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ywbd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xayd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xayd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xayd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zbwd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zbwd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zbwd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xcyd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xcyd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xcyd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_ayzw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_ayzw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_ayzw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xazw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xazw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xazw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyaw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyaw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyaw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xbzw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xbzw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xbzw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xycw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xycw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xycw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyzd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyzd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyzd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xbcd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xbcd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xbcd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_aycd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_aycd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_aycd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_abzd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_abzd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_abzd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_abcw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_abcw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_abcw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xycd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xycd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xycd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xbcw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xbcw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xbcw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_abzw(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_abzw(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_abzw(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zbxz(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zbxz(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zbxz(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wcya(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wcya(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wcya(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xdzb(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xdzb(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xdzb(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zcxa(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zcxa(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zcxa(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xbzd(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xbzd(
    vec_int4_arg xyzw,
    vec_int4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xbzd(
    vec_uint4_arg xyzw,
    vec_uint4_arg abcd
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wdyb(
    vec_float4_arg xyzw,
    vec_float4_arg abcd
);

```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wdyb(  
    vec_int4_arg xyzw,  
    vec_int4_arg abcd  
) ;
```

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wdyb(  
    vec_uint4_arg xyzw,  
    vec_uint4_arg abcd  
) ;
```

## **Arguments**

[in] xyzw	The first vector.
[in] abcd	The second vector.

## **Return Values**

The shuffled vector.

## **Description**

Shuffles two 128-bit vectors.

# Shuffle (64-bit)

## Functions

Shuffles two 64-bit vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_xa(
    vec_float2_arg xy,
    vec_float2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_xa(
    vec_int2_arg xy,
    vec_int2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_xa(
    vec_uint2_arg xy,
    vec_uint2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_xb(
    vec_float2_arg xy,
    vec_float2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_xb(
    vec_int2_arg xy,
    vec_int2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_xb(
    vec_uint2_arg xy,
    vec_uint2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_ya(
    vec_float2_arg xy,
    vec_float2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_ya(
    vec_int2_arg xy,
    vec_int2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_ya(
    vec_uint2_arg xy,
    vec_uint2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_yb(
    vec_float2_arg xy,
    vec_float2_arg ab
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_yb(
    vec_int2_arg xy,
    vec_int2_arg ab
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_yb(
    vec_uint2_arg xy,
    vec_uint2_arg ab
);
```

## **Arguments**

- |                |                    |
|----------------|--------------------|
| [in] <i>xy</i> | The first vector.  |
| [in] <i>ab</i> | The second vector. |

## **Return Values**

The shuffled vector.

## **Description**

Shuffles two 64-bit vectors.

# Shuffle (Template Arguments)

## Functions

Shuffles two vectors based on template arguments. A negative value should be used to indicate undefined/don't care.

### Definition

```
#include <vectormath.h>
template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_double1
sce_vectormath_shuffle2t(
    vec_double1_arg v0,
    vec_double1_arg v1
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_double2
sce_vectormath_shuffle2t(
    vec_double2_arg v0,
    vec_double2_arg v1
);

template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1
sce_vectormath_shuffle2t(
    vec_ullong1_arg v0,
    vec_ullong1_arg v1
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2
sce_vectormath_shuffle2t(
    vec_ullong2_arg v0,
    vec_ullong2_arg v1
);

template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_llong1
sce_vectormath_shuffle2t(
    vec_llong1_arg v0,
    vec_llong1_arg v1
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_llong2
sce_vectormath_shuffle2t(
    vec_llong2_arg v0,
    vec_llong2_arg v1
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_float2
sce_vectormath_shuffle2t(
    vec_float2_arg v0,
    vec_float2_arg v1
);

template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_float4
sce_vectormath_shuffle2t(
    vec_float4_arg v0,
    vec_float4_arg v1
);
```

SCE CONFIDENTIAL

```

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_uint2
sce_vectormath_shuffle2t(
    vec_uint2_arg v0,
    vec_uint2_arg v1
);

template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_uint4
sce_vectormath_shuffle2t(
    vec_uint4_arg v0,
    vec_uint4_arg v1
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_int2
sce_vectormath_shuffle2t(
    vec_int2_arg v0,
    vec_int2_arg v1
);

template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_int4
sce_vectormath_shuffle2t(
    vec_int4_arg v0,
    vec_int4_arg v1
);

```

## Arguments

- |         |                    |
|---------|--------------------|
| [in] v0 | The first vector.  |
| [in] v1 | The second vector. |

## Return Values

The shuffled vector.

## Description

Shuffles two vectors based on template arguments. A negative value should be used to indicate undefined/don't care.

# Shuffling

## Functions

Byte shuffles up to four vectors using runtime lane byte indices.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_shuffle(
    vec_uint4_arg v0,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_shuffle(
    vec_uint4_arg v0,
    vec_uint4_arg v1,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_shuffle(
    vec_uint4_arg v0,
    vec_uint4_arg v1,
    vec_uint4_arg v2,
    vec_uchar16_arg s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_shuffle(
    vec_uint4_arg v0,
    vec_uint4_arg v1,
    vec_uint4_arg v2,
    vec_uint4_arg v3,
    vec_uchar16_arg s
);
```

### Arguments

[in] <i>v0</i>	The first vector to shuffle (byte indices 0-15).
[in] <i>s</i>	A shuffle vector containing the byte index from the supplied vectors for each byte of the result.
<i>v1</i>	The second vector to shuffle (byte indices 16-31).
<i>v2</i>	The third vector to shuffle (byte indices 32-47).
<i>v3</i>	The fourth vector to shuffle (byte indices 48-63).

### Return Values

The shuffled vector.

### Description

Byte shuffles up to four vectors using runtime lane byte indices.

### Notes

CAUTION - VERY SLOW ON SOME PLATFORMS!

# Signed Integer Sign Bit Extension

## Functions

Extends a whole, lower half or upper half signed integer vector by prepending a sign bit.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sign_extend(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sign_extend(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sign_extend(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sign_extend_low(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sign_extend_low(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sign_extend_low(
    vec_short4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sign_extend_low(
    vec_short8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sign_extend_low(
    vec_char8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sign_extend_low(
    vec_char16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sign_extend_high(
    vec_int2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sign_extend_high(
    vec_int4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sign_extend_high(
    vec_short4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sign_extend_high(
    vec_short8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sign_extend_high(
    vec_char8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sign_extend_high(
    vec_char16_arg a
) ;
```

## Arguments

[in] *a* A signed integer vector.

## Return Values

The sign extended signed integer vector.

## Description

Extends a whole, lower half or upper half signed integer vector by prepending a sign bit.

# Splatting (Using a Scalar Value)

## Functions

Splats a vector using a supplied scalar value.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_splats1(
    double s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_splats1(
    int64_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_splats1(
    uint64_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_splats2(
    double s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_splats2(
    int64_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_splats2(
    uint64_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_splats2(
    float s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_splats2(
    int32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_splats2(
    uint32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_splats4(
    float s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_splats4(
    int32_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_splats4(
    uint32_t s
);
```

SCE CONFIDENTIAL

---

```

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_splats4(
    int16_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_splats4(
    uint16_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_splats8(
    int16_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_splats8(
    uint16_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_splats8(
    int8_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_splats8(
    uint8_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_splats16(
    int8_t s
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_splats16(
    uint8_t s
);

```

## Arguments

[in] *s* The scalar value to be splatted to every lane of the returned vector.

## Return Values

The splatted vector.

## Description

Splats a vector using a supplied scalar value.

# Splatting (Using an Index into Another Vector)

## Functions

Splats a vector using an indexed lane of a supplied vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_splat(
    vec_double1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_splat(
    vec_double2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_splat(
    vec_llong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_splat(
    vec_llong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_splat(
    vec_ullong1_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_splat(
    vec_ullong2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_splat(
    vec_float2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_splat(
    vec_float4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_splat(
    vec_int2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_splat(
    vec_int4_arg v,
    uint32_t slot
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_splat(
    vec_uint2_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_splat(
    vec_uint4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_splat(
    vec_short4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_splat(
    vec_short8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_splat(
    vec_ushort4_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_splat(
    vec_ushort8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_splat(
    vec_char8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_splat(
    vec_char16_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_splat(
    vec_uchar8_arg v,
    uint32_t slot
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_splat(
    vec_uchar16_arg v,
    uint32_t slot
);

```

## Arguments

- [in] *v* The vector containing the lane to splat with.  
 [in] *slot* The index of the supplied vector's lane to be splatted to every lane of the returned vector.

## Return Values

The splatted vector.

SCE CONFIDENTIAL

---

**Description**

---

Splats a vector using an indexed lane of a supplied vector.

---

**Notes**

---

CAUTION - VERY SLOW ON SOME PLATFORMS!

000004892117

# Square Root

## Functions

Calculates the square root of a vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_sqrt(
    vec_double1_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_sqrt(
    vec_double2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_sqrt(
    vec_float2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sqrt(
    vec_float4_arg a
);
```

### Arguments

[in] *a* The vector to calculate the square root for.

### Return Values

The square root of the supplied vector.

### Description

Calculates the square root of a vector.

### Notes

The result may be a Newton-Raphson approximation.

# Storing (4 Vector Transposition)

## Functions

Transposes 4 aligned vectors and stores the results as sequential vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4x4(
    vec_float4 *p,
    vec_float4_arg a,
    vec_float4_arg b,
    vec_float4_arg c,
    vec_float4_arg d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4x4(
    vec_uint4 *p,
    vec_uint4_arg a,
    vec_uint4_arg b,
    vec_uint4_arg c,
    vec_uint4_arg d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4x4(
    vec_int4 *p,
    vec_int4_arg a,
    vec_int4_arg b,
    vec_int4_arg c,
    vec_int4_arg d
);
```

### Arguments

[in] <i>p</i>	Receives the four vectors as x0x1x2x3, y0y1y2y3, z0z1z2z3 and w0w1w2w3.
[in] <i>a</i>	The first source vector.
[in] <i>b</i>	The second source vector.
[in] <i>c</i>	The third source vector.
[in] <i>d</i>	The fourth source vector.

### Return Values

None

### Description

Transposes 4 aligned vectors and stores the results as sequential vectors.

# Storing (Aligned - Eight Lanes Only)

## Functions

Stores the first eight lanes of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_uchar8_arg v,
    vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_uchar16_arg v,
    vec_uchar16 *p
);
```

### Arguments

[in] *v*  
 [out] *p*

The vector to store.

Receives the first eight lanes of the vector and is assumed to be vector aligned.

### Return Values

None

### Description

Stores the first eight lanes of a vector to an aligned pointer.

# Storing (Aligned - First Lane Only)

## Functions

Stores the first lane of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_double1_arg v,
    vec_double1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_double2_arg v,
    vec_double2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_llong1_arg v,
    vec_llong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_llong2_arg v,
    vec_llong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ullong1_arg v,
    vec_ullong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ullong2_arg v,
    vec_ullong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_float2_arg v,
    vec_float2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_float4_arg v,
    vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_int2_arg v,
    vec_int2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_int4_arg v,
    vec_int4 *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uint2_arg v,
    vec_uint2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uint4_arg v,
    vec_uint4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_short4_arg v,
    vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ushort4_arg v,
    vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uchar8_arg v,
    vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uchar16_arg v,
    vec_uchar16 *p
);

```

## **Arguments**

[in] v	The vector to store.
[out] p	Receives the first lane of the vector and is assumed to be vector aligned.

## **Return Values**

None

SCE CONFIDENTIAL

**Description**

---

Stores the first lane of a vector to an aligned pointer.

000004892117

# Storing (Aligned - Four Lanes Only)

## Functions

Stores the first four lanes of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_float4_arg v,
    vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_int4_arg v,
    vec_int4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_uint4_arg v,
    vec_uint4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_short4_arg v,
    vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_ushort4_arg v,
    vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_uchar8_arg v,
    vec_uchar8 *p
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(  
    vec_uchar16_arg v,  
    vec_uchar16 *p  
) ;
```

## **Arguments**

---

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first four lanes of the vector and is assumed to be vector aligned.

## **Return Values**

---

None

## **Description**

---

Stores the first four lanes of a vector to an aligned pointer.

# Storing (Aligned - Sixteen Lanes Only)

## Functions

Stores the first sixteen lanes of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16(
    vec_uchar16_arg v,
    vec_uchar16 *p
);
```

### Arguments

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first sixteen lanes of the vector and is assumed to be vector aligned.

### Return Values

None

### Description

Stores the first sixteen lanes of a vector to an aligned pointer.

# Storing (Aligned - Three Lanes Only)

## Functions

Stores the first three lanes of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_float4_arg v,
    vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_int4_arg v,
    vec_int4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_uint4_arg v,
    vec_uint4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_short4_arg v,
    vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_ushort4_arg v,
    vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_uchar8_arg v,
    vec_uchar8 *p
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(  
    vec_uchar16_arg v,  
    vec_uchar16 *p  
) ;
```

## **Arguments**

---

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first three lanes of the vector and is assumed to be vector aligned.

## **Return Values**

---

None

## **Description**

---

Stores the first three lanes of a vector to an aligned pointer.

# Storing (Scalar Type Aligned - First Lane Only)

## Functions

Stores the first lane of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_double1_arg v,
    double *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_double2_arg v,
    double *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_llong1_arg v,
    int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_llong2_arg v,
    int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ullong1_arg v,
    uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ullong2_arg v,
    uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_float2_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_float4_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_int2_arg v,
    int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_int4_arg v,
    int32_t *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uint2_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uint4_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_short4_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_short8_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ushort4_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_ushort8_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_char8_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_char16_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uchar8_arg v,
    uint8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store1(
    vec_uchar16_arg v,
    uint8_t *p
);

```

## Arguments

[in] v	The vector to store.
[out] p	Receives the first lane of the vector and is assumed to be aligned to the scalar type.

## Return Values

None

SCE CONFIDENTIAL

**Description**

---

Stores the first lane of a vector to a scalar aligned pointer.

000004892117

# Storing (Scalar Type Aligned - Two Lanes Only)

## Functions

Stores the first two lanes of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_double2_arg v,
    double *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_llong2_arg v,
    int64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ullong2_arg v,
    uint64_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_float2_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_float4_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_int2_arg v,
    int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_int4_arg v,
    int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uint2_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uint4_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_short4_arg v,
    int16_t *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_short8_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ushort4_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ushort8_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_char8_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_char16_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uchar8_arg v,
    uint8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uchar16_arg v,
    uint8_t *p
);

```

## Arguments

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first two lanes of the vector and is assumed to be aligned to the scalar type.

## Return Values

None

## Description

Stores the first two lanes of a vector to a scalar aligned pointer.

# Storing (Aligned)

## Functions

Stores a whole vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_double1_arg v,
    vec_double1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_double2_arg v,
    vec_double2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_llong1_arg v,
    vec_llong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_llong2_arg v,
    vec_llong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_ullong1_arg v,
    vec_ullong1 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_ullong2_arg v,
    vec_ullong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_float2_arg v,
    vec_float2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_float4_arg v,
    vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_int2_arg v,
    vec_int2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_int4_arg v,
    vec_int4 *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_uint2_arg v,
    vec_uint2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_uint4_arg v,
    vec_uint4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_short4_arg v,
    vec_short4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_ushort4_arg v,
    vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_uchar8_arg v,
    vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store(
    vec_uchar16_arg v,
    vec_uchar16 *p
);

```

## Arguments

- |         |  |
|---------|--|
| [in] v  | The vector to store.                                     |
| [out] p | Receives the vector and is assumed to be vector aligned. |

## Return Values

None

SCE CONFIDENTIAL

**Description**

---

Stores a whole vector to an aligned pointer.

000004892117

# Storing (Aligned- Two Lanes Only)

## Functions

Stores the first two lanes of a vector to an aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_double2_arg v,
    vec_double2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_llong2_arg v,
    vec_llong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ullong2_arg v,
    vec_ullong2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_float2_arg v,
    vec_float2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_float4_arg v,
    vec_float4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_int2_arg v,
    vec_int2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_int4_arg v,
    vec_int4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uint2_arg v,
    vec_uint2 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uint4_arg v,
    vec_uint4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_short4_arg v,
    vec_short4 *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_short8_arg v,
    vec_short8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ushort4_arg v,
    vec_ushort4 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_ushort8_arg v,
    vec_ushort8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_char8_arg v,
    vec_char8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_char16_arg v,
    vec_char16 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uchar8_arg v,
    vec_uchar8 *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2(
    vec_uchar16_arg v,
    vec_uchar16 *p
);

```

## Arguments

[in] *v* The vector to store.  
 [out] *p* Receives the first two lanes of the vector and is assumed to be vector aligned.

## Return Values

None

## Description

Stores the first two lanes of a vector to an aligned pointer.

# Storing (No Alignment - Eight Lanes Only)

## Functions

Stores the first eight lanes of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_short8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_ushort8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_char8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_uchar8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8u(
    vec_uchar16_arg v,
    void *p
);
```

### Arguments

[in] *v*  
 [out] *p*

The vector to store.

Receives the first eight lanes of the vector and is assumed to have no alignment.

### Return Values

None

### Description

Stores the first eight lanes of a vector to an unaligned pointer.

# Storing (No Alignment - First Lane Only)

## Functions

Stores the first lane of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_double1_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_double2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_llong1_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_llong2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_ullong1_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_ullong2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_float2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_float4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_int2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_int4_arg v,
    void *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_uint2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_uint4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_short4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_short8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_ushort4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_ushort8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_char8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_uchar8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_storelu(
    vec_uchar16_arg v,
    void *p
);

```

## **Arguments**

[in] v	The vector to store.
[out] p	Receives the first lane of the vector and is assumed to have no alignment.

## **Return Values**

None

SCE CONFIDENTIAL

**Description**

---

Stores the first lane of a vector to an unaligned pointer.

000004892117

# Storing (No Alignment - Four Lanes Only)

## Functions

Stores the first four lanes of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_float4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_int4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_uint4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_short4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_short8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_ushort4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_ushort8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_char8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(
    vec_uchar8_arg v,
    void *p
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4u(  
    vec_uchar16_arg v,  
    void *p  
) ;
```

## **Arguments**

---

[in] *v*  
[out] *p*

The vector to store.  
Receives the first four lanes of the vector and is assumed to have no alignment.

## **Return Values**

---

None

## **Description**

---

Stores the first four lanes of a vector to an unaligned pointer.

# Storing (No Alignment - Sixteen Lanes Only)

## Functions

Stores the first sixteen lanes of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16u(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16u(
    vec_uchar16_arg v,
    void *p
);
```

### Arguments

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first sixteen lanes of the vector and is assumed to have no alignment.

### Return Values

None

### Description

Stores the first sixteen lanes of a vector to an unaligned pointer.

# Storing (No Alignment - Three Lanes Only)

## Functions

Stores the first three lanes of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_float4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_int4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_uint4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_short4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_short8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_ushort4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_ushort8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_char8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(
    vec_uchar8_arg v,
    void *p
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3u(  
    vec_uchar16_arg v,  
    void *p  
) ;
```

## **Arguments**

---

[in] *v*  
[out] *p*

The vector to store.  
Receives the first three lanes of the vector and is assumed to have no alignment.

## **Return Values**

---

None

## **Description**

---

Stores the first three lanes of a vector to an unaligned pointer.

# Storing (No Alignment - Two Lanes Only)

## Functions

Stores the first two lanes of a vector to an unaligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_double2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_llong2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_ullong2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_float2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_float4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_int2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_int4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_uint2_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_uint4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_short4_arg v,
    void *p
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_short8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_ushort4_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_ushort8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_char8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_char16_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_uchar8_arg v,
    void *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store2u(
    vec_uchar16_arg v,
    void *p
);

```

## Arguments

[in] *v* The vector to store.  
 [out] *p* Receives the first two lanes of the vector and is assumed to have no alignment.

## Return Values

None

## Description

Stores the first two lanes of a vector to an unaligned pointer.

# Storing (Scalar Type Aligned - Eight Lanes Only)

## Functions

Stores the first eight lanes of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_short8_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_ushort8_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_char8_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_char16_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_uchar8_arg v,
    uint8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store8(
    vec_uchar16_arg v,
    uint8_t *p
);
```

### Arguments

[in] *v*  
 [out] *p*

The vector to store.

Receives the first eight lanes of the vector and is assumed to be aligned to the scalar type.

### Return Values

None

### Description

Stores the first eight lanes of a vector to a scalar aligned pointer.

# Storing (Scalar Type Aligned - Four Lanes Only)

## Functions

Stores the first four lanes of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_float4_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_int4_arg v,
    int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_uint4_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_short4_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_short8_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_ushort4_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_ushort8_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_char8_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_char16_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(
    vec_uchar8_arg v,
    uint8_t *p
);
```

SCE CONFIDENTIAL

---

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store4(  
    vec_uchar16_arg v,  
    uint8_t *p  
) ;
```

## **Arguments**

---

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first four lanes of the vector and is assumed to be aligned to the scalar type.

## **Return Values**

---

None

## **Description**

---

Stores the first four lanes of a vector to a scalar aligned pointer.

# Storing (Scalar Type Aligned - Sixteen Lanes Only)

## Functions

Stores the first sixteen lanes of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16(
    vec_char16_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store16(
    vec_uchar16_arg v,
    uint8_t *p
);
```

### Arguments

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first sixteen lanes of the vector and is assumed to be aligned to the scalar type.

### Return Values

None

### Description

Stores the first sixteen lanes of a vector to a scalar aligned pointer.

# Storing (Scalar Type Aligned - Three Lanes Only)

## Functions

Stores the first three lanes of a vector to a scalar aligned pointer.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_float4_arg v,
    float *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_int4_arg v,
    int32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_uint4_arg v,
    uint32_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_short4_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_short8_arg v,
    int16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_ushort4_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_ushort8_arg v,
    uint16_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_char8_arg v,
    int8_t *p
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(
    vec_char16_arg v,
    int8_t *p
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(  
    vec_uchar8_arg v,  
    uint8_t *p  
) ;  
  
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_store3(  
    vec_uchar16_arg v,  
    uint8_t *p  
) ;
```

## **Arguments**

[in] <i>v</i>	The vector to store.
[out] <i>p</i>	Receives the first three lanes of the vector and is assumed to be aligned to the scalar type.

## **Return Values**

None

## **Description**

Stores the first three lanes of a vector to a scalar aligned pointer.

# Subtraction

## Functions

Subtracts one vector from another ( $a - b$ ).

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_double1 sce_vectormath_sub(
    vec_double1_arg a,
    vec_double1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_double2 sce_vectormath_sub(
    vec_double2_arg a,
    vec_double2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong1 sce_vectormath_sub(
    vec_llong1_arg a,
    vec_llong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_llong2 sce_vectormath_sub(
    vec_llong2_arg a,
    vec_llong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_sub(
    vec_ullong1_arg a,
    vec_ullong1_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_sub(
    vec_ullong2_arg a,
    vec_ullong2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_sub(
    vec_float2_arg a,
    vec_float2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_sub(
    vec_float4_arg a,
    vec_float4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_sub(
    vec_int2_arg a,
    vec_int2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_sub(
    vec_int4_arg a,
    vec_int4_arg b
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_sub(
    vec_uint2_arg a,
    vec_uint2_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_sub(
    vec_uint4_arg a,
    vec_uint4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short4 sce_vectormath_sub(
    vec_short4_arg a,
    vec_short4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_short8 sce_vectormath_sub(
    vec_short8_arg a,
    vec_short8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_sub(
    vec_ushort4_arg a,
    vec_ushort4_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_sub(
    vec_ushort8_arg a,
    vec_ushort8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char8 sce_vectormath_sub(
    vec_char8_arg a,
    vec_char8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_char16 sce_vectormath_sub(
    vec_char16_arg a,
    vec_char16_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar8 sce_vectormath_sub(
    vec_uchar8_arg a,
    vec_uchar8_arg b
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uchar16 sce_vectormath_sub(
    vec_uchar16_arg a,
    vec_uchar16_arg b
);

```

## **Arguments**

- |               |                    |
|---------------|--------------------|
| [in] <i>a</i> | The first vector.  |
| [in] <i>b</i> | The second vector. |

## **Return Values**

The difference of the supplied vectors.

SCE CONFIDENTIAL

**Description**

---

Subtracts one vector from another ( $a - b$ ).

000004892117

SCE CONFIDENTIAL

# Swizzle (128-bit)

## Functions

Swizzles a 128-bit vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xxxx(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xxxx(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xxxx(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yyyy(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yyyy(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yyyy(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zzzz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zzzz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zzzz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wwww(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wwww(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wwww(
    vec_uint4_arg xyzw
);
```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yzx(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yzx(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yzx(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zwx(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zwx(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zwx(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wxy(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wxy(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wxy(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xxw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xxw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xxw(
    vec_uint4_arg xyzw
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yyzz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yyzz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yyzz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xzyw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xzyw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xzyw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yxzw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yxzw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yxzw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yxwz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yxwz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yxwz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yzxy(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yzxy(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yzxy(
    vec_uint4_arg xyzw
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yzxw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yzxw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yzxw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yzzw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yzzw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yzzw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zxyz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zxyz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zxyz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_zxyw(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_zxyw(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zxyw(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_wyzz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_wyzz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_wyzz(
    vec_uint4_arg xyzw
);

```

SCE CONFIDENTIAL

```

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyyy(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyyy(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyyy(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xyzz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xyzz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xyzz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_xxzz(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_xxzz(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_xxzz(
    vec_uint4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_yyww(
    vec_float4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_yyww(
    vec_int4_arg xyzw
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_yyww(
    vec_uint4_arg xyzw
);

```

## Arguments

[in] *xyzw* The vector to swizzle.

## Return Values

The swizzled vector.

## Description

Swizzles a 128-bit vector.

# Swizzle (64-bit)

## Functions

Swizzles a 64-bit vector.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_xx(
    vec_float2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_xx(
    vec_int2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_xx(
    vec_uint2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_yy(
    vec_float2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_yy(
    vec_int2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_yy(
    vec_uint2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_xy(
    vec_float2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_xy(
    vec_int2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_xy(
    vec_uint2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_yx(
    vec_float2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_yx(
    vec_int2_arg xy
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_yx(
    vec_uint2_arg xy
);
```

SCE CONFIDENTIAL

---

## **Arguments**

---

[in]  $xy$  The vector to swizzle.

---

## **Return Values**

---

The swizzled vector.

---

## **Description**

---

Swizzles a 64-bit vector.

# Swizzle (Template Arguments)

## Functions

Swizzles a vector based on template arguments. A negative value should be used to indicate undefined/don't care.

### Definition

```
#include <vectormath.h>
template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_double1
sce_vectormath_shuffle1t(
    vec_double1_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_double2
sce_vectormath_shuffle1t(
    vec_double2_arg v0
);

template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1
sce_vectormath_shuffle1t(
    vec_ullong1_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2
sce_vectormath_shuffle1t(
    vec_ullong2_arg v0
);

template <int X> SCE_VECTORMATH_ALWAYS_INLINE vec_llong1
sce_vectormath_shuffle1t(
    vec_llong1_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_llong2
sce_vectormath_shuffle1t(
    vec_llong2_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_float2
sce_vectormath_shuffle1t(
    vec_float2_arg v0
);

template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_float4
sce_vectormath_shuffle1t(
    vec_float4_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_uint2
sce_vectormath_shuffle1t(
    vec_uint2_arg v0
);
```

SCE CONFIDENTIAL

```
template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_uint4
sce_vectormath_shuffle1t
    vec_uint4_arg v0
);

template <int X, int Y> SCE_VECTORMATH_ALWAYS_INLINE vec_int2
sce_vectormath_shuffle1t
    vec_int2_arg v0
);

template <int X, int Y, int Z, int W> SCE_VECTORMATH_ALWAYS_INLINE vec_int4
sce_vectormath_shuffle1t
    vec_int4_arg v0
);
```

## **Arguments**

[in] *v0* The vector to swizzle.

## **Return Values**

The swizzled vector.

## **Description**

Swizzles a vector based on template arguments. A negative value should be used to indicate undefined/don't care.

# Swizzling

## Functions

Swizzles lanes in a vector using runtime lane scalar indices.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_float2 sce_vectormath_swizzle(
    vec_float2_arg v,
    uint32_t idx0,
    uint32_t idx1
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int2 sce_vectormath_swizzle(
    vec_int2_arg v,
    uint32_t idx0,
    uint32_t idx1
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_swizzle(
    vec_uint2_arg v,
    uint32_t idx0,
    uint32_t idx1
);

SCE_VECTORMATH_ALWAYS_INLINE vec_float4 sce_vectormath_swizzle(
    vec_float4_arg v,
    uint32_t idx0,
    uint32_t idx1,
    uint32_t idx2,
    uint32_t idx3
);

SCE_VECTORMATH_ALWAYS_INLINE vec_int4 sce_vectormath_swizzle(
    vec_int4_arg v,
    uint32_t idx0,
    uint32_t idx1,
    uint32_t idx2,
    uint32_t idx3
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_swizzle(
    vec_uint4_arg v,
    uint32_t idx0,
    uint32_t idx1,
    uint32_t idx2,
    uint32_t idx3
);
```

### Arguments

[in] <i>v</i>	The vector to swizzle.
[in] <i>idx0</i>	The index of the lane of the supplied vector to be inserted into the first lane of the result.

SCE CONFIDENTIAL

---

[in] <i>idx1</i>	The index of the lane of the supplied vector to be inserted into the second lane of the result.
<i>idx2</i>	The index of the lane of the supplied vector to be inserted into the third lane of the result.
<i>idx3</i>	The index of the lane of the supplied vector to be inserted into the fourth lane of the result.

### **Return Values**

The swizzled vector.

### **Description**

Swizzles lanes in a vector using runtime lane scalar indices.

### **Notes**

CAUTION - VERY SLOW ON SOME PLATFORMS!

# Transposition

## Functions

Transpose the lanes of either two or four vectors.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_float2 &a,
    vec_float2 &b
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_uint2 &a,
    vec_uint2 &b
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_int2 &a,
    vec_int2 &b
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_float4 &a,
    vec_float4 &b,
    vec_float4 &c,
    vec_float4 &d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_int4 &a,
    vec_int4 &b,
    vec_int4 &c,
    vec_int4 &d
);

SCE_VECTORMATH_ALWAYS_INLINE void sce_vectormath_transpose(
    vec_uint4 &a,
    vec_uint4 &b,
    vec_uint4 &c,
    vec_uint4 &d
);
```

### Arguments

[in,out] a	The first vector to be transposed.
[in,out] b	The second vector to be transposed.
[in,out] c	The third vector to be transposed.
[in,out] d	The fourth vector to be transposed.

### Return Values

None

SCE CONFIDENTIAL

**Description**

---

Transposes the lanes of either two or four vectors.

000004892117

# Unsigned Integer Zero Extension

## Functions

Extends a whole, lower half or upper half vector unsigned integer vector by pre-pending zero.

### Definition

```
#include <vectormath.h>
SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_zero_extend(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zero_extend(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_zero_extend(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_zero_extend_low(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_zero_extend_low(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_zero_extend_low(
    vec_ushort4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zero_extend_low(
    vec_ushort8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_zero_extend_low(
    vec_uchar8_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_zero_extend_low(
    vec_uchar16_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong1 sce_vectormath_zero_extend_high(
    vec_uint2_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_ullong2 sce_vectormath_zero_extend_high(
    vec_uint4_arg a
);

SCE_VECTORMATH_ALWAYS_INLINE vec_uint2 sce_vectormath_zero_extend_high(
    vec_ushort4_arg a
);
```

SCE CONFIDENTIAL

```
SCE_VECTORMATH_ALWAYS_INLINE vec_uint4 sce_vectormath_zero_extend_high(
    vec_ushort8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort4 sce_vectormath_zero_extend_high(
    vec_uchar8_arg a
) ;

SCE_VECTORMATH_ALWAYS_INLINE vec_ushort8 sce_vectormath_zero_extend_high(
    vec_uchar16_arg a
) ;
```

## Arguments

[in] *a* An unsigned integer vector.

## Return Values

The zero extended unsigned integer vector.

## Description

Extends a whole, lower half or upper half vector unsigned integer vector by pre-pending zero.