

# **Vector Math Library C++ PPU Reference**

© 2007 Sony Computer Entertainment Inc.  
All Rights Reserved.

---

# Table of Contents

---

<b>Introduction.....</b>	<b>23</b>
Library Summary .....	24
<b>Vectormath .....</b>	<b>25</b>
Summary .....	26
Vectormath .....	26
<b>Vectormath::Aos .....</b>	<b>27</b>
Summary .....	28
Vectormath::Aos .....	28
3-D Vector Functions.....	33
absPerElem.....	33
copySignPerElem.....	34
cross.....	35
crossMatrix.....	36
crossMatrixMul .....	37
divPerElem.....	38
dot .....	39
length .....	40
lengthSqr .....	41
lerp .....	42
lerp .....	43
loadXYZArray.....	44
maxElem .....	45
maxPerElem.....	46
minElem .....	47
minPerElem.....	48
mulPerElem.....	49
normalize.....	50
operator * .....	51
operator * .....	52
outer .....	53
print .....	54
print .....	55
recipPerElem.....	56
rowMul.....	57
rsqrtPerElem .....	58
select.....	59
select.....	60
slerp.....	61
slerp.....	62
sqrtPerElem .....	63
storeHalfFloats .....	64
storeXYZ .....	65
storeXYZArray.....	66
sum.....	67

---

4-D Vector Functions.....	68
absPerElem.....	68
copySignPerElem.....	69
divPerElem.....	70
dot .....	71
length .....	72
lengthSqr .....	73
lerp .....	74
lerp .....	75
maxElem .....	76
maxPerElem.....	77
minElem .....	78
minPerElem.....	79
mulPerElem.....	80
normalize.....	81
operator * .....	82
operator * .....	83
outer .....	84
print .....	85
print .....	86
recipPerElem.....	87
rsqrtPerElem .....	88
select.....	89
select.....	90
slerp.....	91
slerp.....	92
sqrtPerElem .....	93
storeHalfFloats .....	94
sum.....	95
3-D Point Functions.....	96
absPerElem.....	96
copySignPerElem.....	97
dist.....	98
distFromOrigin.....	99
distSqr .....	100
distSqrFromOrigin .....	101
divPerElem.....	102
lerp .....	103
lerp .....	104
loadXYZArray.....	105
maxElem .....	106
maxPerElem.....	107
minElem .....	108
minPerElem.....	109
mulPerElem.....	110
print .....	111
print .....	112
projection.....	113

---

recipPerElem.....	114
rsqrtPerElem .....	115
scale.....	116
scale.....	117
scale.....	118
select.....	119
select.....	120
sqrtPerElem .....	121
storeHalfFloats .....	122
storeXYZ .....	123
storeXYZArray.....	124
sum.....	125
Quaternion Functions.....	126
conj.....	126
dot .....	127
length .....	128
lerp .....	129
lerp .....	130
norm .....	131
normalize.....	132
operator * .....	133
operator * .....	134
print .....	135
print .....	136
rotate .....	137
select.....	138
select.....	139
slerp.....	140
slerp.....	141
squad .....	142
squad .....	143
3x3 Matrix Functions .....	144
absPerElem.....	144
appendScale .....	145
determinant .....	146
inverse.....	147
mulPerElem.....	148
operator * .....	149
operator * .....	150
prependScale .....	151
print .....	152
print .....	153
select.....	154
select.....	155
transpose .....	156
4x4 Matrix Functions .....	157
absPerElem.....	157
affineInverse.....	158

appendScale .....	159
determinant .....	160
inverse.....	161
mulPerElem.....	162
operator * .....	163
operator * .....	164
orthoInverse .....	165
prependScale .....	166
print .....	167
print .....	168
select.....	169
select.....	170
transpose .....	171
3x4 Transformation Matrix Functions .....	172
absPerElem.....	172
appendScale .....	173
inverse.....	174
mulPerElem.....	175
orthoInverse .....	176
prependScale .....	177
print .....	178
print .....	179
select.....	180
select.....	181
<b>Vectormath::Aos::Matrix3 .....</b>	<b>182</b>
Summary .....	183
Vectormath::Aos::Matrix3 .....	183
Constructors and Destructors .....	185
Matrix3 .....	185
Matrix3 .....	186
Matrix3 .....	187
Matrix3 .....	188
Matrix3 .....	189
Matrix3 .....	190
Operator Methods .....	191
operator * .....	191
operator * .....	192
operator * .....	193
operator * .....	194
operator *= .....	195
operator *= .....	196
operator *= .....	197
operator+.....	198
operator+=.....	199
operator-.....	200
operator-.....	201
operator-=.....	202
operator=.....	203

operator[] .....	204
operator[] .....	205
Public Static Methods .....	206
identity .....	206
rotation .....	207
rotation .....	208
rotation .....	209
rotationX .....	210
rotationX .....	211
rotationY .....	212
rotationY .....	213
rotationZ .....	214
rotationZ .....	215
rotationZYX .....	216
scale .....	217
Public Instance Methods .....	218
getCol .....	218
getCol0 .....	219
getCol1 .....	220
getCol2 .....	221
getElem .....	222
getRow .....	223
setCol .....	224
setCol0 .....	225
setCol1 .....	226
setCol2 .....	227
setElem .....	228
setElem .....	229
setRow .....	230
<b>Vectormath::Aos::Matrix4 .....</b>	<b>231</b>
Summary .....	232
Vectormath::Aos::Matrix4 .....	232
Constructors and Destructors .....	234
Matrix4 .....	234
Matrix4 .....	235
Matrix4 .....	236
Matrix4 .....	237
Matrix4 .....	238
Matrix4 .....	239
Matrix4 .....	240
Matrix4 .....	241
Operator Methods .....	242
operator * .....	242
operator * .....	243
operator * .....	244
operator * .....	245
operator * .....	246
operator * .....	247

operator *	248
operator *=	249
operator *=	250
operator *=	251
operator *=	252
operator+	253
operator+=	254
operator-	255
operator-	256
operator-=	257
operator=	258
operator[]	259
operator[]	260
Public Static Methods	261
frustum	261
identity	262
lookAt	263
orthographic	264
perspective	265
rotation	266
rotation	267
rotation	268
rotationX	269
rotationX	270
rotationY	271
rotationY	272
rotationZ	273
rotationZ	274
rotationZYX	275
scale	276
translation	277
Public Instance Methods	278
getCol	278
getCol0	279
getCol1	280
getCol2	281
getCol3	282
getElem	283
getRow	284
getTranslation	285
getUpper3x3	286
setCol	287
setCol0	288
setCol1	289
setCol2	290
setCol3	291
setElem	292
setElem	293

setRow .....	294
setTranslation.....	295
setUpper3x3.....	296
<b>Vectormath::Aos::Point3.....</b>	<b>297</b>
Summary .....	298
Vectormath::Aos::Point3 .....	298
Constructors and Destructors .....	300
Point3 .....	300
Point3 .....	301
Point3 .....	302
Point3 .....	303
Point3 .....	304
Point3 .....	305
Point3 .....	306
Operator Methods .....	307
operator+ .....	307
operator+= .....	308
operator- .....	309
operator- .....	310
operator-= .....	311
operator= .....	312
operator[] .....	313
operator[] .....	314
Public Instance Methods .....	315
get128 .....	315
getElem .....	316
getX .....	317
getY .....	318
getZ .....	319
setElem .....	320
setElem .....	321
setX .....	322
setX .....	323
setY .....	324
setY .....	325
setZ .....	326
setZ .....	327
<b>Vectormath::Aos::Quat .....</b>	<b>328</b>
Summary .....	329
Vectormath::Aos::Quat .....	329
Constructors and Destructors .....	331
Quat.....	331
Quat.....	332
Quat.....	333
Quat.....	334
Quat.....	335
Quat.....	336
Quat.....	337



Quat.....	338
Quat.....	339
Quat.....	340
Operator Methods .....	341
operator * .....	341
operator * .....	342
operator * .....	343
operator *= .....	344
operator *= .....	345
operator *= .....	346
operator+ .....	347
operator+= .....	348
operator- .....	349
operator- .....	350
operator-= .....	351
operator/ .....	352
operator/ .....	353
operator/= .....	354
operator/= .....	355
operator= .....	356
operator[] .....	357
operator[] .....	358
Public Static Methods .....	359
identity .....	359
rotation .....	360
rotation .....	361
rotation .....	362
rotationX .....	363
rotationX .....	364
rotationY .....	365
rotationY .....	366
rotationZ .....	367
rotationZ .....	368
Public Instance Methods .....	369
get128 .....	369
getElem .....	370
getW .....	371
getX .....	372
getXYZ .....	373
getY .....	374
getZ .....	375
setElem .....	376
setElem .....	377
setW .....	378
setW .....	379
setX .....	380
setX .....	381
setXYZ .....	382

setY .....	383
setY .....	384
setZ .....	385
setZ .....	386
<b>Vectormath::Aos::Transform3 .....</b>	<b>387</b>
Summary .....	388
Vectormath::Aos::Transform3 .....	388
Constructors and Destructors .....	390
Transform3 .....	390
Transform3 .....	391
Transform3 .....	392
Transform3 .....	393
Transform3 .....	394
Transform3 .....	395
Transform3 .....	396
Operator Methods .....	397
operator * .....	397
operator * .....	398
operator * .....	399
operator *= .....	400
operator= .....	401
operator[] .....	402
operator[] .....	403
Public Static Methods .....	404
identity .....	404
rotation .....	405
rotation .....	406
rotation .....	407
rotationX .....	408
rotationX .....	409
rotationY .....	410
rotationY .....	411
rotationZ .....	412
rotationZ .....	413
rotationZYX .....	414
scale .....	415
translation .....	416
Public Instance Methods .....	417
getCol .....	417
getCol0 .....	418
getCol1 .....	419
getCol2 .....	420
getCol3 .....	421
getElem .....	422
getRow .....	423
getTranslation .....	424
getUpper3x3 .....	425
setCol .....	426

setCol0 .....	427
setCol1 .....	428
setCol2 .....	429
setCol3 .....	430
setElem .....	431
setElem .....	432
setRow .....	433
setTranslation .....	434
setUpper3x3 .....	435
<b>Vectormath::Aos::Vector3.....</b>	<b>436</b>
Summary .....	437
Vectormath::Aos::Vector3 .....	437
Constructors and Destructors .....	439
Vector3 .....	439
Vector3 .....	440
Vector3 .....	441
Vector3 .....	442
Vector3 .....	443
Vector3 .....	444
Vector3 .....	445
Operator Methods .....	446
operator * .....	446
operator * .....	447
operator *= .....	448
operator *= .....	449
operator+ .....	450
operator+ .....	451
operator+= .....	452
operator- .....	453
operator- .....	454
operator-= .....	455
operator/ .....	456
operator/ .....	457
operator/= .....	458
operator/= .....	459
operator= .....	460
operator[] .....	461
operator[] .....	462
Public Static Methods .....	463
xAxis .....	463
yAxis .....	464
zAxis .....	465
Public Instance Methods .....	466
get128 .....	466
getElem .....	467
getX .....	468
getY .....	469
getZ .....	470

setElem .....	471
setElem .....	472
setX .....	473
setX .....	474
setY .....	475
setY .....	476
setZ .....	477
setZ .....	478
<b>Vectormath::Aos::Vector4.....</b>	<b>479</b>
Summary .....	480
Vectormath::Aos::Vector4 .....	480
Constructors and Destructors .....	482
Vector4 .....	482
Vector4 .....	483
Vector4 .....	484
Vector4 .....	485
Vector4 .....	486
Vector4 .....	487
Vector4 .....	488
Vector4 .....	489
Vector4 .....	490
Vector4 .....	491
Vector4 .....	492
Operator Methods .....	493
operator * .....	493
operator * .....	494
operator *= .....	495
operator *= .....	496
operator+ .....	497
operator+= .....	498
operator- .....	499
operator- .....	500
operator-= .....	501
operator/ .....	502
operator/ .....	503
operator/= .....	504
operator/= .....	505
operator= .....	506
operator[] .....	507
operator[] .....	508
Public Static Methods .....	509
wAxis .....	509
xAxis .....	510
yAxis .....	511
zAxis .....	512
Public Instance Methods .....	513
get128 .....	513
getElem .....	514

getW .....	515
getX .....	516
getXYZ .....	517
getY .....	518
getZ .....	519
setElem .....	520
setElem .....	521
setW .....	522
setW .....	523
setX .....	524
setX .....	525
setXYZ .....	526
setY .....	527
setY .....	528
setZ .....	529
setZ .....	530
<b>Vectormath::Soa .....</b>	<b>531</b>
Summary .....	532
Vectormath::Soa.....	532
3-D Vector Functions.....	536
absPerElem.....	536
copySignPerElem.....	537
cross.....	538
crossMatrix.....	539
crossMatrixMul .....	540
divPerElem.....	541
dot .....	542
length .....	543
lengthSqr .....	544
lerp .....	545
loadXYZArray.....	546
maxElem .....	547
maxPerElem.....	548
minElem .....	549
minPerElem.....	550
mulPerElem.....	551
normalize.....	552
operator * .....	553
outer .....	554
print .....	555
print .....	556
recipPerElem.....	557
rowMul.....	558
rsqrtPerElem .....	559
select.....	560
slerp.....	561
sqrtPerElem .....	562
storeHalfFloats .....	563

---

storeXYZArray.....	564
sum.....	565
4-D Vector Functions.....	566
absPerElem.....	566
copySignPerElem.....	567
divPerElem.....	568
dot .....	569
length .....	570
lengthSqr .....	571
lerp .....	572
maxElem .....	573
maxPerElem.....	574
minElem .....	575
minPerElem.....	576
mulPerElem.....	577
normalize.....	578
operator * .....	579
outer .....	580
print .....	581
print .....	582
recipPerElem.....	583
rsqrtPerElem .....	584
select.....	585
slerp.....	586
sqrPerElem .....	587
storeHalfFloats .....	588
sum.....	589
3-D Point Functions.....	590
absPerElem.....	590
copySignPerElem.....	591
dist.....	592
distFromOrigin.....	593
distSqr .....	594
distSqrFromOrigin .....	595
divPerElem.....	596
lerp .....	597
loadXYZArray.....	598
maxElem .....	599
maxPerElem.....	600
minElem .....	601
minPerElem.....	602
mulPerElem.....	603
print .....	604
print .....	605
projection.....	606
recipPerElem.....	607
rsqrtPerElem .....	608
scale .....	609

---

scale .....	610
select .....	611
sqrtPerElem .....	612
storeHalfFloats .....	613
storeXYZArray .....	614
sum .....	615
Quaternion Functions .....	616
conj .....	616
dot .....	617
length .....	618
lerp .....	619
norm .....	620
normalize .....	621
operator * .....	622
print .....	623
print .....	624
rotate .....	625
select .....	626
slerp .....	627
squad .....	628
3x3 Matrix Functions .....	629
absPerElem .....	629
appendScale .....	630
determinant .....	631
inverse .....	632
mulPerElem .....	633
operator * .....	634
prependScale .....	635
print .....	636
print .....	637
select .....	638
transpose .....	639
4x4 Matrix Functions .....	640
absPerElem .....	640
affineInverse .....	641
appendScale .....	642
determinant .....	643
inverse .....	644
mulPerElem .....	645
operator * .....	646
orthoInverse .....	647
prependScale .....	648
print .....	649
print .....	650
select .....	651
transpose .....	652
3x4 Transformation Matrix Functions .....	653
absPerElem .....	653

appendScale .....	654
inverse.....	655
mulPerElem.....	656
orthoInverse .....	657
prependScale .....	658
print .....	659
print .....	660
select.....	661
<b>Vectormath::Soa::Matrix3 .....</b>	<b>662</b>
Summary .....	663
Vectormath::Soa::Matrix3.....	663
Constructors and Destructors .....	665
Matrix3 .....	665
Matrix3 .....	666
Matrix3 .....	667
Matrix3 .....	668
Matrix3 .....	669
Matrix3 .....	670
Matrix3 .....	671
Operator Methods .....	672
operator * .....	672
operator * .....	673
operator * .....	674
operator *= .....	675
operator *= .....	676
operator+ .....	677
operator+= .....	678
operator-.....	679
operator-.....	680
operator-= .....	681
operator= .....	682
operator[] .....	683
operator[] .....	684
Public Static Methods .....	685
identity .....	685
rotation .....	686
rotation .....	687
rotationX .....	688
rotationY .....	689
rotationZ .....	690
rotationZYX .....	691
scale .....	692
Public Instance Methods .....	693
get4Aos .....	693
getCol .....	694
getCol0 .....	695
getCol1 .....	696
getCol2 .....	697



getElem .....	698
getRow .....	699
setCol .....	700
setCol0 .....	701
setCol1 .....	702
setCol2 .....	703
setElem .....	704
setRow .....	705
<b>Vectormath::Soa::Matrix4 .....</b>	<b>706</b>
Summary .....	707
Vectormath::Soa::Matrix4 .....	707
Constructors and Destructors .....	709
Matrix4 .....	709
Matrix4 .....	710
Matrix4 .....	711
Matrix4 .....	712
Matrix4 .....	713
Matrix4 .....	714
Matrix4 .....	715
Matrix4 .....	716
Matrix4 .....	717
Operator Methods .....	718
operator * .....	718
operator * .....	719
operator * .....	720
operator * .....	721
operator * .....	722
operator * .....	723
operator *= .....	724
operator *= .....	725
operator *= .....	726
operator+ .....	727
operator+= .....	728
operator- .....	729
operator- .....	730
operator-= .....	731
operator= .....	732
operator[] .....	733
operator[] .....	734
Public Static Methods .....	735
frustum .....	735
identity .....	736
lookAt .....	737
orthographic .....	738
perspective .....	739
rotation .....	740
rotation .....	741
rotationX .....	742

rotationY .....	743
rotationZ .....	744
rotationZYX .....	745
scale .....	746
translation .....	747
Public Instance Methods .....	748
get4Aos .....	748
getCol .....	749
getCol0 .....	750
getCol1 .....	751
getCol2 .....	752
getCol3 .....	753
getElem .....	754
getRow .....	755
getTranslation .....	756
getUpper3x3 .....	757
setCol .....	758
setCol0 .....	759
setCol1 .....	760
setCol2 .....	761
setCol3 .....	762
setElem .....	763
setRow .....	764
setTranslation .....	765
setUpper3x3 .....	766
<b>Vectormath::Soa::Point3 .....</b>	<b>767</b>
Summary .....	768
Vectormath::Soa::Point3 .....	768
Constructors and Destructors .....	769
Point3 .....	769
Point3 .....	770
Point3 .....	771
Point3 .....	772
Point3 .....	773
Point3 .....	774
Point3 .....	775
Operator Methods .....	776
operator+ .....	776
operator+= .....	777
operator- .....	778
operator- .....	779
operator-= .....	780
operator= .....	781
operator[] .....	782
operator[] .....	783
Public Instance Methods .....	784
get4Aos .....	784
getElem .....	785

getX.....	786
getY.....	787
getZ.....	788
setElem.....	789
setX.....	790
setY.....	791
setZ.....	792
<b>Vectormath::Soa::Quat.....</b>	<b>793</b>
Summary.....	794
Vectormath::Soa::Quat.....	794
Constructors and Destructors.....	796
Quat.....	796
Quat.....	797
Quat.....	798
Quat.....	799
Quat.....	800
Quat.....	801
Quat.....	802
Quat.....	803
Quat.....	804
Operator Methods.....	805
operator *.....	805
operator *.....	806
operator *=.....	807
operator *=.....	808
operator+.....	809
operator+=.....	810
operator-.....	811
operator-.....	812
operator-=.....	813
operator/.....	814
operator/=.....	815
operator=.....	816
operator[].....	817
operator[].....	818
Public Static Methods.....	819
identity.....	819
rotation.....	820
rotation.....	821
rotationX.....	822
rotationY.....	823
rotationZ.....	824
Public Instance Methods.....	825
get4Aos.....	825
getElem.....	826
getW.....	827
getX.....	828
getXYZ.....	829

getY .....	830
getZ .....	831
setElem .....	832
setW .....	833
setX .....	834
setXYZ .....	835
setY .....	836
setZ .....	837
<b>Vectormath::Soa::Transform3 .....</b>	<b>838</b>
Summary .....	839
Vectormath::Soa::Transform3 .....	839
Constructors and Destructors .....	841
Transform3 .....	841
Transform3 .....	842
Transform3 .....	843
Transform3 .....	844
Transform3 .....	845
Transform3 .....	846
Transform3 .....	847
Transform3 .....	848
Operator Methods .....	849
operator * .....	849
operator * .....	850
operator * .....	851
operator *= .....	852
operator= .....	853
operator[] .....	854
operator[] .....	855
Public Static Methods .....	856
identity .....	856
rotation .....	857
rotation .....	858
rotationX .....	859
rotationY .....	860
rotationZ .....	861
rotationZYX .....	862
scale .....	863
translation .....	864
Public Instance Methods .....	865
get4Aos .....	865
getCol .....	866
getCol0 .....	867
getCol1 .....	868
getCol2 .....	869
getCol3 .....	870
getElem .....	871
getRow .....	872
getTranslation .....	873

getUpper3x3.....	874
setCol .....	875
setCol0 .....	876
setCol1 .....	877
setCol2 .....	878
setCol3 .....	879
setElem .....	880
setRow .....	881
setTranslation.....	882
setUpper3x3.....	883
<b>Vectormath::Soa::Vector3.....</b>	<b>884</b>
Summary .....	885
Vectormath::Soa::Vector3 .....	885
Constructors and Destructors .....	887
Vector3 .....	887
Vector3 .....	888
Vector3 .....	889
Vector3 .....	890
Vector3 .....	891
Vector3 .....	892
Vector3 .....	893
Operator Methods .....	894
operator * .....	894
operator *= .....	895
operator+ .....	896
operator+ .....	897
operator+= .....	898
operator- .....	899
operator- .....	900
operator-= .....	901
operator/ .....	902
operator/= .....	903
operator= .....	904
operator[] .....	905
operator[] .....	906
Public Static Methods .....	907
xAxis .....	907
yAxis .....	908
zAxis .....	909
Public Instance Methods .....	910
get4Aos .....	910
getElem .....	911
getX .....	912
getY .....	913
getZ .....	914
setElem .....	915
setX .....	916
setY .....	917

setZ .....	918
<b>Vectormath::Soa::Vector4.....</b>	<b>919</b>
Summary .....	920
Vectormath::Soa::Vector4 .....	920
Constructors and Destructors .....	922
Vector4 .....	922
Vector4 .....	923
Vector4 .....	924
Vector4 .....	925
Vector4 .....	926
Vector4 .....	927
Vector4 .....	928
Vector4 .....	929
Vector4 .....	930
Vector4 .....	931
Operator Methods .....	932
operator * .....	932
operator *= .....	933
operator+ .....	934
operator+= .....	935
operator- .....	936
operator- .....	937
operator-= .....	938
operator/ .....	939
operator/= .....	940
operator= .....	941
operator[] .....	942
operator[] .....	943
Public Static Methods .....	944
wAxis .....	944
xAxis .....	945
yAxis .....	946
zAxis .....	947
Public Instance Methods .....	948
get4Aos .....	948
getElem .....	949
getW .....	950
getX .....	951
getXYZ .....	952
getY .....	953
getZ .....	954
setElem .....	955
setW .....	956
setX .....	957
setXYZ .....	958
setY .....	959
setZ .....	960

---

# Introduction

---

# Library Summary

---

## Library Contents

---

Item	Description
<a href="#">Vectormath</a>	The namespace containing the Vectormath library.
<a href="#">Vectormath::Aos</a>	The namespace containing array-of-structures (AoS) classes.
<a href="#">Vectormath::Aos::Matrix3</a>	A 3x3 matrix in array-of-structures format.
<a href="#">Vectormath::Aos::Matrix4</a>	A 4x4 matrix in array-of-structures format.
<a href="#">Vectormath::Aos::Point3</a>	A 3-D point in array-of-structures format.
<a href="#">Vectormath::Aos::Quat</a>	A quaternion in array-of-structures format.
<a href="#">Vectormath::Aos::Transform3</a>	A 3x4 transformation matrix in array-of-structures format.
<a href="#">Vectormath::Aos::Vector3</a>	A 3-D vector in array-of-structures format.
<a href="#">Vectormath::Aos::Vector4</a>	A 4-D vector in array-of-structures format.
<a href="#">Vectormath::Soa</a>	The namespace containing structure-of-arrays (SoA) classes.
<a href="#">Vectormath::Soa::Matrix3</a>	A set of four 3x3 matrices in structure-of-arrays format.
<a href="#">Vectormath::Soa::Matrix4</a>	A set of four 4x4 matrices in structure-of-arrays format.
<a href="#">Vectormath::Soa::Point3</a>	A set of four 3-D points in structure-of-arrays format.
<a href="#">Vectormath::Soa::Quat</a>	A set of four quaternions in structure-of-arrays format.
<a href="#">Vectormath::Soa::Transform3</a>	A set of four 3x4 transformation matrices in structure-of-arrays format.
<a href="#">Vectormath::Soa::Vector3</a>	A set of four 3-D vectors in structure-of-arrays format.
<a href="#">Vectormath::Soa::Vector4</a>	A set of four 4-D vectors in structure-of-arrays format.



---

# **Vectormath**

---

# Summary

## 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="#">Vectormath::Aos</a>	The namespace containing array-of-structures (AoS) classes.
<a href="#">Vectormath::Soa</a>	The namespace containing structure-of-arrays (SoA) classes.

---

# **Vectormath::Aos**

# Summary

## Vectormath::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 3-D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4-D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3-D 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="#">appendScale</a>	Append (post-multiply) a scale transformation to a 3x3 matrix.
<a href="#">appendScale</a>	Append (post-multiply) a scale transformation to a 4x4 matrix.
<a href="#">appendScale</a>	Append (post-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#">conj</a>	Compute the conjugate of a quaternion.
<a href="#">copySignPerElem</a>	Copy sign from one 3-D vector to another, per element.
<a href="#">copySignPerElem</a>	Copy sign from one 4-D vector to another, per element.
<a href="#">copySignPerElem</a>	Copy sign from one 3-D point to another, per element.
<a href="#">cross</a>	Compute cross product of two 3-D vectors.
<a href="#">crossMatrix</a>	Cross-product matrix of a 3-D vector.
<a href="#">crossMatrixMul</a>	Create cross-product matrix and multiply.
<a href="#">determinant</a>	Determinant of a 3x3 matrix.
<a href="#">determinant</a>	Determinant of a 4x4 matrix.
<a href="#">dist</a>	Compute the distance between two 3-D points.
<a href="#">distFromOrigin</a>	Compute the distance of a 3-D point from the coordinate-system origin.
<a href="#">distSqr</a>	Compute the square of the distance between two 3-D points.
<a href="#">distSqrFromOrigin</a>	Compute the square of the distance of a 3-D point from the coordinate-system origin.
<a href="#">divPerElem</a>	Divide two 3-D vectors per element.
<a href="#">divPerElem</a>	Divide two 4-D vectors per element.
<a href="#">divPerElem</a>	Divide two 3-D points per element.
<a href="#">dot</a>	Compute the dot product of two 3-D vectors.
<a href="#">dot</a>	Compute the dot product of two 4-D vectors.

Function	Description
<a href="#"><u>dot</u></a>	Compute the dot product of two quaternions.
<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 3-D vector.
<a href="#"><u>length</u></a>	Compute the length of a 4-D 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 3-D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 4-D vector.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D vectors (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4-D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4-D vectors (scalar data contained in vector data type).
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D points.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D 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>loadXYZArray</u></a>	Load four three-float 3-D vectors, stored in three quadwords.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3-D points, stored in three quadwords.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3-D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 4-D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3-D point.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3-D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 4-D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3-D points per element.
<a href="#"><u>minElem</u></a>	Minimum element of a 3-D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 4-D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3-D point.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3-D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 4-D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3-D points per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3-D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4-D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3-D 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 3-D vector.
<a href="#"><u>normalize</u></a>	Normalize a 4-D vector.
<a href="#"><u>normalize</u></a>	Normalize a quaternion.
<a href="#"><u>operator *</u></a>	Multiply a 3-D vector by a scalar.
<a href="#"><u>operator *</u></a>	Multiply a 3-D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator *</u></a>	Multiply a 4-D vector by a scalar.
<a href="#"><u>operator *</u></a>	Multiply a 4-D vector by a scalar (scalar data contained in vector data type).
<a href="#"><u>operator *</u></a>	Multiply a quaternion by a scalar.

Function	Description
<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 3-D vectors.
<a href="#"><u>outer</u></a>	Outer product of two 4-D vectors.
<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 3-D vector.
<a href="#"><u>print</u></a>	Print a 3-D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 4-D vector.
<a href="#"><u>print</u></a>	Print a 4-D vector and an associated string identifier.
<a href="#"><u>print</u></a>	Print a 3-D point.
<a href="#"><u>print</u></a>	Print a 3-D 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 3-D point on a unit-length 3-D vector.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3-D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 4-D vector per element.
<a href="#"><u>recipPerElem</u></a>	Compute the reciprocal of a 3-D point per element.
<a href="#"><u>rotate</u></a>	Use a unit-length quaternion to rotate a 3-D vector.
<a href="#"><u>rowMul</u></a>	Pre-multiply a row vector by a 3x3 matrix.
<a href="#"><u>rsqrtPerElem</u></a>	Compute the reciprocal square root of a 3-D vector per element.
<a href="#"><u>rsqrtPerElem</u></a>	Compute the reciprocal square root of a 4-D vector per element.
<a href="#"><u>rsqrtPerElem</u></a>	Compute the reciprocal square root of a 3-D point per element.
<a href="#"><u>scale</u></a>	Apply uniform scale to a 3-D point.
<a href="#"><u>scale</u></a>	Apply uniform scale to a 3-D point (scalar data contained in vector data type).
<a href="#"><u>scale</u></a>	Apply non-uniform scale to a 3-D point.
<a href="#"><u>select</u></a>	Conditionally select between two 3-D vectors.

Function	Description
<a href="#">select</a>	Conditionally select between two 3-D vectors (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two 4-D vectors.
<a href="#">select</a>	Conditionally select between two 4-D vectors (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two 3-D points.
<a href="#">select</a>	Conditionally select between two 3-D points (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two quaternions.
<a href="#">select</a>	Conditionally select between two quaternions (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two 3x3 matrices.
<a href="#">select</a>	Conditionally select between two 3x3 matrices (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two 4x4 matrices.
<a href="#">select</a>	Conditionally select between two 4x4 matrices (scalar data contained in vector data type).
<a href="#">select</a>	Conditionally select between two 3x4 transformation matrices.
<a href="#">select</a>	Conditionally select between two 3x4 transformation matrices (scalar data contained in vector data type).
<a href="#">slerp</a>	Spherical linear interpolation between two 3-D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two 3-D vectors (scalar data contained in vector data type).
<a href="#">slerp</a>	Spherical linear interpolation between two 4-D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two 4-D vectors (scalar data contained in vector data type).
<a href="#">slerp</a>	Spherical linear interpolation between two quaternions.
<a href="#">slerp</a>	Spherical linear interpolation between two quaternions (scalar data contained in vector data type).
<a href="#">sqrtPerElem</a>	Compute the square root of a 3-D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 4-D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 3-D point per element.
<a href="#">squad</a>	Spherical quadrangle interpolation.
<a href="#">squad</a>	Spherical quadrangle interpolation (scalar data contained in vector data type).
<a href="#">storeHalfFloats</a>	Store eight 3-D vectors as half-floats.
<a href="#">storeHalfFloats</a>	Store four 4-D vectors as half-floats.
<a href="#">storeHalfFloats</a>	Store eight 3-D points as half-floats.
<a href="#">storeXYZ</a>	Store x, y, and z elements of a 3-D vector in the first three words of a quadword. The value of the fourth word (the word with the highest address) remains unchanged.
<a href="#">storeXYZ</a>	Store x, y, and z elements of a 3-D point in the first three words of a quadword. The value of the fourth word (the word with the highest address) remains unchanged.
<a href="#">storeXYZArray</a>	Store four 3-D vectors in three quadwords.
<a href="#">storeXYZArray</a>	Store four 3-D points in three quadwords.
<a href="#">sum</a>	Compute the sum of all elements of a 3-D vector.
<a href="#">sum</a>	Compute the sum of all elements of a 4-D vector.
<a href="#">sum</a>	Compute the sum of all elements of a 3-D point.
<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="#"><u>Vectormath::Aos::Matrix3</u></a>	A 3x3 matrix in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Matrix4</u></a>	A 4x4 matrix in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Point3</u></a>	A 3-D point in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Quat</u></a>	A quaternion in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Transform3</u></a>	A 3x4 transformation matrix in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Vector3</u></a>	A 3-D vector in array-of-structures format.
<a href="#"><u>Vectormath::Aos::Vector4</u></a>	A 4-D vector in array-of-structures format.



---

# 3-D Vector Functions

## absPerElem

---

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

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 absPerElem(
            Vector3 vec
        );
    }
}
```

### Arguments

---

*vec* 3-D vector

### Return Values

---

3-D 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 3-D vector.

---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 copySignPerElem(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

3-D 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*.

---

# cross

---

Compute cross product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 cross(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

Cross product of the specified 3-D vectors

## Description

---

Compute cross product of two 3-D vectors.

---

# crossMatrix

---

Cross-product matrix of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 crossMatrix(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Cross-product matrix of *vec*

## Description

---

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

---

# crossMatrixMul

---

Create cross-product matrix and multiply.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 crossMatrixMul(
            Vector3 vec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*vec* 3-D vector  
*mat* 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 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 divPerElem(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Divide two 3-D vectors element by element.

## Notes

---

Floating-point behavior matches standard library function divf4.

---

# dot

---

Compute the dot product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec dot(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

Dot product of the specified 3-D vectors

## Description

---

Compute the dot product of two 3-D vectors.

---

# length

---

Compute the length of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec length(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Length of the specified 3-D vector

## Description

---

Compute the length of a 3-D vector.



---

# lengthSqr

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec lengthSqr(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Square of the length of the specified 3-D vector

## Description

---

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

---

# lerp

---

Linear interpolation between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 lerp(
            float t,
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

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

## Return Values

---

Interpolated 3-D vector

## Description

---

Linearly interpolate between two 3-D vectors.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# lerp

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 lerp(
            floatInVec t,
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*t*      Interpolation parameter  
*vec0*   3-D vector  
*vec1*   3-D vector

## Return Values

---

Interpolated 3-D vector

## Description

---

Linearly interpolate between two 3-D vectors.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# loadXYZArray

---

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

## Definition

---

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

## Arguments

---

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

## Return Values

---

None

## Description

---

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

---

# maxElem

---

Maximum element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec maxElem(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Maximum value of all elements of *vec*

## Description

---

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

---

# maxPerElem

---

Maximum of two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 maxPerElem(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

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

---

# minElem

---

Minimum element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec minElem(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Minimum value of all elements of *vec*

## Description

---

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

---

# minPerElem

---

Minimum of two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 minPerElem(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

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



---

# mulPerElem

---

Multiply two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 mulPerElem(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Multiply two 3-D vectors element by element.

---

# normalize

---

Normalize a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 normalize(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

The specified 3-D vector scaled to unit length

## Description

---

Compute a normalized 3-D vector.

## Notes

---

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

---

# operator \*

---

Multiply a 3-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 operator *(
            float scalar,
            Vector3 vec
        );
    }
}
```

## Arguments

---

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

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 3-D vector by a scalar.

---

# operator \*

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 operator *(
            floatInVec scalar,
            Vector3 vec
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)  
*vec*       3-D vector

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 3-D vector by a scalar.

---

# outer

---

Outer product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 outer(
            Vector3 vec0,
            Vector3 vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Compute the outer product of two 3-D vectors.

---

# print

---

Print a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3-D vector and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Vector3 vec,
            const char *name
        );
    }
}
```

## Arguments

---

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

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# recipPerElem

---

Compute the reciprocal of a 3-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 recipPerElem(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

Floating-point behavior matches standard library function recipf4.



---

# rowMul

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 rowMul (
            Vector3 vec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*vec* 3-D vector  
*mat* 3x3 matrix

## Return Values

---

Product of a row-vector and a 3x3 matrix

## Description

---

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

## Notes

---

Slower than column post-multiply.

---

# rsqrtPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 rsqrtPerElem(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# select

---

Conditionally select between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 select(
            Vector3 vec0,
            Vector3 vec1,
            bool select1
        );
    }
}
```

## Arguments

---

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

## Notes

---

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a VMX register may use more processing time than a branch. Use the `boolInVec` version for better performance.

---

# select

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 select(
            Vector3 vec0,
            Vector3 vec1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

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

## Notes

---

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

---

# slerp

---

Spherical linear interpolation between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 slerp(
            float t,
            Vector3 unitVec0,
            Vector3 unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	3-D vector, expected to be unit-length
<i>unitVec1</i>	3-D vector, expected to be unit-length

## Return Values

---

Interpolated 3-D vector

## Description

---

Perform spherical linear interpolation between two 3-D vectors.

## Notes

---

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

---

# slerp

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 slerp(
            floatInVec t,
            Vector3 unitVec0,
            Vector3 unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	3-D vector, expected to be unit-length
<i>unitVec1</i>	3-D vector, expected to be unit-length

## Return Values

---

Interpolated 3-D vector

## Description

---

Perform spherical linear interpolation between two 3-D 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 3-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 sqrtPerElem(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store eight 3-D vectors as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeHalfFloats(
            Vector3 vec0,
            Vector3 vec1,
            Vector3 vec2,
            Vector3 vec3,
            Vector3 vec4,
            Vector3 vec5,
            Vector3 vec6,
            Vector3 vec7,
            vec_ushort8 *threeQuads
        );
    }
}
```

## Arguments

---

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

## Return Values

---

None

## Description

---

Store eight 3-D vectors in three quadwords of half-float values. 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}.



---

# storeXYZ

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeXYZ(
            Vector3 vec,
            vec_float4 *quad
        );
    }
}
```

## Arguments

---

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

## Return Values

---

None

## Description

---

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

---

# storeXYZArray

---

Store four 3-D vectors in three quadwords.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeXYZArray(
            Vector3 vec0,
            Vector3 vec1,
            Vector3 vec2,
            Vector3 vec3,
            vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

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

## Return Values

---

None

## Description

---

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

---

# sum

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec sum(
            Vector3 vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Sum of all elements of *vec*

## Description

---

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

---

# 4-D Vector Functions

## absPerElem

---

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

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 absPerElem(
            Vector4 vec
        );
    }
}
```

### Arguments

---

*vec* 4-D vector

### Return Values

---

4-D 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 4-D vector.

---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 copySignPerElem(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

4-D 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*.

---

# divPerElem

---

Divide two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 divPerElem(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Divide two 4-D vectors element by element.

## Notes

---

Floating-point behavior matches standard library function divf4.

---

# dot

---

Compute the dot product of two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec dot(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

Dot product of the specified 4-D vectors

## Description

---

Compute the dot product of two 4-D vectors.

---

# length

---

Compute the length of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec length(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Length of the specified 4-D vector

## Description

---

Compute the length of a 4-D vector.



---

# lengthSqr

---

Compute the square of the length of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec lengthSqr(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Square of the length of the specified 4-D vector

## Description

---

Compute the square of the length of a 4-D vector.

---

# lerp

---

Linear interpolation between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 lerp(
            float t,
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*t*      Interpolation parameter  
*vec0*   4-D vector  
*vec1*   4-D vector

## Return Values

---

Interpolated 4-D vector

## Description

---

Linearly interpolate between two 4-D vectors.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# lerp

---

Linear interpolation between two 4-D vectors (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 lerp(
            floatInVec t,
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

$t$	Interpolation parameter
$vec0$	4-D vector
$vec1$	4-D vector

## Return Values

---

Interpolated 4-D vector

## Description

---

Linearly interpolate between two 4-D vectors.

## Notes

---

Does not clamp  $t$  between 0 and 1.

---

# maxElem

---

Maximum element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec maxElem(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Maximum value of all elements of *vec*

## Description

---

Compute the maximum value of all elements of a 4-D vector.

---

# maxPerElem

---

Maximum of two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 maxPerElem(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Create a 4-D vector in which each element is the maximum of the corresponding elements of the specified 4-D vectors.

---

# minElem

---

Minimum element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec minElem(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Minimum value of all elements of *vec*

## Description

---

Compute the minimum value of all elements of a 4-D vector.

---

# minPerElem

---

Minimum of two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 minPerElem(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Create a 4-D vector in which each element is the minimum of the corresponding elements of two specified 4-D vectors.

---

# mulPerElem

---

Multiply two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 mulPerElem(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Multiply two 4-D vectors element by element.



---

# normalize

---

Normalize a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 normalize(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

The specified 4-D vector scaled to unit length

## Description

---

Compute a normalized 4-D vector.

## Notes

---

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

---

# operator \*

---

Multiply a 4-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 operator *(
            float scalar,
            Vector4 vec
        );
    }
}
```

## Arguments

---

<i>scalar</i>	Scalar value
<i>vec</i>	4-D vector

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 4-D vector by a scalar.

---

# operator \*

---

Multiply a 4-D vector by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 operator *(
            floatInVec scalar,
            Vector4 vec
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)  
*vec*        4-D vector

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 4-D vector by a scalar.

---

# outer

---

Outer product of two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 outer(
            Vector4 vec0,
            Vector4 vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

The 4x4 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

---

Compute the outer product of two 4-D vectors.

---

# print

---

Print a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

None

## Description

---

Print a 4-D vector. Prints the 4-D vector transposed, that is, as a row instead of a column.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 4-D vector and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Vector4 vec,
            const char *name
        );
    }
}
```

## Arguments

---

<i>vec</i>	4-D vector
<i>name</i>	String printed with the 4-D vector

## Return Values

---

None

## Description

---

Print a 4-D vector and an associated string identifier. Prints the 4-D vector transposed, that is, as a row instead of a column.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# recipPerElem

---

Compute the reciprocal of a 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 recipPerElem(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the reciprocal of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the reciprocal of the corresponding element of the specified 4-D vector.

## Notes

---

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

---

# rsqrtPerElem

---

Compute the reciprocal square root of a 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 rsqrtPerElem(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the reciprocal square root of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the reciprocal square root of the corresponding element of the specified 4-D vector.

## Notes

---

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



---

# select

---

Conditionally select between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 select(
            Vector4 vec0,
            Vector4 vec1,
            bool select1
        );
    }
}
```

## Arguments

---

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

## Notes

---

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a VMX register may use more processing time than a branch. Use the `boolInVec` version for better performance.

---

# select

---

Conditionally select between two 4-D vectors (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 select(
            Vector4 vec0,
            Vector4 vec1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

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

## Notes

---

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

---

# slerp

---

Spherical linear interpolation between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 slerp(
            float t,
            Vector4 unitVec0,
            Vector4 unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	4-D vector, expected to be unit-length
<i>unitVec1</i>	4-D vector, expected to be unit-length

## Return Values

---

Interpolated 4-D vector

## Description

---

Perform spherical linear interpolation between two 4-D vectors.

## Notes

---

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

---

# slerp

---

Spherical linear interpolation between two 4-D vectors (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 slerp(
            floatInVec t,
            Vector4 unitVec0,
            Vector4 unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	4-D vector, expected to be unit-length
<i>unitVec1</i>	4-D vector, expected to be unit-length

## Return Values

---

Interpolated 4-D vector

## Description

---

Perform spherical linear interpolation between two 4-D 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 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector4 sqrtPerElem(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the square root of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the square root of the corresponding element of the specified 4-D vector.

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store four 4-D vectors as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeHalfFloats(
            Vector4 vec0,
            Vector4 vec1,
            Vector4 vec2,
            Vector4 vec3,
            vec_ushort8 *twoQuads
        );
    }
}
```

## Arguments

---

<i>vec0</i>	4-D vector
<i>vec1</i>	4-D vector
<i>vec2</i>	4-D vector
<i>vec3</i>	4-D vector
<i>twoQuads</i>	An output array of 2 quadwords containing 16 half-floats

## Return Values

---

None

## Description

---

Store four 4-D 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}.

---

# sum

---

Compute the sum of all elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec sum(
            Vector4 vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Sum of all elements of *vec*

## Description

---

Compute the sum of all elements of a 4-D vector.

---

# 3-D Point Functions

## absPerElem

---

Compute the absolute value of a 3-D point per element.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 absPerElem(
            Point3 pnt
        );
    }
}
```

### Arguments

---

*pnt* 3-D point

### Return Values

---

3-D 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 3-D point.



---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 copySignPerElem(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D 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*.

---

# dist

---

Compute the distance between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec dist(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

Distance between two 3-D points

## Description

---

Compute the distance between two 3-D points.

---

# distFromOrigin

---

Compute the distance of a 3-D point from the coordinate-system origin.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec distFromOrigin(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Distance of a 3-D point from the origin

## Description

---

Compute the distance of a 3-D point from the coordinate-system origin.

---

# distSqr

---

Compute the square of the distance between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec distSqr(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

Square of the distance between two 3-D points

## Description

---

Compute the square of the distance between two 3-D points.

---

# distSqrFromOrigin

---

Compute the square of the distance of a 3-D point from the coordinate-system origin.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec distSqrFromOrigin(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Square of the distance of a 3-D point from the origin

## Description

---

Compute the square of the distance of a 3-D point from the coordinate-system origin.

---

# divPerElem

---

Divide two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 divPerElem(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the quotient of the corresponding elements of the specified 3-D points

## Description

---

Divide two 3-D points element by element.

## Notes

---

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

---

# lerp

---

Linear interpolation between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 lerp(
            float t,
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point

## Return Values

---

Interpolated 3-D point

## Description

---

Linearly interpolate between two 3-D points.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# lerp

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 lerp(
            floatInVec t,
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point

## Return Values

---

Interpolated 3-D point

## Description

---

Linearly interpolate between two 3-D points.

## Notes

---

Does not clamp *t* between 0 and 1.



---

# loadXYZArray

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void loadXYZArray(
            Point3 &pnt0,
            Point3 &pnt1,
            Point3 &pnt2,
            Point3 &pnt3,
            const vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	An output 3-D point
<i>pnt1</i>	An output 3-D point
<i>pnt2</i>	An output 3-D point
<i>pnt3</i>	An output 3-D point
<i>threeQuads</i>	Array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

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

---

# maxElem

---

Maximum element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec maxElem(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Maximum value of all elements of *pnt*

## Description

---

Compute the maximum value of all elements of a 3-D point.

---

# maxPerElem

---

Maximum of two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 maxPerElem(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the maximum of the corresponding elements of the specified 3-D points

## Description

---

Create a 3-D point in which each element is the maximum of the corresponding elements of the specified 3-D points.

---

# minElem

---

Minimum element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec minElem(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Minimum value of all elements of *pnt*

## Description

---

Compute the minimum value of all elements of a 3-D point.

---

# minPerElem

---

Minimum of two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 minPerElem(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the minimum of the corresponding elements of the specified 3-D points

## Description

---

Create a 3-D point in which each element is the minimum of the corresponding elements of two specified 3-D points.

---

# mulPerElem

---

Multiply two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 mulPerElem(
            Point3 pnt0,
            Point3 pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the product of the corresponding elements of the specified 3-D points

## Description

---

Multiply two 3-D points element by element.

---

# print

---

Print a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3-D point and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Point3 pnt,
            const char *name
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>name</i>	String printed with the 3-D point

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.



---

# projection

---

Scalar projection of a 3-D point on a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec projection(
            Point3 pnt,
            Vector3 unitVec
        );
    }
}
```

## Arguments

---

*pnt*            3-D point  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

Scalar projection of the 3-D point on the unit-length 3-D vector

## Description

---

Scalar projection of a 3-D point on a unit-length 3-D vector (dot product).

---

# recipPerElem

---

Compute the reciprocal of a 3-D point per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 recipPerElem(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# rsqrtPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 rsqrtPerElem(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# scale

---

Apply uniform scale to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 scale(
            Point3 pnt,
            float scaleVal
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>scaleVal</i>	Scalar value

## Return Values

---

3-D point in which every element is multiplied by the scalar value

## Description

---

Apply uniform scale to a 3-D point.

---

# scale

---

Apply uniform scale to a 3-D point (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 scale(
            Point3 pnt,
            floatInVec scaleVal
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>scaleVal</i>	Scalar value (stored in vector data type)

## Return Values

---

3-D point in which every element is multiplied by the scalar value

## Description

---

Apply uniform scale to a 3-D point.

---

# scale

---

Apply non-uniform scale to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 scale(
            Point3 pnt,
            Vector3 scaleVec
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>scaleVec</i>	3-D vector

## Return Values

---

3-D point in which each element is the product of the corresponding elements of the specified 3-D point and 3-D vector

## Description

---

Apply non-uniform scale to a 3-D point.

---

# select

---

Conditionally select between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 select(
            Point3 pnt0,
            Point3 pnt1,
            bool select1
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>select1</i>	False selects the pnt0 argument, true selects the pnt1 argument

## Return Values

---

Equal to *pnt0* if *select1* is false, or to *pnt1* if *select1* is true

## Description

---

Conditionally select one of the 3-D point arguments.

## Notes

---

This function uses a conditional select instruction to avoid a branch. However, the transfer of *select1* to a VMX register may use more processing time than a branch. Use the `boolInVec` version for better performance.

---

# select

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 select(
            Point3 pnt0,
            Point3 pnt1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>select1</i>	False selects the pnt0 argument, true selects the pnt1 argument

## Return Values

---

Equal to *pnt0* if *select1* is false, or to *pnt1* if *select1* is true

## Description

---

Conditionally select one of the 3-D point arguments.

## Notes

---

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



---

# sqrtPerElem

---

Compute the square root of a 3-D point per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Point3 sqrtPerElem(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store eight 3-D points as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeHalfFloats(
            Point3 pnt0,
            Point3 pnt1,
            Point3 pnt2,
            Point3 pnt3,
            Point3 pnt4,
            Point3 pnt5,
            Point3 pnt6,
            Point3 pnt7,
            vec_ushort8 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>pnt2</i>	3-D point
<i>pnt3</i>	3-D point
<i>pnt4</i>	3-D point
<i>pnt5</i>	3-D point
<i>pnt6</i>	3-D point
<i>pnt7</i>	3-D point
<i>threeQuads</i>	An output array of 3 quadwords containing 24 half-floats

## Return Values

---

None

## Description

---

Store eight 3-D points in three quadwords of half-float values. 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}.

---

# storeXYZ

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeXYZ(
            Point3 pnt,
            vec_float4 *quad
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>quad</i>	Pointer to a quadword in which x, y, and z will be stored

## Return Values

---

None

## Description

---

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

---

# storeXYZArray

---

Store four 3-D points in three quadwords.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void storeXYZArray(
            Point3 pnt0,
            Point3 pnt1,
            Point3 pnt2,
            Point3 pnt3,
            vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>pnt2</i>	3-D point
<i>pnt3</i>	3-D point
<i>threeQuads</i>	An output array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

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

---

# sum

---

Compute the sum of all elements of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec sum(
            Point3 pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Sum of all elements of *pnt*

## Description

---

Compute the sum of all elements of a 3-D point.

---

# Quaternion Functions

## conj

---

Compute the conjugate of a quaternion.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat conj (
            Quat quat
        );
    }
}
```

### Arguments

---

*quat* Quaternion

### Return Values

---

Conjugate of the specified quaternion

### Description

---

Compute the conjugate of a quaternion.

---

# dot

---

Compute the dot product of two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec dot(
            Quat quat0,
            Quat quat1
        );
    }
}
```

## Arguments

---

*quat0* Quaternion  
*quat1* Quaternion

## Return Values

---

Dot product of the specified quaternions

## Description

---

Compute the dot product of two quaternions.

---

# length

---

Compute the length of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec length(
            Quat quat
        );
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Length of the specified quaternion

## Description

---

Compute the length of a quaternion.



---

# lerp

---

Linear interpolation between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat lerp(
            float t,
            Quat quat0,
            Quat 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat lerp(
            floatInVec t,
            Quat quat0,
            Quat 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.

---

# norm

---

Compute the norm of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec norm(
            Quat 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.

---

# normalize

---

Normalize a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat normalize(
            Quat 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.

---

# operator \*

---

Multiply a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat operator *(
            float scalar,
            Quat 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.

---

# operator \*

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat operator *(
            floatInVec scalar,
            Quat quat
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)  
*quat*      Quaternion

## Return Values

---

Scalar product of *quat* and *scalar*

## Description

---

Multiply a quaternion by a scalar.

---

# print

---

Print a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Quat quat
        );
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

None

## Description

---

Print a quaternion.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a quaternion and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            Quat 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 `_VECTORMATH_DEBUG` is defined.



---

# rotate

---

Use a unit-length quaternion to rotate a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Vector3 rotate(
            Quat unitQuat,
            Vector3 vec
        );
    }
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>vec</i>	3-D vector

## Return Values

---

The rotated 3-D vector, equivalent to  $\text{unitQuat} * \text{Quat}(\text{vec}, 0) * \text{conj}(\text{unitQuat})$

## Description

---

Rotate a 3-D vector by applying a unit-length quaternion.

---

# select

---

Conditionally select between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat select(
            Quat quat0,
            Quat quat1,
            bool select1
        );
    }
}
```

## Arguments

---

<i>quat0</i>	Quaternion
<i>quat1</i>	Quaternion
<i>select1</i>	False selects the quat0 argument, true selects the quat1 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 VMX register may use more processing time than a branch. Use the `boolInVec` version for better performance.

---

# select

---

Conditionally select between two quaternions (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat select(
            Quat quat0,
            Quat quat1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

<i>quat0</i>	Quaternion
<i>quat1</i>	Quaternion
<i>select1</i>	False selects the quat0 argument, true selects the quat1 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.

---

# slerp

---

Spherical linear interpolation between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat slerp(
            float t,
            Quat unitQuat0,
            Quat unitQuat1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitQuat0</i>	Quaternion, expected to be unit-length
<i>unitQuat1</i>	Quaternion, expected to be unit-length

## 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.

---

# slerp

---

Spherical linear interpolation between two quaternions (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat slerp(
            floatInVec t,
            Quat unitQuat0,
            Quat unitQuat1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitQuat0</i>	Quaternion, expected to be unit-length
<i>unitQuat1</i>	Quaternion, expected to be unit-length

## 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.

---

# squad

---

Spherical quadrangle interpolation.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat squad(
            float t,
            Quat unitQuat0,
            Quat unitQuat1,
            Quat unitQuat2,
            Quat 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Quat squad(
            floatInVec t,
            Quat unitQuat0,
            Quat unitQuat1,
            Quat unitQuat2,
            Quat 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.

---

# 3x3 Matrix Functions

## absPerElem

---

Compute the absolute value of a 3x3 matrix per element.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 absPerElem(
            const Matrix3 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 appendScale(
            const Matrix3 &mat,
            Vector3 scaleVec
        );
    }
}
```

## Arguments

---

<i>mat</i>	3x3 matrix
<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# determinant

---

Determinant of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec determinant(
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

The determinant of *mat*

## Description

---

Compute the determinant of a 3x3 matrix.

---

# inverse

---

Compute the inverse of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 inverse(
            const Matrix3 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 mulPerElem(
            const Matrix3 &mat0,
            const Matrix3 &mat1
        );
    }
}
```

## Arguments

---

*mat0* 3x3 matrix  
*mat1* 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.

---

# operator \*

---

Multiply a 3x3 matrix by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 operator *(
            float scalar,
            const Matrix3 &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.

---

# operator \*

---

Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 operator *(
            floatInVec scalar,
            const Matrix3 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 prependScale(
            Vector3 scaleVec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# print

---

Print a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Matrix3 &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 `_VECTORMATH_DEBUG` is defined.



---

# print

---

Print a 3x3 matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Matrix3 &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 `_VECTORMATH_DEBUG` is defined.

---

# select

---

Conditionally select between two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 select(
            const Matrix3 &mat0,
            const Matrix3 &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 VMX 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 select(
            const Matrix3 &mat0,
            const Matrix3 &mat1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

<i>mat0</i>	3x3 matrix
<i>mat1</i>	3x3 matrix
<i>select1</i>	False selects the mat0 argument, true selects the mat1 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.

---

# transpose

---

Transpose of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix3 transpose(
            const Matrix3 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 absPerElem(
            const Matrix4 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 affineInverse(
            const Matrix4 &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  $\text{inverse}(M)$ , whose translation vector is  $-\text{inverse}(M)*v$ , 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 appendScale(
            const Matrix4 &mat,
            Vector3 scaleVec
        );
    }
}
```

## Arguments

---

<i>mat</i>	4x4 matrix
<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# determinant

---

Determinant of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const floatInVec determinant(
            const Matrix4 &mat
        );
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

The determinant of *mat*

## Description

---

Compute the determinant of a 4x4 matrix.



---

# inverse

---

Compute the inverse of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 inverse(
            const Matrix4 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 mulPerElem(
            const Matrix4 &mat0,
            const Matrix4 &mat1
        );
    }
}
```

## Arguments

---

*mat0* 4x4 matrix  
*mat1* 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.

---

# operator \*

---

Multiply a 4x4 matrix by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 operator *(
            float scalar,
            const Matrix4 &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.

---

# operator \*

---

Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 operator *(
            floatInVec scalar,
            const Matrix4 &mat
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)  
*mat*        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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 orthoInverse (
            const Matrix4 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 prependScale(
            Vector3 scaleVec,
            const Matrix4 &mat
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# print

---

Print a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Matrix4 &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 `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 4x4 matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Matrix4 &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 `_VECTORMATH_DEBUG` is defined.



---

# select

---

Conditionally select between two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 select(
            const Matrix4 &mat0,
            const Matrix4 &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 VMX 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 select(
            const Matrix4 &mat0,
            const Matrix4 &mat1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

<i>mat0</i>	4x4 matrix
<i>mat1</i>	4x4 matrix
<i>select1</i>	False selects the mat0 argument, true selects the mat1 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.

---

# transpose

---

Transpose of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Matrix4 transpose(
            const Matrix4 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 absPerElem(
            const Transform3 &tfrm
        );
    }
}
```

### 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 appendScale (
            const Transform3 &tfrm,
            Vector3 scaleVec
        );
    }
}
```

## Arguments

---

*tfrm*          3x4 transformation matrix  
*scaleVec*    3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# inverse

---

Inverse of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 inverse(
            const Transform3 &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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 mulPerElem(
            const Transform3 &tfrm0,
            const Transform3 &tfrm1
        );
    }
}
```

## Arguments

---

*tfrm0* 3x4 transformation matrix  
*tfrm1* 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.

---

# orthoInverse

---

Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 orthoInverse (
            const Transform3 &trfm
        );
    }
}
```

## Arguments

---

*trfm* 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 prependScale(
            Vector3 scaleVec,
            const Transform3 &tfrm
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# print

---

Print a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Transform3 &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 `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3x4 transformation matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline void print(
            const Transform3 &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 `_VECTORMATH_DEBUG` is defined.

---

# select

---

Conditionally select between two 3x4 transformation matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 select(
            const Transform3 &tfrm0,
            const Transform3 &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 VMX register may use more processing time than a branch. Use the *boolInVec* version for better performance.

---

# select

---

Conditionally select between two 3x4 transformation matrices (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        inline const Transform3 select(
            const Transform3 &tfrm0,
            const Transform3 &tfrm1,
            boolInVec select1
        );
    }
}
```

## Arguments

---

<i>tfrm0</i>	3x4 transformation matrix
<i>tfrm1</i>	3x4 transformation matrix
<i>select1</i>	False selects the tfrm0 argument, true selects the tfrm1 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.

---

# **Vectormath::Aos::Matrix3**

# Summary

## Vectormath::Aos::Matrix3

A 3x3 matrix in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.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 a 3x3 matrix.
<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 3-D vector.
<a href="#">operator *</a>	Multiply two 3x3 matrices.
<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 3x3 matrix.
<a href="#">operator +</a>	Add two 3x3 matrices.
<a href="#">operator +=</a>	Perform compound assignment and addition with a 3x3 matrix.
<a href="#">operator -</a>	Subtract a 3x3 matrix from another 3x3 matrix.
<a href="#">operator -</a>	Negate all elements of a 3x3 matrix.

Methods	Description
<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 3-D vector.
<a href="#"><u>rotation</u></a>	Construct a 3x3 matrix to rotate around a unit-length 3-D 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 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.



---

# Constructors and Destructors

## Matrix3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Matrix3

---

Copy a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

None

## Description

---

Construct a copy of a 3x3 matrix.

---

# Matrix3

---

Construct a 3x3 matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3(
                Vector3 col0,
                Vector3 col1,
                Vector3 col2
            );
        };
    };
}
```

## Arguments

---

*col0* 3-D vector  
*col1* 3-D vector  
*col2* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3x3 matrix containing the specified columns.

---

# Matrix3

---

Construct a 3x3 rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            explicit inline Matrix3(
                Quat 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.

---

# Matrix3

---

Set all elements of a 3x3 matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            explicit 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.

---

# Matrix3

---

Set all elements of a 3x3 matrix to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            explicit inline Matrix3(
                floatInVec 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator *(
                float scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 3x3 matrix and scalar

### Description

---

Multiply a 3x3 matrix by a scalar.

---

# operator \*

---

Multiply a 3x3 matrix by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator *(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* 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.



---

# operator \*

---

Multiply a 3x3 matrix by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 operator *(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Product of the specified 3x3 matrix and 3-D vector

## Description

---

Multiply a 3x3 matrix by a 3-D vector.

---

# operator \*

---

Multiply two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator *(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Product of the specified 3x3 matrices

## Description

---

Multiply two 3x3 matrices.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            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.

---

## operator \*=

---

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &operator *=(
                floatInVec scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* 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.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &operator *=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and multiplication by a 3x3 matrix.

---

# operator+

---

Add two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator+(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Sum of the specified 3x3 matrices

## Description

---

Add two 3x3 matrices.

---

# operator+=

---

Perform compound assignment and addition with a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &operator+=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and addition with a 3x3 matrix.

---

# operator-

---

Subtract a 3x3 matrix from another 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator-(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Difference of the specified 3x3 matrices

## Description

---

Subtract a 3x3 matrix from another 3x3 matrix.



---

# operator-

---

Negate all elements of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Matrix3 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3x3 matrix containing negated elements of the specified 3x3 matrix

## Description

---

Negate all elements of a 3x3 matrix.

---

# operator-=

---

Perform compound assignment and subtraction by a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &operator-=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and subtraction by a 3x3 matrix.

---

# operator=

---

Assign one 3x3 matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &operator=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Assign one 3x3 matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Vector3 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-2

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Matrix3](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 operator[] (
                int col
            );
        }
    }
}
```

## 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static 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 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotation(
                float radians,
                Vector3 unitVec
            );
        }
    }
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 3x3 matrix

## Description

---

Construct a 3x3 matrix to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a 3x3 matrix to rotate around a unit-length 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotation(
                floatInVec radians,
                Vector3 unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value (stored in vector data type)  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 3x3 matrix

## Description

---

Construct a 3x3 matrix to rotate around a unit-length 3-D vector by the specified radians angle.



---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotation(
                Quat 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.

---

# rotationX

---

Construct a 3x3 matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static 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.

---

# rotationX

---

Construct a 3x3 matrix to rotate around the x axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotationX(
                floatInVec 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 x axis by the specified radians angle.

---

# rotationY

---

Construct a 3x3 matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static 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.

---

# rotationY

---

Construct a 3x3 matrix to rotate around the y axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotationY(
                floatInVec 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 y axis by the specified radians angle.

---

# rotationZ

---

Construct a 3x3 matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static 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.

---

# rotationZ

---

Construct a 3x3 matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotationZ(
                floatInVec 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.

---

# rotationZYX

---

Construct a 3x3 matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 rotationZYX(
                Vector3 radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to *rotationZ(radiansXYZ.getZ()) \* rotationY(radiansXYZ.getY()) \* rotationX(radiansXYZ.getX())*.



---

# scale

---

Construct a 3x3 matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            static inline const Matrix3 scale(
                Vector3 scaleVec
            );
        };
    };
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# Public Instance Methods

## getCol

---

Get the column of a 3x3 matrix referred to by the specified index.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 getCol(
                int col
            );
        }
    }
}
```

### 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.

---

# getCol0

---

Get column 0 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 3x3 matrix.

---

# getCol1

---

Get column 1 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 3x3 matrix.

---

# getCol2

---

Get column 2 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 3x3 matrix.

---

# getElem

---

Get the element of a 3x3 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const floatInVec getElem(
                int col,
                int row
            );
        };
    };
}
```

## 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.

---

# getRow

---

Get the row of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline const Vector3 getRow(
                int row
            );
        }
    }
}
```

## 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.

---

# setCol

---

Set the column of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setCol(
                int col,
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-2
<i>vec</i>	3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set the column of a 3x3 matrix referred to by the specified index.



---

# setCol0

---

Set column 0 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setCol0(
                Vector3 col0
            );
        }
    }
}
```

## Arguments

---

*col0* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 0 of a 3x3 matrix.

---

# setCol1

---

Set column 1 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setCol1(
                Vector3 col1
            );
        }
    }
}
```

## Arguments

---

*col1* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 1 of a 3x3 matrix.

---

# setCol2

---

Set column 2 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setCol2(
                Vector3 col2
            );
        }
    }
}
```

## Arguments

---

*col2* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 2 of a 3x3 matrix.

---

# setElem

---

Set the element of a 3x3 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            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.

---

# 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setElem(
                int col,
                int row,
                floatInVec 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.

---

# setRow

---

Set the row of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix3 {
            inline Matrix3 &setRow(
                int row,
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-2
<i>vec</i>	3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set the row of a 3x3 matrix referred to by the specified index.

---

# **Vectormath::Aos::Matrix4**

# Summary

## Vectormath::Aos::Matrix4

A 4x4 matrix in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.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="#">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 a 4x4 matrix.
<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 3-D vector.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a unit-length quaternion and a 3-D 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 4-D vector.
<a href="#">operator *</a>	Multiply a 4x4 matrix by a 3-D vector.
<a href="#">operator *</a>	Multiply a 4x4 matrix by a 3-D point.
<a href="#">operator *</a>	Multiply two 4x4 matrices.
<a href="#">operator *</a>	Multiply a 4x4 matrix by a 3x4 transformation matrix.



Methods	Description
<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>	Perform compound assignment and multiplication by a 4x4 matrix.
<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>perspective</u></a>	Construct a perspective projection matrix.
<a href="#"><u>rotation</u></a>	Construct a 4x4 matrix to rotate around a unit-length 3-D vector.
<a href="#"><u>rotation</u></a>	Construct a 4x4 matrix to rotate around a unit-length 3-D 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.

---

# Constructors and Destructors

## Matrix4

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Matrix4

---

Copy a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4(
                const Matrix4 &mat
            );
        };
    };
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

None

## Description

---

Construct a copy of a 4x4 matrix.

---

# Matrix4

---

Construct a 4x4 matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4(
                Vector4 col0,
                Vector4 col1,
                Vector4 col2,
                Vector4 col3
            );
        };
    };
}
```

## Arguments

---

*col0* 4-D vector  
*col1* 4-D vector  
*col2* 4-D vector  
*col3* 4-D vector

## Return Values

---

None

## Description

---

Construct a 4x4 matrix containing the specified columns.

---

# Matrix4

---

Construct a 4x4 matrix from a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            explicit inline Matrix4(
                const Transform3 &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).

---

# Matrix4

---

Construct a 4x4 matrix from a 3x3 matrix and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4(
                const Matrix3 &mat,
                Vector3 translateVec
            );
        };
    };
}
```

## Arguments

---

<i>mat</i>	3x3 matrix
<i>translateVec</i>	3-D 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 3-D vector argument, and whose bottom row is (0,0,0,1).

---

# Matrix4

---

Construct a 4x4 matrix from a unit-length quaternion and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4(
                Quat unitQuat,
                Vector3 translateVec
            );
        };
    };
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>translateVec</i>	3-D 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 3-D vector argument, and whose bottom row is (0,0,0,1).

---

# Matrix4

---

Set all elements of a 4x4 matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            explicit 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.



---

# Matrix4

---

Set all elements of a 4x4 matrix to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            explicit inline Matrix4(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator *(
                float scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 4x4 matrix and scalar

### Description

---

Multiply a 4x4 matrix by a scalar.

---

# operator \*

---

Multiply a 4x4 matrix by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator *(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* 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.

---

# operator \*

---

Multiply a 4x4 matrix by a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 operator *(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Product of the specified 4x4 matrix and 4-D vector

## Description

---

Multiply a 4x4 matrix by a 4-D vector.

---

# operator \*

---

Multiply a 4x4 matrix by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 operator *(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Product of the specified 4x4 matrix and 3-D vector

## Description

---

Multiply a 4x4 matrix by a 3-D vector treated as if it were a 4-D vector with the w element equal to 0.

---

# operator \*

---

Multiply a 4x4 matrix by a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 operator *(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Product of the specified 4x4 matrix and 3-D point

## Description

---

Multiply a 4x4 matrix by a 3-D point treated as if it were a 4-D vector with the w element equal to 1.

---

# operator \*

---

Multiply two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator *(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Product of the specified 4x4 matrices

## Description

---

Multiply two 4x4 matrices.

---

# operator \*

---

Multiply a 4x4 matrix by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator *(
                const Transform3 &tfrm
            );
        }
    }
}
```

## 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).



---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            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.

---

## operator \*=

---

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator *=(
                floatInVec scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* 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.

---

# operator \*=

---

Perform compound assignment and multiplication by a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator *=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and multiplication by a 4x4 matrix.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator *=(
                const Transform3 &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.

---

# operator+

---

Add two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator+(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Sum of the specified 4x4 matrices

## Description

---

Add two 4x4 matrices.

---

# operator+=

---

Perform compound assignment and addition with a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator+=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and addition with a 4x4 matrix.

---

# operator-

---

Subtract a 4x4 matrix from another 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator-(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Difference of the specified 4x4 matrices

## Description

---

Subtract a 4x4 matrix from another 4x4 matrix.

---

# operator-

---

Negate all elements of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix4 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

4x4 matrix containing negated elements of the specified 4x4 matrix

## Description

---

Negate all elements of a 4x4 matrix.



---

# operator-=

---

Perform compound assignment and subtraction by a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator-=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and subtraction by a 4x4 matrix.

---

# operator=

---

Assign one 4x4 matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &operator=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Assign one 4x4 matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Vector4 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Matrix4](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

Indexed column

## Description

---

Subscripting operator invoked when applied to const [Matrix4](#).

## frustum

### Definition

## Arguments

## Return Values

### Description

$$\begin{array}{ccccccc} 2*zNear/(right-left) & 0 & (right+left)/(right-left) & 0 & & & \\ & 0 & 2*zNear/(top-bottom) & (top+bottom)/(top-bottom) & 0 & & \\ & 0 & 0 & -(zFar+zNear)/(zFar-zNear) & & & \\ -2*zFar*zNear/(zFar-zNear) & & & & & & \\ & 0 & 0 & -1 & 0 & & \end{array}$$

---

# identity

---

Construct an identity 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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.

---

# lookAt

---

Construct viewing matrix based on eye position, position looked at, and up direction.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 lookAt(
                Point3 eyePos,
                Point3 lookAtPos,
                Vector3 upVec
            );
        };
    };
}
```

## Arguments

---

<i>eyePos</i>	3-D point
<i>lookAtPos</i>	3-D point
<i>upVec</i>	3-D 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.

---

# orthographic

---

Construct an orthographic projection matrix.

## Definition

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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

$$\begin{array}{cccc} 2/(right-left) & 0 & 0 & -(right+left)/(right-left) \\ 0 & 2/(top-bottom) & 0 & -(top+bottom)/(top-bottom) \\ 0 & 0 & -2/(zFar-zNear) & -(zFar+zNear)/(zFar-zNear) \\ 0 & 0 & 0 & 1 \end{array}.$$



---

# perspective

---

Construct a perspective projection matrix.

## Definition

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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:

$$\begin{bmatrix} \cot(fovyRadians/2)/aspect & 0 & 0 & 0 \\ 0 & \cot(fovyRadians/2) & 0 & 0 \\ 0 & 0 & (zFar+zNear)/(zNear-zFar) & 2*zFar*zNear/(zNear-zFar) \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

---

# rotation

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotation(
                float radians,
                Vector3 unitVec
            );
        }
    }
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 4x4 matrix

## Description

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotation(
                floatInVec radians,
                Vector3 unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value (stored in vector data type)  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 4x4 matrix

## Description

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotation(
                Quat 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.

---

# rotationX

---

Construct a 4x4 matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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.

---

# rotationX

---

Construct a 4x4 matrix to rotate around the x axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotationX(
                floatInVec 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.

---

# rotationY

---

Construct a 4x4 matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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.

---

# rotationY

---

Construct a 4x4 matrix to rotate around the y axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotationY(
                floatInVec 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.



---

# rotationZ

---

Construct a 4x4 matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static 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.

---

# rotationZ

---

Construct a 4x4 matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotationZ(
                floatInVec 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.

---

# rotationZYX

---

Construct a 4x4 matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 rotationZYX(
                Vector3 radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to `rotationZ(radiansXYZ.getZ()) * rotationY(radiansXYZ.getY()) * rotationX(radiansXYZ.getX())`.

---

# scale

---

Construct a 4x4 matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 scale(
                Vector3 scaleVec
            );
        }
    }
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# translation

---

Construct a 4x4 matrix to perform translation.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            static inline const Matrix4 translation(
                Vector3 translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D 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*.

---

# Public Instance Methods

## getCol

---

Get the column of a 4x4 matrix referred to by the specified index.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getCol(
                int col
            );
        }
    }
}
```

### 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.

---

# getCol0

---

Get column 0 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 4x4 matrix.

---

# getCol1

---

Get column 1 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 4x4 matrix.



---

# getCol2

---

Get column 2 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 4x4 matrix.

---

# getCol3

---

Get column 3 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getCol3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 3

## Description

---

Get column 3 of a 4x4 matrix.

---

# getElem

---

Get the element of a 4x4 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const floatInVec getElem(
                int col,
                int row
            );
        };
    };
}
```

## 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.

---

# getRow

---

Get the row of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector4 getRow(
                int row
            );
        }
    }
}
```

## 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.

---

# getTranslation

---

Get the translation component of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Vector3 getTranslation();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Translation component

## Description

---

Get the translation component of a 4x4 matrix.

---

# getUpper3x3

---

Get the upper-left 3x3 submatrix of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline const Matrix3 getUpper3x3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Upper-left 3x3 submatrix

## Description

---

Get the upper-left 3x3 submatrix of a 4x4 matrix.

---

# setCol

---

Set the column of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setCol(
                int col,
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-3
<i>vec</i>	4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the column of a 4x4 matrix referred to by the specified index.

---

# setCol0

---

Set column 0 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setCol0(
                Vector4 col0
            );
        }
    }
}
```

## Arguments

---

*col0* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 0 of a 4x4 matrix.



---

# setCol1

---

Set column 1 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setCol1(
                Vector4 col1
            );
        }
    }
}
```

## Arguments

---

*col1* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 1 of a 4x4 matrix.

---

# setCol2

---

Set column 2 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setCol2(
                Vector4 col2
            );
        }
    }
}
```

## Arguments

---

*col2* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 2 of a 4x4 matrix.

---

# setCol3

---

Set column 3 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setCol3(
                Vector4 col3
            );
        }
    }
}
```

## Arguments

---

*col3* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 3 of a 4x4 matrix.

---

# setElem

---

Set the element of a 4x4 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            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.

---

# 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setElem(
                int col,
                int row,
                floatInVec 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.

---

# setRow

---

Set the row of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setRow(
                int row,
                Vector4 vec
            );
        };
    };
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-3
<i>vec</i>	4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the row of a 4x4 matrix referred to by the specified index.

---

# setTranslation

---

Set translation component.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setTranslation(
                Vector3 translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the translation component of a 4x4 matrix equal to the specified 3-D vector.

## Notes

---

This function does not change the bottom row elements.

---

# setUpper3x3

---

Set the upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Matrix4 {
            inline Matrix4 &setUpper3x3(
                const Matrix3 &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.



---

# **Vectormath::Aos::Point3**

# Summary

## Vectormath::Aos::Point3

A 3-D point in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.h>
class Point3;
```

### Description

A class representing a 3-D point stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a 3-D point.
<a href="#">getElem</a>	Get an x, y, or z element of a 3-D point by index.
<a href="#">getX</a>	Get the x element of a 3-D point.
<a href="#">getY</a>	Get the y element of a 3-D point.
<a href="#">getZ</a>	Get the z element of a 3-D point.
<a href="#">operator+</a>	Add a 3-D point to a 3-D vector.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3-D vector.
<a href="#">operator-</a>	Subtract a 3-D point from another 3-D point.
<a href="#">operator-</a>	Subtract a 3-D vector from a 3-D point.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3-D vector.
<a href="#">operator=</a>	Assign one 3-D 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="#">Point3</a>	Default constructor; does no initialization.
<a href="#">Point3</a>	Construct a 3-D point from x, y, and z elements.
<a href="#">Point3</a>	Construct a 3-D point from x, y, and z elements (scalar data contained in vector data type).
<a href="#">Point3</a>	Copy elements from a 3-D vector into a 3-D point.
<a href="#">Point3</a>	Set all elements of a 3-D point to the same scalar value.
<a href="#">Point3</a>	Set all elements of a 3-D point to the same scalar value (scalar data contained in vector data type).
<a href="#">Point3</a>	Set vector float data in a 3-D point.
<a href="#">setElem</a>	Set an x, y, or z element of a 3-D point by index.
<a href="#">setElem</a>	Set an x, y, or z element of a 3-D point by index (scalar data contained in vector data type).
<a href="#">setX</a>	Set the x element of a 3-D point.
<a href="#">setX</a>	Set the x element of a 3-D point (scalar data contained in vector data type).
<a href="#">setY</a>	Set the y element of a 3-D point.
<a href="#">setY</a>	Set the y element of a 3-D point (scalar data contained in vector data type).
<a href="#">setZ</a>	Set the z element of a 3-D point.

---

Methods	Description
<a href="#">setZ</a>	Set the z element of a 3-D point (scalar data contained in vector data type).

---

# Constructors and Destructors

## Point3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Point3

---

Construct a 3-D point from x, y, and z elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3(
                float x,
                float y,
                float z
            );
        };
    };
}
```

## Arguments

---

*x*    Scalar value  
*y*    Scalar value  
*z*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D point containing the specified x, y, and z elements.

---

# Point3

---

Construct a 3-D point from x, y, and z elements (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3(
                floatInVec x,
                floatInVec y,
                floatInVec 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 3-D point containing the specified x, y, and z elements.

---

# Point3

---

Copy elements from a 3-D vector into a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            explicit inline Point3(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3-D point containing the x, y, and z elements of the specified 3-D vector.

---

# Point3

---

Set all elements of a 3-D point to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            explicit inline Point3(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D point with all elements set to the scalar value argument.



---

# Point3

---

Set all elements of a 3-D point to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            explicit inline Point3(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

None

## Description

---

Construct a 3-D point with all elements set to the scalar value argument.

---

# Point3

---

Set vector float data in a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            explicit inline Point3(
                vec_float4 vf4
            );
        }
    }
}
```

## Arguments

---

*vf4*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D point whose internal vector float data is set to the vector float argument.

---

# Operator Methods

## operator+

---

Add a 3-D point to a 3-D vector.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const Point3 operator+(
                Vector3 vec
            );
        }
    }
}
```

### Arguments

---

*vec* 3-D vector

### Return Values

---

Sum of the specified 3-D point and 3-D vector

### Description

---

Add a 3-D point to a 3-D vector.

---

# operator+=

---

Perform compound assignment and addition with a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &operator+=(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Perform compound assignment and addition with a 3-D vector.

---

# operator-

---

Subtract a 3-D point from another 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const Vector3 operator-(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Difference of the specified 3-D points

## Description

---

Subtract a 3-D point from another 3-D point.

---

# operator-

---

Subtract a 3-D vector from a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const Point3 operator-(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Difference of the specified 3-D point and 3-D vector

## Description

---

Subtract a 3-D vector from a 3-D point.

---

# operator-=

---

Perform compound assignment and subtraction by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &operator-=(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Perform compound assignment and subtraction by a 3-D vector.

---

# operator=

---

Assign one 3-D point to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &operator=(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Assign one 3-D point to another.



---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const floatInVec operator[] (
                int idx
            );
        }
    }
}
```

## Arguments

---

*idx*     Index, expected in the range 0-2

## Return Values

---

Indexed element

## Description

---

Subscripting operator invoked when applied to const [Point3](#).

---

# Public Instance Methods

## get128

---

Get vector float data from a 3-D point.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline vec_float4 get128();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

Internal vector float data

### Description

---

Get internal vector float data from a 3-D point.

---

# getElem

---

Get an x, y, or z element of a 3-D point by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const floatInVec getElem(
                int idx
            );
        }
    }
}
```

## 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 3-D point by specifying an index of 0, 1, or 2, respectively.

---

# getX

---

Get the x element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const floatInVec getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 3-D point

## Description

---

Get the x element of a 3-D point.

---

# getY

---

Get the y element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const floatInVec getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 3-D point

## Description

---

Get the y element of a 3-D point.

---

# getZ

---

Get the z element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline const floatInVec getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 3-D point

## Description

---

Get the z element of a 3-D point.

---

# setElem

---

Set an x, y, or z element of a 3-D point by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setElem(
                int idx,
                float value
            );
        };
    };
}
```

## Arguments

---

*idx*      Index, expected in the range 0-2  
*value*    Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set an x, y, or z element of a 3-D point by specifying an index of 0, 1, or 2, respectively.



---

# setElem

---

Set an x, y, or z element of a 3-D point by index (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setElem(
                int idx,
                floatInVec 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 3-D point

## Description

---

Set an x, y, or z element of a 3-D point by specifying an index of 0, 1, or 2, respectively.

---

# setX

---

Set the x element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setX(
                float x
            );
        }
    }
}
```

## Arguments

---

$x$  Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the x element of a 3-D point to the specified scalar value.

---

# setX

---

Set the x element of a 3-D point (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setX(
                floatInVec x
            );
        }
    }
}
```

## Arguments

---

$x$       Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the x element of a 3-D point to the specified scalar value.

---

# setY

---

Set the y element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setY(
                float y
            );
        }
    }
}
```

## Arguments

---

*y* Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the y element of a 3-D point to the specified scalar value.

---

# setY

---

Set the y element of a 3-D point (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setY(
                floatInVec y
            );
        }
    }
}
```

## Arguments

---

*y*      Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the y element of a 3-D point to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setZ(
                float z
            );
        }
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the z element of a 3-D point to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D point (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Point3 {
            inline Point3 &setZ(
                floatInVec z
            );
        }
    }
}
```

## Arguments

---

*z*      Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the z element of a 3-D point to the specified scalar value.

---

# **Vectormath::Aos::Quat**



# Summary

## Vectormath::Aos::Quat

A quaternion in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.h>
class Quat;
```

### Description

A class representing a quaternion stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a quaternion.
<a href="#">getElem</a>	Get an x, y, z, or w element of a quaternion by index.
<a href="#">getW</a>	Get the w element of a quaternion.
<a href="#">getX</a>	Get the x element of a quaternion.
<a href="#">getXYZ</a>	Get the x, y, and z elements of a quaternion.
<a href="#">getY</a>	Get the y element of a quaternion.
<a href="#">getZ</a>	Get the z element of a quaternion.
<a href="#">identity</a>	Construct an identity quaternion.
<a href="#">operator *</a>	Multiply two quaternions.
<a href="#">operator *</a>	Multiply a quaternion by a scalar.
<a href="#">operator *</a>	Multiply a quaternion by a scalar (scalar data contained in vector data type).
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a quaternion.
<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 quaternions.
<a href="#">operator+=</a>	Perform compound assignment and addition with a quaternion.
<a href="#">operator-</a>	Subtract a quaternion from another quaternion.
<a href="#">operator-</a>	Negate all elements of a quaternion.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a quaternion.
<a href="#">operator/</a>	Divide a quaternion by a scalar.
<a href="#">operator/</a>	Divide a quaternion 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 quaternion 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="#">Quat</a>	Default constructor; does no initialization.

Methods	Description
<a href="#">Quat</a>	Construct a quaternion from x, y, z, and w elements.
<a href="#">Quat</a>	Construct a quaternion from x, y, z, and w elements (scalar data contained in vector data type).
<a href="#">Quat</a>	Construct a quaternion from a 3-D vector and a scalar.
<a href="#">Quat</a>	Construct a quaternion from a 3-D vector and a scalar (scalar data contained in vector data type).
<a href="#">Quat</a>	Copy elements from a 4-D vector into a quaternion.
<a href="#">Quat</a>	Convert a rotation matrix to a unit-length quaternion.
<a href="#">Quat</a>	Set all elements of a quaternion to the same scalar value.
<a href="#">Quat</a>	Set all elements of a quaternion to the same scalar value (scalar data contained in vector data type).
<a href="#">Quat</a>	Set vector float data in a quaternion.
<a href="#">rotation</a>	Construct a quaternion to rotate between two unit-length 3-D vectors.
<a href="#">rotation</a>	Construct a quaternion to rotate around a unit-length 3-D vector.
<a href="#">rotation</a>	Construct a quaternion to rotate around a unit-length 3-D vector (scalar data contained in vector data type).
<a href="#">rotationX</a>	Construct a quaternion to rotate around the x axis.
<a href="#">rotationX</a>	Construct a quaternion to rotate around the x axis (scalar data contained in vector data type).
<a href="#">rotationY</a>	Construct a quaternion to rotate around the y axis.
<a href="#">rotationY</a>	Construct a quaternion to rotate around the y axis (scalar data contained in vector data type).
<a href="#">rotationZ</a>	Construct a quaternion to rotate around the z axis.
<a href="#">rotationZ</a>	Construct a quaternion to rotate around the z axis (scalar data contained in vector data type).
<a href="#">setElem</a>	Set an x, y, z, or w element of a quaternion by index.
<a href="#">setElem</a>	Set an x, y, z, or w element of a quaternion by index (scalar data contained in vector data type).
<a href="#">setW</a>	Set the w element of a quaternion.
<a href="#">setW</a>	Set the w element of a quaternion (scalar data contained in vector data type).
<a href="#">setX</a>	Set the x element of a quaternion.
<a href="#">setX</a>	Set the x element of a quaternion (scalar data contained in vector data type).
<a href="#">setXYZ</a>	Set the x, y, and z elements of a quaternion.
<a href="#">setY</a>	Set the y element of a quaternion.
<a href="#">setY</a>	Set the y element of a quaternion (scalar data contained in vector data type).
<a href="#">setZ</a>	Set the z element of a quaternion.
<a href="#">setZ</a>	Set the z element of a quaternion (scalar data contained in vector data type).

---

# Constructors and Destructors

## Quat

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Quat

---

Construct a quaternion from x, y, z, and w elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# Quat

---

Construct a quaternion from x, y, z, and w elements (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat(
                floatInVec x,
                floatInVec y,
                floatInVec z,
                floatInVec 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.

---

# Quat

---

Construct a quaternion from a 3-D vector and a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat(
                Vector3 xyz,
                float w
            );
        };
    };
}
```

## Arguments

---

<i>xyz</i>	3-D vector
<i>w</i>	Scalar value

## Return Values

---

None

## Description

---

Construct a quaternion with the x, y, and z elements of the specified 3-D vector and with the w element set to the specified scalar.

---

# Quat

---

Construct a quaternion from a 3-D vector and a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat(
                Vector3 xyz,
                floatInVec w
            );
        }
    }
}
```

## Arguments

---

<i>xyz</i>	3-D 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 3-D vector and with the w element set to the specified scalar.

---

# Quat

---

Copy elements from a 4-D vector into a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            explicit inline Quat(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

None

## Description

---

Construct a quaternion containing the x, y, z, and w elements of the specified 4-D vector.



---

# Quat

---

Convert a rotation matrix to a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            explicit inline Quat(
                const Matrix3 &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.

---

# Quat

---

Set all elements of a quaternion to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            explicit inline Quat(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

None

## Description

---

Construct a quaternion with all elements set to the scalar value argument.

---

# Quat

---

Set all elements of a quaternion to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            explicit inline Quat(
                floatInVec 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.

---

# Quat

---

Set vector float data in a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            explicit inline Quat(
                vec_float4 vf4
            );
        }
    }
}
```

## Arguments

---

*vf4* Scalar value

## Return Values

---

None

## Description

---

Construct a quaternion whose internal vector float data is set to the vector float argument.

---

# Operator Methods

## operator \*

---

Multiply two quaternions.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator *(
                Quat quat
            );
        };
    }
}
```

### Arguments

---

*quat* Quaternion

### Return Values

---

Product of the specified quaternions

### Description

---

Multiply two quaternions.

---

# operator \*

---

Multiply a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator *(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

Product of the specified quaternion and scalar

## Description

---

Multiply a quaternion by a scalar.

---

# operator \*

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator *(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

Product of the specified quaternion and scalar

## Description

---

Multiply a quaternion by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator *=(
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and multiplication by a quaternion.



---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

## operator \*=

---

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator *=(
                floatInVec scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value (stored in vector data type)

### Return Values

---

A reference to the resulting quaternion

### Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator+

---

Add two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator+(
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Sum of the specified quaternions

## Description

---

Add two quaternions.

---

# operator+=

---

Perform compound assignment and addition with a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator+=(
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and addition with a quaternion.

---

# operator-

---

Subtract a quaternion from another quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator-(
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Difference of the specified quaternions

## Description

---

Subtract a quaternion from another quaternion.

---

# operator-

---

Negate all elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Quaternion containing negated elements of the specified quaternion

## Description

---

Negate all elements of a quaternion.

---

# operator-=

---

Perform compound assignment and subtraction by a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator-= (
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and subtraction by a quaternion.

---

# operator/

---

Divide a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator/(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

Quotient of the specified quaternion and scalar

## Description

---

Divide a quaternion by a scalar.



---

# operator/

---

Divide a quaternion by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Quat operator/(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

Quotient of the specified quaternion and scalar

## Description

---

Divide a quaternion by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# operator/=

---

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator/=(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one quaternion to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &operator=(
                Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Assign one quaternion to another.

---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec operator[](
                int idx
            );
        }
    }
}
```

## 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

## identity

---

Construct an identity quaternion.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat identity();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed quaternion

### Description

---

Construct an identity quaternion equal to (0,0,0,1).

---

# rotation

---

Construct a quaternion to rotate between two unit-length 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotation(
                Vector3 unitVec0,
                Vector3 unitVec1
            );
        };
    };
}
```

## Arguments

---

*unitVec0* 3-D vector, expected to be unit-length  
*unitVec1* 3-D vector, expected to be unit-length

## Return Values

---

The constructed quaternion

## Description

---

Construct a quaternion to rotate between two unit-length 3-D vectors.

## Notes

---

The result is unpredictable if *unitVec0* and *unitVec1* point in opposite directions.



---

# rotation

---

Construct a quaternion to rotate around a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotation(
                float radians,
                Vector3 unitVec
            );
        }
    }
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed quaternion

## Description

---

Construct a quaternion to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a quaternion to rotate around a unit-length 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotation(
                floatInVec radians,
                Vector3 unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value (stored in vector data type)  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed quaternion

## Description

---

Construct a quaternion to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotationX

---

Construct a quaternion to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static 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.

---

# rotationX

---

Construct a quaternion to rotate around the x axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotationX(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians*    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.

---

# rotationY

---

Construct a quaternion to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static 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.

---

# rotationY

---

Construct a quaternion to rotate around the y axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotationY(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians*    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.

---

# rotationZ

---

Construct a quaternion to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static 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.

---

# rotationZ

---

Construct a quaternion to rotate around the z axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            static inline const Quat rotationZ(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians*    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.



---

# Public Instance Methods

## get128

---

Get vector float data from a quaternion.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline vec_float4 get128();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

Internal vector float data

### Description

---

Get internal vector float data from a quaternion.

---

# getElem

---

Get an x, y, z, or w element of a quaternion by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec getElem(
                int idx
            );
        }
    }
}
```

## 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.

---

# getW

---

Get the w element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec getW();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

w element of a quaternion

## Description

---

Get the w element of a quaternion.

---

# getX

---

Get the x element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a quaternion

## Description

---

Get the x element of a quaternion.

---

# getXYZ

---

Get the x, y, and z elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const Vector3 getXYZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing x, y, and z elements

## Description

---

Extract a quaternion's x, y, and z elements into a 3-D vector.

---

# getY

---

Get the y element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a quaternion

## Description

---

Get the y element of a quaternion.

---

# getZ

---

Get the z element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline const floatInVec getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a quaternion

## Description

---

Get the z element of a quaternion.

---

# setElem

---

Set an x, y, z, or w element of a quaternion by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setElem(
                int idx,
                float value
            );
        }
    }
}
```

## Arguments

---

*idx*      Index, expected in the range 0-3  
*value*    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.



---

# setElem

---

Set an x, y, z, or w element of a quaternion by index (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setElem(
                int idx,
                floatInVec 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.

---

# setW

---

Set the  $w$  element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# setW

---

Set the w element of a quaternion (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setW(
                floatInVec 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.

---

# setX

---

Set the x element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# setX

---

Set the x element of a quaternion (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setX(
                floatInVec 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.

---

# setXYZ

---

Set the x, y, and z elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setXYZ(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting quaternion

## Description

---

Set the x, y, and z elements to those of the specified 3-D vector.

## Notes

---

This function does not change the w element.

---

# setY

---

Set the y element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# setY

---

Set the y element of a quaternion (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setY(
                floatInVec 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.



---

# setZ

---

Set the z element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            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.

---

# setZ

---

Set the z element of a quaternion (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Quat {
            inline Quat &setZ(
                floatInVec 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.

---

## **Vectormath::Aos::Transform3**

# Summary

## Vectormath::Aos::Transform3

A 3x4 transformation matrix in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.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 3-D vector.
<a href="#">operator *</a>	Multiply a 3x4 transformation matrix by a 3-D 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 3-D vector.
<a href="#">rotation</a>	Construct a 3x4 transformation matrix to rotate around a unit-length 3-D 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.

Methods	Description
<a href="#"><u>rotationY</u></a>	Construct a 3x4 transformation matrix to rotate around the y axis (scalar data contained in vector data type).
<a href="#"><u>rotationZ</u></a>	Construct a 3x4 transformation matrix to rotate around the z axis.
<a href="#"><u>rotationZ</u></a>	Construct a 3x4 transformation matrix to rotate around the z axis (scalar data contained in vector data type).
<a href="#"><u>rotationZYX</u></a>	Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.
<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 a 3x4 transformation matrix.
<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 3-D vector.
<a href="#"><u>Transform3</u></a>	Construct a 3x4 transformation matrix from a unit-length quaternion and a 3-D 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Transform3

---

Copy a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

None

## Description

---

Construct a copy of a 3x4 transformation matrix.

---

# Transform3

---

Construct a 3x4 transformation matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3(
                Vector3 col0,
                Vector3 col1,
                Vector3 col2,
                Vector3 col3
            );
        };
    };
}
```

## Arguments

---

*col0* 3-D vector  
*col1* 3-D vector  
*col2* 3-D vector  
*col3* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3x4 transformation matrix containing the specified columns.



---

# Transform3

---

Construct a 3x4 transformation matrix from a 3x3 matrix and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3(
                const Matrix3 &tfrm,
                Vector3 translateVec
            );
        };
    };
}
```

## Arguments

---

<i>tfrm</i>	3x3 matrix
<i>translateVec</i>	3-D 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 3-D vector argument.

---

# Transform3

---

Construct a 3x4 transformation matrix from a unit-length quaternion and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3(
                Quat unitQuat,
                Vector3 translateVec
            );
        };
    };
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>translateVec</i>	3-D 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 3-D vector argument.

---

# Transform3

---

Set all elements of a 3x4 transformation matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            explicit 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.

---

# Transform3

---

Set all elements of a 3x4 transformation matrix to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            explicit inline Transform3(
                floatInVec 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 3-D vector.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 operator *(
                Vector3 vec
            );
        }
    }
}
```

### Arguments

---

*vec* 3-D vector

### Return Values

---

Product of the specified 3x4 transformation matrix and 3-D vector

### Description

---

Applies the 3x3 upper-left submatrix (but not the translation component) of a 3x4 transformation matrix to a 3-D vector.

---

# operator \*

---

Multiply a 3x4 transformation matrix by a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Point3 operator *(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Product of the specified 3x4 transformation matrix and 3-D point

## Description

---

Applies the 3x3 upper-left submatrix and the translation component of a 3x4 transformation matrix to a 3-D point.

---

# operator \*

---

Multiply two 3x4 transformation matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Transform3 operator *(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

Product of the specified 3x4 transformation matrices

## Description

---

Multiply two 3x4 transformation matrices.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &operator *=(
                const Transform3 &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.



---

# operator=

---

Assign one 3x4 transformation matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &operator=(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Assign one 3x4 transformation matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Vector3 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Transform3](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 operator[] (
                int col
            );
        }
    }
}
```

## 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static 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 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotation(
                float radians,
                Vector3 unitVec
            );
        };
    };
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D 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 3-D vector by the specified radians angle.

---

# rotation

---

Construct a 3x4 transformation matrix to rotate around a unit-length 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotation(
                floatInVec radians,
                Vector3 unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value (stored in vector data type)  
*unitVec*    3-D 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 3-D vector by the specified radians angle.

---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotation(
                Quat 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.

---

# rotationX

---

Construct a 3x4 transformation matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationX(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians* 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.

---

# rotationY

---

Construct a 3x4 transformation matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationY(
                float radians
            );
        }
    }
}
```

## Arguments

---

*radians*    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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationY(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians* 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.

---

# rotationZ

---

Construct a 3x4 transformation matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationZ(
                float 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.

---

# rotationZ

---

Construct a 3x4 transformation matrix to rotate around the z axis (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationZ(
                floatInVec radians
            );
        }
    }
}
```

## Arguments

---

*radians* 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.

---

# rotationZYX

---

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 rotationZYX(
                Vector3 radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to *rotationZ(radiansXYZ.getZ()) \* rotationY(radiansXYZ.getY()) \* rotationX(radiansXYZ.getX())*.

---

# scale

---

Construct a 3x4 transformation matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 scale(
                Vector3 scaleVec
            );
        }
    }
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# translation

---

Construct a 3x4 transformation matrix to perform translation.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            static inline const Transform3 translation(
                Vector3 translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getCol(
                int col
            );
        }
    }
}
```

### 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.

---

# getCol0

---

Get column 0 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 3x4 transformation matrix.

---

# getCol1

---

Get column 1 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 3x4 transformation matrix.

---

# getCol2

---

Get column 2 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 3x4 transformation matrix.

---

# getCol3

---

Get column 3 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getCol3();
        }
    }
}
```

## 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const floatInVec getElem(
                int col,
                int row
            );
        };
    }
}
```

## 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.

---

# getRow

---

Get the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector4 getRow(
                int row
            );
        }
    }
}
```

## 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.

---

# getTranslation

---

Get the translation component of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Vector3 getTranslation();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Translation component

## Description

---

Get the translation component of a 3x4 transformation matrix.



---

# getUpper3x3

---

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline const Matrix3 getUpper3x3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Upper-left 3x3 submatrix

## Description

---

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

---

# setCol

---

Set the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setCol(
                int col,
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-3
<i>vec</i>	3-D 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.

---

# setCol0

---

Set column 0 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setCol0(
                Vector3 col0
            );
        }
    }
}
```

## Arguments

---

*col0* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 0 of a 3x4 transformation matrix.

---

# setCol1

---

Set column 1 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setCol1(
                Vector3 col1
            );
        }
    }
}
```

## Arguments

---

*col1* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 1 of a 3x4 transformation matrix.

---

# setCol2

---

Set column 2 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setCol2(
                Vector3 col2
            );
        }
    }
}
```

## Arguments

---

*col2* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 2 of a 3x4 transformation matrix.

---

# setCol3

---

Set column 3 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setCol3(
                Vector3 col3
            );
        }
    }
}
```

## Arguments

---

*col3* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 3 of a 3x4 transformation matrix.

---

# setElem

---

Set the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            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.

---

# 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setElem(
                int col,
                int row,
                floatInVec 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.



---

# setRow

---

Set the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setRow(
                int row,
                Vector4 vec
            );
        };
    };
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-2
<i>vec</i>	4-D 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.

---

# setTranslation

---

Set translation component.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setTranslation(
                Vector3 translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D 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 3-D vector.

---

# setUpper3x3

---

Set the upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Transform3 {
            inline Transform3 &setUpper3x3(
                const Matrix3 &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.

---

## **Vectormath::Aos::Vector3**

# Summary

## Vectormath::Aos::Vector3

A 3-D vector in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.h>
class Vector3;
```

### Description

A class representing a 3-D vector stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a 3-D vector.
<a href="#">getElem</a>	Get an x, y, or z element of a 3-D vector by index.
<a href="#">getX</a>	Get the x element of a 3-D vector.
<a href="#">getY</a>	Get the y element of a 3-D vector.
<a href="#">getZ</a>	Get the z element of a 3-D vector.
<a href="#">operator *</a>	Multiply a 3-D vector by a scalar.
<a href="#">operator *</a>	Multiply a 3-D 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 3-D vectors.
<a href="#">operator+</a>	Add a 3-D vector to a 3-D point.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3-D vector.
<a href="#">operator-</a>	Subtract a 3-D vector from another 3-D vector.
<a href="#">operator-</a>	Negate all elements of a 3-D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3-D vector.
<a href="#">operator/</a>	Divide a 3-D vector by a scalar.
<a href="#">operator/</a>	Divide a 3-D 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 3-D 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 x, y, or z element of a 3-D vector by index.
<a href="#">setElem</a>	Set an x, y, or z element of a 3-D vector by index (scalar data contained in vector data type).
<a href="#">setX</a>	Set the x element of a 3-D vector.
<a href="#">setX</a>	Set the x element of a 3-D vector (scalar data contained in vector data type).

---

Methods	Description
<a href="#"><u>setY</u></a>	Set the y element of a 3-D vector.
<a href="#"><u>setY</u></a>	Set the y element of a 3-D vector (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the z element of a 3-D vector.
<a href="#"><u>setZ</u></a>	Set the z element of a 3-D 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>	Construct a 3-D vector from x, y, and z elements.
<a href="#"><u>Vector3</u></a>	Construct a 3-D vector from x, y, and z elements (scalar data contained in vector data type).
<a href="#"><u>Vector3</u></a>	Copy elements from a 3-D point into a 3-D vector.
<a href="#"><u>Vector3</u></a>	Set all elements of a 3-D vector to the same scalar value.
<a href="#"><u>Vector3</u></a>	Set all elements of a 3-D 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 3-D vector.
<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.

---

# Constructors and Destructors

## Vector3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Vector3

---

Construct a 3-D vector from x, y, and z elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3(
                float x,
                float y,
                float z
            );
        };
    };
}
```

## Arguments

---

*x*    Scalar value  
*y*    Scalar value  
*z*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D vector containing the specified x, y, and z elements.



---

# Vector3

---

Construct a 3-D vector from x, y, and z elements (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3(
                floatInVec x,
                floatInVec y,
                floatInVec 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 3-D vector containing the specified x, y, and z elements.

---

# Vector3

---

Copy elements from a 3-D point into a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            explicit inline Vector3(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

Construct a 3-D vector containing the x, y, and z elements of the specified 3-D point.

---

# Vector3

---

Set all elements of a 3-D vector to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            explicit inline Vector3(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D vector with all elements set to the scalar value argument.

---

# Vector3

---

Set all elements of a 3-D vector to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            explicit inline Vector3(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

None

## Description

---

Construct a 3-D vector with all elements set to the scalar value argument.

---

# Vector3

---

Set vector float data in a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            explicit inline Vector3(
                vec_float4 vf4
            );
        }
    }
}
```

## Arguments

---

*vf4*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D vector whose internal vector float data is set to the vector float argument.

---

# Operator Methods

## operator \*

---

Multiply a 3-D vector by a scalar.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator *(
                float scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 3-D vector and scalar

### Description

---

Multiply a 3-D vector by a scalar.

---

# operator \*

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator *(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

Product of the specified 3-D vector and scalar

## Description

---

Multiply a 3-D vector by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator *=(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and multiplication by a scalar.



---

## operator \*=

---

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator *=(
                floatInVec scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value (stored in vector data type)

### Return Values

---

A reference to the resulting 3-D vector

### Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator+

---

Add two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator+(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Sum of the specified 3-D vectors

## Description

---

Add two 3-D vectors.

---

# operator+

---

Add a 3-D vector to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Point3 operator+(
                Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Sum of the specified 3-D vector and 3-D point

## Description

---

Add a 3-D vector to a 3-D point.

---

# operator+=

---

Perform compound assignment and addition with a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator+=(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and addition with a 3-D vector.

---

# operator-

---

Subtract a 3-D vector from another 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator-(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Difference of the specified 3-D vectors

## Description

---

Subtract a 3-D vector from another 3-D vector.

---

# operator-

---

Negate all elements of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing negated elements of the specified 3-D vector

## Description

---

Negate all elements of a 3-D vector.

---

# operator-=

---

Perform compound assignment and subtraction by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator-= (
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and subtraction by a 3-D vector.

---

# operator/

---

Divide a 3-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator/(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

Quotient of the specified 3-D vector and scalar

## Description

---

Divide a 3-D vector by a scalar.



---

# operator/

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const Vector3 operator/(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

Quotient of the specified 3-D vector and scalar

## Description

---

Divide a 3-D vector by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator/=(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and division by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator/=(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one 3-D vector to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &operator=(
                Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Assign one 3-D vector to another.

---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const floatInVec operator[] (
                int idx
            );
        }
    }
}
```

## 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            static inline const Vector3 xAxis();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed 3-D vector

### Description

---

Construct a 3-D vector equal to (1,0,0).

---

# yAxis

---

Construct y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            static inline const Vector3 yAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 3-D vector

## Description

---

Construct a 3-D vector equal to (0,1,0).



---

# zAxis

---

Construct z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            static inline const Vector3 zAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 3-D vector

## Description

---

Construct a 3-D vector equal to (0,0,1).

---

# Public Instance Methods

## get128

---

Get vector float data from a 3-D vector.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline vec_float4 get128();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

Internal vector float data

### Description

---

Get internal vector float data from a 3-D vector.

---

# getElem

---

Get an x, y, or z element of a 3-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const floatInVec getElem(
                int idx
            );
        }
    }
}
```

## 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 3-D vector by specifying an index of 0, 1, or 2, respectively.

---

# getX

---

Get the x element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const floatInVec getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 3-D vector

## Description

---

Get the x element of a 3-D vector.

---

# getY

---

Get the y element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const floatInVec getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 3-D vector

## Description

---

Get the y element of a 3-D vector.

---

# getZ

---

Get the z element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline const floatInVec getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 3-D vector

## Description

---

Get the z element of a 3-D vector.

---

# setElem

---

Set an x, y, or z element of a 3-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setElem(
                int idx,
                float value
            );
        };
    };
}
```

## Arguments

---

*idx*      Index, expected in the range 0-2  
*value*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set an x, y, or z element of a 3-D vector by specifying an index of 0, 1, or 2, respectively.

---

# setElem

---

Set an x, y, or z element of a 3-D vector by index (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setElem(
                int idx,
                floatInVec 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 3-D vector

## Description

---

Set an x, y, or z element of a 3-D vector by specifying an index of 0, 1, or 2, respectively.



---

# setX

---

Set the x element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setX(
                float x
            );
        };
    }
}
```

## Arguments

---

x    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the x element of a 3-D vector to the specified scalar value.

---

# setX

---

Set the x element of a 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setX(
                floatInVec x
            );
        }
    }
}
```

## Arguments

---

$x$       Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the x element of a 3-D vector to the specified scalar value.

---

# setY

---

Set the y element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setY(
                float y
            );
        }
    }
}
```

## Arguments

---

*y*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the y element of a 3-D vector to the specified scalar value.

---

# setY

---

Set the y element of a 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setY(
                floatInVec y
            );
        }
    }
}
```

## Arguments

---

*y*      Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the y element of a 3-D vector to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setZ(
                float z
            );
        }
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the z element of a 3-D vector to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector3 {
            inline Vector3 &setZ(
                floatInVec z
            );
        }
    }
}
```

## Arguments

---

*z*      Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the z element of a 3-D vector to the specified scalar value.

---

# **Vectormath::Aos::Vector4**

# Summary

## Vectormath::Aos::Vector4

A 4-D vector in array-of-structures format.

### Definition

```
#include <vectormath/cpp/vectormath_aos.h>
class Vector4;
```

### Description

A class representing a 4-D vector stored in array-of-structures (AoS) format.

### Methods Summary

Methods	Description
<a href="#">get128</a>	Get vector float data from a 4-D vector.
<a href="#">getElem</a>	Get an x, y, z, or w element of a 4-D vector by index.
<a href="#">getW</a>	Get the w element of a 4-D vector.
<a href="#">getX</a>	Get the x element of a 4-D vector.
<a href="#">getXYZ</a>	Get the x, y, and z elements of a 4-D vector.
<a href="#">getY</a>	Get the y element of a 4-D vector.
<a href="#">getZ</a>	Get the z element of a 4-D vector.
<a href="#">operator *</a>	Multiply a 4-D vector by a scalar.
<a href="#">operator *</a>	Multiply a 4-D 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 4-D vectors.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 4-D vector.
<a href="#">operator-</a>	Subtract a 4-D vector from another 4-D vector.
<a href="#">operator-</a>	Negate all elements of a 4-D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 4-D vector.
<a href="#">operator/</a>	Divide a 4-D vector by a scalar.
<a href="#">operator/</a>	Divide a 4-D 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 4-D 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 x, y, z, or w element of a 4-D vector by index.
<a href="#">setElem</a>	Set an x, y, z, or w element of a 4-D vector by index (scalar data contained in vector data type).
<a href="#">setW</a>	Set the w element of a 4-D vector.



Methods	Description
<a href="#"><u>setW</u></a>	Set the w element of a 4-D vector (scalar data contained in vector data type).
<a href="#"><u>setX</u></a>	Set the x element of a 4-D vector.
<a href="#"><u>setX</u></a>	Set the x element of a 4-D vector (scalar data contained in vector data type).
<a href="#"><u>setXYZ</u></a>	Set the x, y, and z elements of a 4-D vector.
<a href="#"><u>setY</u></a>	Set the y element of a 4-D vector.
<a href="#"><u>setY</u></a>	Set the y element of a 4-D vector (scalar data contained in vector data type).
<a href="#"><u>setZ</u></a>	Set the z element of a 4-D vector.
<a href="#"><u>setZ</u></a>	Set the z element of a 4-D 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>	Construct a 4-D vector from x, y, z, and w elements.
<a href="#"><u>Vector4</u></a>	Construct a 4-D vector from x, y, z, and w elements (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Construct a 4-D vector from a 3-D vector and a scalar.
<a href="#"><u>Vector4</u></a>	Construct a 4-D vector from a 3-D vector and a scalar (scalar data contained in vector data type).
<a href="#"><u>Vector4</u></a>	Copy x, y, and z from a 3-D vector into a 4-D vector, and set w to 0.
<a href="#"><u>Vector4</u></a>	Copy x, y, and z from a 3-D point into a 4-D vector, and set w to 1.
<a href="#"><u>Vector4</u></a>	Copy elements from a quaternion into a 4-D vector.
<a href="#"><u>Vector4</u></a>	Set all elements of a 4-D vector to the same scalar value.
<a href="#"><u>Vector4</u></a>	Set all elements of a 4-D 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 4-D vector.
<a href="#"><u>wAxis</u></a>	Construct w axis.
<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.

---

# Constructors and Destructors

## Vector4

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4() ;
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Vector4

---

Construct a 4-D vector from x, y, z, and w elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            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 4-D vector containing the specified x, y, z, and w elements.

---

# Vector4

---

Construct a 4-D vector from x, y, z, and w elements (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4(
                floatInVec x,
                floatInVec y,
                floatInVec z,
                floatInVec w
            );
        };
    };
}
```

## 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)
<i>w</i>	Scalar value (stored in vector data type)

## Return Values

---

None

## Description

---

Construct a 4-D vector containing the specified x, y, z, and w elements.

---

# Vector4

---

Construct a 4-D vector from a 3-D vector and a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4(
                Vector3 xyz,
                float w
            );
        };
    };
}
```

## Arguments

---

<i>xyz</i>	3-D vector
<i>w</i>	Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D vector and with the w element set to the specified scalar.

---

# Vector4

---

Construct a 4-D vector from a 3-D vector and a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4(
                Vector3 xyz,
                floatInVec w
            );
        };
    }
}
```

## Arguments

---

<i>xyz</i>	3-D vector
<i>w</i>	Scalar value (stored in vector data type)

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D vector and with the w element set to the specified scalar.

---

# Vector4

---

Copy x, y, and z from a 3-D vector into a 4-D vector, and set w to 0.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                Vector3 vec
            );
        };
    };
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D vector and with the w element set to 0.

---

# Vector4

---

Copy x, y, and z from a 3-D point into a 4-D vector, and set w to 1.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                Point3 pnt
            );
        };
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D point and with the w element set to 1.



---

# Vector4

---

Copy elements from a quaternion into a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                Quat quat
            );
        };
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

None

## Description

---

Construct a 4-D vector containing the x, y, z, and w elements of the specified quaternion.

---

# Vector4

---

Set all elements of a 4-D vector to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector with all elements set to the scalar value argument.

---

# Vector4

---

Set all elements of a 4-D vector to the same scalar value (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

None

## Description

---

Construct a 4-D vector with all elements set to the scalar value argument.

---

# Vector4

---

Set vector float data in a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            explicit inline Vector4(
                vec_float4 vf4
            );
        }
    }
}
```

## Arguments

---

*vf4*    Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector whose internal vector float data is set to the vector float argument.

---

# Operator Methods

## operator \*

---

Multiply a 4-D vector by a scalar.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator *(
                float scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 4-D vector and scalar

### Description

---

Multiply a 4-D vector by a scalar.

---

# operator \*

---

Multiply a 4-D vector by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator *(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

Product of the specified 4-D vector and scalar

## Description

---

Multiply a 4-D vector by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator *=(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and multiplication by a scalar.

---

## operator \*=

---

Perform compound assignment and multiplication by a scalar (scalar data contained in vector data type).

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator *=(
                floatInVec scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value (stored in vector data type)

### Return Values

---

A reference to the resulting 4-D vector

### Description

---

Perform compound assignment and multiplication by a scalar.



---

# operator+

---

Add two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator+(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Sum of the specified 4-D vectors

## Description

---

Add two 4-D vectors.

---

# operator+=

---

Perform compound assignment and addition with a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator+=(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and addition with a 4-D vector.

---

# operator-

---

Subtract a 4-D vector from another 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator-(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Difference of the specified 4-D vectors

## Description

---

Subtract a 4-D vector from another 4-D vector.

---

# operator-

---

Negate all elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

4-D vector containing negated elements of the specified 4-D vector

## Description

---

Negate all elements of a 4-D vector.

---

# operator-=

---

Perform compound assignment and subtraction by a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator-= (
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and subtraction by a 4-D vector.

---

# operator/

---

Divide a 4-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator/(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

Quotient of the specified 4-D vector and scalar

## Description

---

Divide a 4-D vector by a scalar.

---

# operator/

---

Divide a 4-D vector by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector4 operator/(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value (stored in vector data type)

## Return Values

---

Quotient of the specified 4-D vector and scalar

## Description

---

Divide a 4-D vector by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator/=(
                float scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and division by a scalar.



---

# operator/=

---

Perform compound assignment and division by a scalar (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator/=(
                floatInVec scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one 4-D vector to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &operator=(
                Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Assign one 4-D vector to another.

---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec operator[] (
                int idx
            );
        }
    }
}
```

## 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/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            static inline const Vector4 wAxis();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed 4-D vector

### Description

---

Construct a 4-D vector equal to (0,0,0,1).

---

# xAxis

---

Construct x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            static inline const Vector4 xAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (1,0,0,0).

---

# yAxis

---

Construct y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            static inline const Vector4 yAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (0,1,0,0).

---

# zAxis

---

Construct z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            static inline const Vector4 zAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (0,0,1,0).



---

# Public Instance Methods

## get128

---

Get vector float data from a 4-D vector.

### Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline vec_float4 get128();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

Internal vector float data

### Description

---

Get internal vector float data from a 4-D vector.

---

# getElem

---

Get an x, y, z, or w element of a 4-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec getElem(
                int idx
            );
        }
    }
}
```

## 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 4-D vector by specifying an index of 0, 1, 2, or 3, respectively.

---

# getW

---

Get the w element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec getW();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

w element of a 4-D vector

## Description

---

Get the w element of a 4-D vector.

---

# getX

---

Get the x element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 4-D vector

## Description

---

Get the x element of a 4-D vector.

---

# getXYZ

---

Get the x, y, and z elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const Vector3 getXYZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing x, y, and z elements

## Description

---

Extract a 4-D vector's x, y, and z elements into a 3-D vector.

---

# getY

---

Get the y element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 4-D vector

## Description

---

Get the y element of a 4-D vector.

---

# getZ

---

Get the z element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline const floatInVec getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 4-D vector

## Description

---

Get the z element of a 4-D vector.

---

# setElem

---

Set an x, y, z, or w element of a 4-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setElem(
                int idx,
                float value
            );
        };
    };
}
```

## Arguments

---

*idx*      Index, expected in the range 0-3  
*value*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set an x, y, z, or w element of a 4-D vector by specifying an index of 0, 1, 2, or 3, respectively.



---

# setElem

---

Set an x, y, z, or w element of a 4-D vector by index (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setElem(
                int idx,
                floatInVec 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 4-D vector

## Description

---

Set an x, y, z, or w element of a 4-D vector by specifying an index of 0, 1, 2, or 3, respectively.

---

# setW

---

Set the w element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setW(
                float w
            );
        }
    }
}
```

## Arguments

---

*w*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the w element of a 4-D vector to the specified scalar value.

---

# setW

---

Set the  $w$  element of a 4-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setW(
                floatInVec w
            );
        }
    }
}
```

## Arguments

---

$w$       Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the  $w$  element of a 4-D vector to the specified scalar value.

---

# setX

---

Set the x element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setX(
                float x
            );
        };
    }
}
```

## Arguments

---

x    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the x element of a 4-D vector to the specified scalar value.

---

# setX

---

Set the x element of a 4-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setX(
                floatInVec x
            );
        }
    }
}
```

## Arguments

---

$x$       Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the x element of a 4-D vector to the specified scalar value.

---

# setXYZ

---

Set the x, y, and z elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setXYZ(
                Vector3 vec
            );
        };
    };
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the x, y, and z elements to those of the specified 3-D vector.

## Notes

---

This function does not change the w element.

---

# setY

---

Set the y element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setY(
                float y
            );
        }
    }
}
```

## Arguments

---

*y* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the y element of a 4-D vector to the specified scalar value.

---

# setY

---

Set the y element of a 4-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setY(
                floatInVec y
            );
        }
    }
}
```

## Arguments

---

*y*      Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the y element of a 4-D vector to the specified scalar value.



---

# setZ

---

Set the z element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setZ(
                float z
            );
        };
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the z element of a 4-D vector to the specified scalar value.

---

# setZ

---

Set the z element of a 4-D vector (scalar data contained in vector data type).

## Definition

---

```
#include <vectormath/cpp/vectormath_aos.h>
namespace Vectormath {
    namespace Aos {
        class Vector4 {
            inline Vector4 &setZ(
                floatInVec z
            );
        };
    };
}
```

## Arguments

---

*z*          Scalar value (stored in vector data type)

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the z element of a 4-D vector to the specified scalar value.

---

# **Vectormath::Soa**

# Summary

## Vectormath::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 3-D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 4-D vector per element.
<a href="#">absPerElem</a>	Compute the absolute value of a 3-D 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="#">appendScale</a>	Append (post-multiply) a scale transformation to a 3x3 matrix.
<a href="#">appendScale</a>	Append (post-multiply) a scale transformation to a 4x4 matrix.
<a href="#">appendScale</a>	Append (post-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#">conj</a>	Compute the conjugate of a quaternion.
<a href="#">copySignPerElem</a>	Copy sign from one 3-D vector to another, per element.
<a href="#">copySignPerElem</a>	Copy sign from one 4-D vector to another, per element.
<a href="#">copySignPerElem</a>	Copy sign from one 3-D point to another, per element.
<a href="#">cross</a>	Compute cross product of two 3-D vectors.
<a href="#">crossMatrix</a>	Cross-product matrix of a 3-D vector.
<a href="#">crossMatrixMul</a>	Create cross-product matrix and multiply.
<a href="#">determinant</a>	Determinant of a 3x3 matrix.
<a href="#">determinant</a>	Determinant of a 4x4 matrix.
<a href="#">dist</a>	Compute the distance between two 3-D points.
<a href="#">distFromOrigin</a>	Compute the distance of a 3-D point from the coordinate-system origin.
<a href="#">distSqr</a>	Compute the square of the distance between two 3-D points.
<a href="#">distSqrFromOrigin</a>	Compute the square of the distance of a 3-D point from the coordinate-system origin.
<a href="#">divPerElem</a>	Divide two 3-D vectors per element.
<a href="#">divPerElem</a>	Divide two 4-D vectors per element.
<a href="#">divPerElem</a>	Divide two 3-D points per element.
<a href="#">dot</a>	Compute the dot product of two 3-D vectors.
<a href="#">dot</a>	Compute the dot product of two 4-D vectors.

Function	Description
<a href="#"><u>dot</u></a>	Compute the dot product of two quaternions.
<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 3-D vector.
<a href="#"><u>length</u></a>	Compute the length of a 4-D 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 3-D vector.
<a href="#"><u>lengthSqr</u></a>	Compute the square of the length of a 4-D vector.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 4-D vectors.
<a href="#"><u>lerp</u></a>	Linear interpolation between two 3-D points.
<a href="#"><u>lerp</u></a>	Linear interpolation between two quaternions.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3-D vectors, stored in three quadwords.
<a href="#"><u>loadXYZArray</u></a>	Load four three-float 3-D points, stored in three quadwords.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3-D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 4-D vector.
<a href="#"><u>maxElem</u></a>	Maximum element of a 3-D point.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3-D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 4-D vectors per element.
<a href="#"><u>maxPerElem</u></a>	Maximum of two 3-D points per element.
<a href="#"><u>minElem</u></a>	Minimum element of a 3-D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 4-D vector.
<a href="#"><u>minElem</u></a>	Minimum element of a 3-D point.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3-D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 4-D vectors per element.
<a href="#"><u>minPerElem</u></a>	Minimum of two 3-D points per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3-D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 4-D vectors per element.
<a href="#"><u>mulPerElem</u></a>	Multiply two 3-D 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 3-D vector.
<a href="#"><u>normalize</u></a>	Normalize a 4-D vector.
<a href="#"><u>normalize</u></a>	Normalize a quaternion.
<a href="#"><u>operator *</u></a>	Multiply a 3-D vector by a scalar.
<a href="#"><u>operator *</u></a>	Multiply a 4-D vector by a scalar.
<a href="#"><u>operator *</u></a>	Multiply a quaternion 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 3-D vectors.
<a href="#"><u>outer</u></a>	Outer product of two 4-D vectors.
<a href="#"><u>prependScale</u></a>	Prepend (pre-multiply) a scale transformation to a 3x3 matrix.

Function	Description
<a href="#">prependScale</a>	Prepend (pre-multiply) a scale transformation to a 4x4 matrix.
<a href="#">prependScale</a>	Prepend (pre-multiply) a scale transformation to a 3x4 transformation matrix.
<a href="#">print</a>	Print a 3-D vector.
<a href="#">print</a>	Print a 3-D vector and an associated string identifier.
<a href="#">print</a>	Print a 4-D vector.
<a href="#">print</a>	Print a 4-D vector and an associated string identifier.
<a href="#">print</a>	Print a 3-D point.
<a href="#">print</a>	Print a 3-D point and an associated string identifier.
<a href="#">print</a>	Print a quaternion.
<a href="#">print</a>	Print a quaternion and an associated string identifier.
<a href="#">print</a>	Print a 3x3 matrix.
<a href="#">print</a>	Print a 3x3 matrix and an associated string identifier.
<a href="#">print</a>	Print a 4x4 matrix.
<a href="#">print</a>	Print a 4x4 matrix and an associated string identifier.
<a href="#">print</a>	Print a 3x4 transformation matrix.
<a href="#">print</a>	Print a 3x4 transformation matrix and an associated string identifier.
<a href="#">projection</a>	Scalar projection of a 3-D point on a unit-length 3-D vector.
<a href="#">recipPerElem</a>	Compute the reciprocal of a 3-D vector per element.
<a href="#">recipPerElem</a>	Compute the reciprocal of a 4-D vector per element.
<a href="#">recipPerElem</a>	Compute the reciprocal of a 3-D point per element.
<a href="#">rotate</a>	Use a unit-length quaternion to rotate a 3-D 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 3-D vector per element.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 4-D vector per element.
<a href="#">rsqrtPerElem</a>	Compute the reciprocal square root of a 3-D point per element.
<a href="#">scale</a>	Apply uniform scale to a 3-D point.
<a href="#">scale</a>	Apply non-uniform scale to a 3-D point.
<a href="#">select</a>	Conditionally select between two 3-D vectors.
<a href="#">select</a>	Conditionally select between two 4-D vectors.
<a href="#">select</a>	Conditionally select between two 3-D points.
<a href="#">select</a>	Conditionally select between two quaternions.
<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 3-D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two 4-D vectors.
<a href="#">slerp</a>	Spherical linear interpolation between two quaternions.
<a href="#">sqrtPerElem</a>	Compute the square root of a 3-D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 4-D vector per element.
<a href="#">sqrtPerElem</a>	Compute the square root of a 3-D point per element.
<a href="#">squad</a>	Spherical quadrangle interpolation.
<a href="#">storeHalfFloats</a>	Store eight slots of two SoA 3-D vectors as half-floats.
<a href="#">storeHalfFloats</a>	Store four slots of an SoA 4-D vector as half-floats.
<a href="#">storeHalfFloats</a>	Store eight slots of two SoA 3-D points as half-floats.
<a href="#">storeXYZArray</a>	Store four slots of an SoA 3-D vector in three quadwords.
<a href="#">storeXYZArray</a>	Store four slots of an SoA 3-D point in three quadwords.

---

Function	Description
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 3-D vector.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 4-D vector.
<a href="#"><u>sum</u></a>	Compute the sum of all elements of a 3-D point.
<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>Vectormath::Soa::Matrix3</u></a>	A set of four 3x3 matrices in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Matrix4</u></a>	A set of four 4x4 matrices in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Point3</u></a>	A set of four 3-D points in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Quat</u></a>	A set of four quaternions in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Transform3</u></a>	A set of four 3x4 transformation matrices in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Vector3</u></a>	A set of four 3-D vectors in structure-of-arrays format.
<a href="#"><u>Vectormath::Soa::Vector4</u></a>	A set of four 4-D vectors in structure-of-arrays format.

---

# 3-D Vector Functions

## absPerElem

---

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

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 absPerElem(
            const Vector3 &vec
        );
    }
}
```

### Arguments

---

*vec* 3-D vector

### Return Values

---

3-D 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 3-D vector.



---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 copySignPerElem(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

3-D 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*.

---

# cross

---

Compute cross product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 cross(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

Cross product of the specified 3-D vectors

## Description

---

Compute cross product of two 3-D vectors.

---

# crossMatrix

---

Cross-product matrix of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 crossMatrix(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Cross-product matrix of *vec*

## Description

---

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

---

# crossMatrixMul

---

Create cross-product matrix and multiply.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 crossMatrixMul(
            const Vector3 &vec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*vec* 3-D vector  
*mat* 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 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 divPerElem(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Divide two 3-D vectors element by element.

## Notes

---

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

---

# dot

---

Compute the dot product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 dot(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

Dot product of the specified 3-D vectors

## Description

---

Compute the dot product of two 3-D vectors.

---

# length

---

Compute the length of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 length(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Length of the specified 3-D vector

## Description

---

Compute the length of a 3-D vector.

---

# lengthSqr

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 lengthSqr(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Square of the length of the specified 3-D vector

## Description

---

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



---

# lerp

---

Linear interpolation between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 lerp(
            vec_float4 t,
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

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

## Return Values

---

Interpolated 3-D vector

## Description

---

Linearly interpolate between two 3-D vectors.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# loadXYZArray

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void loadXYZArray(
            Vector3 &vec,
            const vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>vec</i>	An output 3-D vector
<i>threeQuads</i>	Array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

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

---

# maxElem

---

Maximum element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 maxElem(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Maximum value of all elements of *vec*

## Description

---

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

---

# maxPerElem

---

Maximum of two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 maxPerElem(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

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

---

# minElem

---

Minimum element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 minElem(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Minimum value of all elements of *vec*

## Description

---

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

---

# minPerElem

---

Minimum of two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 minPerElem(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

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

---

# mulPerElem

---

Multiply two 3-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 mulPerElem(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Multiply two 3-D vectors element by element.

---

# normalize

---

Normalize a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 normalize(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

The specified 3-D vector scaled to unit length

## Description

---

Compute a normalized 3-D vector.

## Notes

---

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



---

# operator \*

---

Multiply a 3-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 operator *(
            vec_float4 scalar,
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value  
*vec*        3-D vector

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 3-D vector by a scalar.

---

# outer

---

Outer product of two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 outer(
            const Vector3 &vec0,
            const Vector3 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 3-D vector  
*vec1* 3-D vector

## Return Values

---

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

## Description

---

Compute the outer product of two 3-D vectors.

---

# print

---

Print a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3-D vector and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Vector3 &vec,
            const char *name
        );
    }
}
```

## Arguments

---

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

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# recipPerElem

---

Compute the reciprocal of a 3-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 recipPerElem(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# rowMul

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 rowMul (
            const Vector3 &vec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*vec* 3-D vector  
*mat* 3x3 matrix

## Return Values

---

Product of a row-vector and a 3x3 matrix

## Description

---

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

---

# rsqrtPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 rsqrtPerElem(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# select

---

Conditionally select between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 select(
            const Vector3 &vec0,
            const Vector3 &vec1,
            vec_uint4 select1
        );
    }
}
```

## Arguments

---

<i>vec0</i>	3-D vector
<i>vec1</i>	3-D vector
<i>select1</i>	For each of the four word slots, this mask selects either the 3-D vector in the corresponding slot of <i>vec0</i> or the 3-D 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 3-D 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 3-D vectors at each of the corresponding slots of *vec0* or *vec1*.

## Notes

---

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



---

# slerp

---

Spherical linear interpolation between two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 slerp(
            vec_float4 t,
            const Vector3 &unitVec0,
            const Vector3 &unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	3-D vector, expected to be unit-length
<i>unitVec1</i>	3-D vector, expected to be unit-length

## Return Values

---

Interpolated 3-D vector

## Description

---

Perform spherical linear interpolation between two 3-D 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 3-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 sqrtPerElem(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

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

## Description

---

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

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store eight slots of two SoA 3-D vectors as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void storeHalfFloats(
            const Vector3 &vec0,
            const Vector3 &vec1,
            vec_ushort8 *threeQuads
        );
    }
}
```

## Arguments

---

<i>vec0</i>	3-D vector
<i>vec1</i>	3-D vector
<i>threeQuads</i>	An output array of 3 quadwords containing 24 half-floats

## Return Values

---

None

## Description

---

Store eight slots of two SoA 3-D vectors in three quadwords of half-float values. Numbering slots of *vec0* as 0..3 and slots of *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}.

---

# storeXYZArray

---

Store four slots of an SoA 3-D vector in three quadwords.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void storeXYZArray(
            const Vector3 &vec,
            vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>vec</i>	3-D vector
<i>threeQuads</i>	An output array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

Store four slots of an SoA 3-D vector in three quadwords as {x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3}.

---

# sum

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 sum(
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Sum of all elements of *vec*

## Description

---

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

---

# 4-D Vector Functions

## absPerElem

---

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

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 absPerElem(
            const Vector4 &vec
        );
    }
}
```

### Arguments

---

*vec* 4-D vector

### Return Values

---

4-D 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 4-D vector.

---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 copySignPerElem(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

4-D 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*.

---

# divPerElem

---

Divide two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 divPerElem(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Divide two 4-D vectors element by element.

## Notes

---

Floating-point behavior matches standard library function divf4.



---

# dot

---

Compute the dot product of two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 dot(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

Dot product of the specified 4-D vectors

## Description

---

Compute the dot product of two 4-D vectors.

---

# length

---

Compute the length of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 length(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Length of the specified 4-D vector

## Description

---

Compute the length of a 4-D vector.

---

# lengthSqr

---

Compute the square of the length of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 lengthSqr(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Square of the length of the specified 4-D vector

## Description

---

Compute the square of the length of a 4-D vector.

---

# lerp

---

Linear interpolation between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 lerp(
            vec_float4 t,
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>vec0</i>	4-D vector
<i>vec1</i>	4-D vector

## Return Values

---

Interpolated 4-D vector

## Description

---

Linearly interpolate between two 4-D vectors.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# maxElem

---

Maximum element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 maxElem(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Maximum value of all elements of *vec*

## Description

---

Compute the maximum value of all elements of a 4-D vector.

---

# maxPerElem

---

Maximum of two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 maxPerElem(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Create a 4-D vector in which each element is the maximum of the corresponding elements of the specified 4-D vectors.

---

# minElem

---

Minimum element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 minElem(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Minimum value of all elements of *vec*

## Description

---

Compute the minimum value of all elements of a 4-D vector.

---

# minPerElem

---

Minimum of two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 minPerElem(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Create a 4-D vector in which each element is the minimum of the corresponding elements of two specified 4-D vectors.



---

# mulPerElem

---

Multiply two 4-D vectors per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 mulPerElem(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

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

## Description

---

Multiply two 4-D vectors element by element.

---

# normalize

---

Normalize a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 normalize(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

The specified 4-D vector scaled to unit length

## Description

---

Compute a normalized 4-D vector.

## Notes

---

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

---

# operator \*

---

Multiply a 4-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 operator *(
            vec_float4 scalar,
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

<i>scalar</i>	Scalar value
<i>vec</i>	4-D vector

## Return Values

---

Scalar product of *vec* and *scalar*

## Description

---

Multiply a 4-D vector by a scalar.

---

# outer

---

Outer product of two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 outer(
            const Vector4 &vec0,
            const Vector4 &vec1
        );
    }
}
```

## Arguments

---

*vec0* 4-D vector  
*vec1* 4-D vector

## Return Values

---

The 4x4 matrix product of a column-vector, *vec0*, and a row-vector, *vec1*

## Description

---

Compute the outer product of two 4-D vectors.

---

# print

---

Print a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

None

## Description

---

Print a 4-D vector. Prints the 4-D vector transposed, that is, as a row instead of a column.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 4-D vector and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Vector4 &vec,
            const char *name
        );
    }
}
```

## Arguments

---

<i>vec</i>	4-D vector
<i>name</i>	String printed with the 4-D vector

## Return Values

---

None

## Description

---

Print a 4-D vector and an associated string identifier. Prints the 4-D vector transposed, that is, as a row instead of a column.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# recipPerElem

---

Compute the reciprocal of a 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 recipPerElem(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the reciprocal of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the reciprocal of the corresponding element of the specified 4-D vector.

## Notes

---

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

---

# rsqrtPerElem

---

Compute the reciprocal square root of a 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 rsqrtPerElem(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the reciprocal square root of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the reciprocal square root of the corresponding element of the specified 4-D vector.

## Notes

---

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



---

# select

---

Conditionally select between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 select(
            const Vector4 &vec0,
            const Vector4 &vec1,
            vec_uint4 select1
        );
    }
}
```

## Arguments

---

<i>vec0</i>	4-D vector
<i>vec1</i>	4-D vector
<i>select1</i>	For each of the four word slots, this mask selects either the 4-D vector in the corresponding slot of <i>vec0</i> or the 4-D 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 4-D 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 4-D vectors at each of the corresponding slots of *vec0* or *vec1*.

## Notes

---

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

---

# slerp

---

Spherical linear interpolation between two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 slerp(
            vec_float4 t,
            const Vector4 &unitVec0,
            const Vector4 &unitVec1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitVec0</i>	4-D vector, expected to be unit-length
<i>unitVec1</i>	4-D vector, expected to be unit-length

## Return Values

---

Interpolated 4-D vector

## Description

---

Perform spherical linear interpolation between two 4-D 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 4-D vector per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector4 sqrtPerElem(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

4-D vector in which each element is the square root of the corresponding element of the specified 4-D vector

## Description

---

Create a 4-D vector in which each element is the square root of the corresponding element of the specified 4-D vector.

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store four slots of an SoA 4-D vector as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void storeHalfFloats(
            const Vector4 &vec,
            vec_ushort8 *twoQuads
        );
    }
}
```

## Arguments

---

<i>vec</i>	4-D vector
<i>twoQuads</i>	An output array of 2 quadwords containing 16 half-floats

## Return Values

---

None

## Description

---

Store four slots of an SoA 4-D vector in two quadwords of half-float values. Numbering slots of *vec* as 0..3, the output is {x0,y0,z0,w0,x1,y1,z1,w1,x2,y2,z2,w2,x3,y3,z3,w3}.

---

# sum

---

Compute the sum of all elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 sum(
            const Vector4 &vec
        );
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Sum of all elements of *vec*

## Description

---

Compute the sum of all elements of a 4-D vector.

---

# 3-D Point Functions

## absPerElem

---

Compute the absolute value of a 3-D point per element.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 absPerElem(
            const Point3 &pnt
        );
    }
}
```

### Arguments

---

*pnt* 3-D point

### Return Values

---

3-D 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 3-D point.

---

# copySignPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 copySignPerElem(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D 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*.

---

# dist

---

Compute the distance between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 dist(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

Distance between two 3-D points

## Description

---

Compute the distance between two 3-D points.



---

# distFromOrigin

---

Compute the distance of a 3-D point from the coordinate-system origin.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 distFromOrigin(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Distance of a 3-D point from the origin

## Description

---

Compute the distance of a 3-D point from the coordinate-system origin.

---

# distSqr

---

Compute the square of the distance between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 distSqr(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

Square of the distance between two 3-D points

## Description

---

Compute the square of the distance between two 3-D points.

---

# distSqrFromOrigin

---

Compute the square of the distance of a 3-D point from the coordinate-system origin.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 distSqrFromOrigin(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Square of the distance of a 3-D point from the origin

## Description

---

Compute the square of the distance of a 3-D point from the coordinate-system origin.

---

# divPerElem

---

Divide two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 divPerElem(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the quotient of the corresponding elements of the specified 3-D points

## Description

---

Divide two 3-D points element by element.

## Notes

---

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

---

# lerp

---

Linear interpolation between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 lerp(
            vec_float4 t,
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point

## Return Values

---

Interpolated 3-D point

## Description

---

Linearly interpolate between two 3-D points.

## Notes

---

Does not clamp *t* between 0 and 1.

---

# loadXYZArray

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void loadXYZArray(
            Point3 &pnt,
            const vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt</i>	An output 3-D point
<i>threeQuads</i>	Array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

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

---

# maxElem

---

Maximum element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 maxElem(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Maximum value of all elements of *pnt*

## Description

---

Compute the maximum value of all elements of a 3-D point.

---

# maxPerElem

---

Maximum of two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 maxPerElem(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the maximum of the corresponding elements of the specified 3-D points

## Description

---

Create a 3-D point in which each element is the maximum of the corresponding elements of the specified 3-D points.



---

# minElem

---

Minimum element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 minElem(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Minimum value of all elements of *pnt*

## Description

---

Compute the minimum value of all elements of a 3-D point.

---

# minPerElem

---

Minimum of two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 minPerElem(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the minimum of the corresponding elements of the specified 3-D points

## Description

---

Create a 3-D point in which each element is the minimum of the corresponding elements of two specified 3-D points.

---

# mulPerElem

---

Multiply two 3-D points per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 mulPerElem(
            const Point3 &pnt0,
            const Point3 &pnt1
        );
    }
}
```

## Arguments

---

*pnt0* 3-D point  
*pnt1* 3-D point

## Return Values

---

3-D point in which each element is the product of the corresponding elements of the specified 3-D points

## Description

---

Multiply two 3-D points element by element.

---

# print

---

Print a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3-D point and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Point3 &pnt,
            const char *name
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>name</i>	String printed with the 3-D point

## Return Values

---

None

## Description

---

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

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# projection

---

Scalar projection of a 3-D point on a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 projection(
            const Point3 &pnt,
            const Vector3 &unitVec
        );
    }
}
```

## Arguments

---

*pnt*            3-D point  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

Scalar projection of the 3-D point on the unit-length 3-D vector

## Description

---

Scalar projection of a 3-D point on a unit-length 3-D vector (dot product).

---

# recipPerElem

---

Compute the reciprocal of a 3-D point per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 recipPerElem(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

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

---

# rsqrtPerElem

---

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

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 rsqrtPerElem(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

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



---

# scale

---

Apply uniform scale to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 scale(
            const Point3 &pnt,
            vec_float4 scaleVal
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>scaleVal</i>	Scalar value

## Return Values

---

3-D point in which every element is multiplied by the scalar value

## Description

---

Apply uniform scale to a 3-D point.

---

# scale

---

Apply non-uniform scale to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 scale(
            const Point3 &pnt,
            const Vector3 &scaleVec
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>scaleVec</i>	3-D vector

## Return Values

---

3-D point in which each element is the product of the corresponding elements of the specified 3-D point and 3-D vector

## Description

---

Apply non-uniform scale to a 3-D point.

---

# select

---

Conditionally select between two 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 select(
            const Point3 &pnt0,
            const Point3 &pnt1,
            vec_uint4 select1
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>select1</i>	For each of the four word slots, this mask selects either the 3-D point in the corresponding slot of <i>pnt0</i> or the 3-D 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 3-D 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 3-D 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 3-D point per element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Point3 sqrtPerElem(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

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

## Description

---

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

## Notes

---

Floating-point behavior matches standard library function sqrtf4.

---

# storeHalfFloats

---

Store eight slots of two SoA 3-D points as half-floats.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void storeHalfFloats(
            const Point3 &pnt0,
            const Point3 &pnt1,
            vec_ushort8 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt0</i>	3-D point
<i>pnt1</i>	3-D point
<i>threeQuads</i>	An output array of 3 quadwords containing 24 half-floats

## Return Values

---

None

## Description

---

Store eight slots of two SoA 3-D points in three quadwords of half-float values. Numbering slots of *pnt0* as 0..3 and slots of *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}.

---

# storeXYZArray

---

Store four slots of an SoA 3-D point in three quadwords.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void storeXYZArray(
            const Point3 &pnt,
            vec_float4 *threeQuads
        );
    }
}
```

## Arguments

---

<i>pnt</i>	3-D point
<i>threeQuads</i>	An output array of 3 quadwords containing 12 floats

## Return Values

---

None

## Description

---

Store four slots of an SoA 3-D point in three quadwords as {x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3}.

---

# sum

---

Compute the sum of all elements of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 sum(
            const Point3 &pnt
        );
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Sum of all elements of *pnt*

## Description

---

Compute the sum of all elements of a 3-D point.

---

# Quaternion Functions

## conj

---

Compute the conjugate of a quaternion.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat conj (
            const Quat &quat
        );
    }
}
```

### Arguments

---

*quat* Quaternion

### Return Values

---

Conjugate of the specified quaternion

### Description

---

Compute the conjugate of a quaternion.



---

# dot

---

Compute the dot product of two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 dot(
            const Quat &quat0,
            const Quat &quat1
        );
    }
}
```

## Arguments

---

*quat0* Quaternion  
*quat1* Quaternion

## Return Values

---

Dot product of the specified quaternions

## Description

---

Compute the dot product of two quaternions.

---

# length

---

Compute the length of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 length(
            const Quat &quat
        );
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Length of the specified quaternion

## Description

---

Compute the length of a quaternion.

---

# lerp

---

Linear interpolation between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat lerp(
            vec_float4 t,
            const Quat &quat0,
            const Quat &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.

---

# norm

---

Compute the norm of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 norm(
            const Quat &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.

---

# normalize

---

Normalize a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat normalize(
            const Quat &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.

---

# operator \*

---

Multiply a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat operator *(
            vec_float4 scalar,
            const Quat &quat
        );
    }
}
```

## Arguments

---

*scalar*    Scalar value  
*quat*      Quaternion

## Return Values

---

Scalar product of *quat* and *scalar*

## Description

---

Multiply a quaternion by a scalar.

---

# print

---

Print a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Quat &quat
        );
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

None

## Description

---

Print a quaternion.

## Notes

---

Function is only defined when `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a quaternion and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Quat &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 `_VECTORMATH_DEBUG` is defined.



---

# rotate

---

Use a unit-length quaternion to rotate a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Vector3 rotate(
            const Quat &unitQuat,
            const Vector3 &vec
        );
    }
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>vec</i>	3-D vector

## Return Values

---

The rotated 3-D vector, equivalent to  $\text{unitQuat} * \text{Quat}(\text{vec}, 0) * \text{conj}(\text{unitQuat})$

## Description

---

Rotate a 3-D vector by applying a unit-length quaternion.

---

# select

---

Conditionally select between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat select(
            const Quat &quat0,
            const Quat &quat1,
            vec_uint4 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.

---

# slerp

---

Spherical linear interpolation between two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat slerp(
            vec_float4 t,
            const Quat &unitQuat0,
            const Quat &unitQuat1
        );
    }
}
```

## Arguments

---

<i>t</i>	Interpolation parameter
<i>unitQuat0</i>	Quaternion, expected to be unit-length
<i>unitQuat1</i>	Quaternion, expected to be unit-length

## 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.

---

# squad

---

Spherical quadrangle interpolation.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Quat squad(
            vec_float4 t,
            const Quat &unitQuat0,
            const Quat &unitQuat1,
            const Quat &unitQuat2,
            const Quat &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.

---

# 3x3 Matrix Functions

## absPerElem

---

Compute the absolute value of a 3x3 matrix per element.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 absPerElem(
            const Matrix3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 appendScale (
            const Matrix3 &mat,
            const Vector3 &scaleVec
        );
    }
}
```

## Arguments

---

<i>mat</i>	3x3 matrix
<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# determinant

---

Determinant of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 determinant(
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

The determinant of *mat*

## Description

---

Compute the determinant of a 3x3 matrix.

---

# inverse

---

Compute the inverse of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 inverse(
            const Matrix3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 mulPerElem(
            const Matrix3 &mat0,
            const Matrix3 &mat1
        );
    }
}
```

## Arguments

---

*mat0* 3x3 matrix  
*mat1* 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.

---

# operator \*

---

Multiply a 3x3 matrix by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 operator *(
            vec_float4 scalar,
            const Matrix3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 prependScale (
            const Vector3 &scaleVec,
            const Matrix3 &mat
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# print

---

Print a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Matrix3 &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 `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3x3 matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Matrix3 &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 `_VECTORMATH_DEBUG` is defined.

---

# select

---

Conditionally select between two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 select(
            const Matrix3 &mat0,
            const Matrix3 &mat1,
            vec_uint4 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.

---

# transpose

---

Transpose of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix3 transpose(
            const Matrix3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 absPerElem(
            const Matrix4 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 affineInverse(
            const Matrix4 &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  $\text{inverse}(M)$ , whose translation vector is  $-\text{inverse}(M)*v$ , 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 appendScale (
            const Matrix4 &mat,
            const Vector3 &scaleVec
        );
    }
}
```

## Arguments

---

<i>mat</i>	4x4 matrix
<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# determinant

---

Determinant of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline vec_float4 determinant(
            const Matrix4 &mat
        );
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

The determinant of *mat*

## Description

---

Compute the determinant of a 4x4 matrix.

---

# inverse

---

Compute the inverse of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 inverse(
            const Matrix4 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 mulPerElem(
            const Matrix4 &mat0,
            const Matrix4 &mat1
        );
    }
}
```

## Arguments

---

*mat0* 4x4 matrix  
*mat1* 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.

---

# operator \*

---

Multiply a 4x4 matrix by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 operator *(
            vec_float4 scalar,
            const Matrix4 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 orthoInverse (
            const Matrix4 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 prependScale (
            const Vector3 &scaleVec,
            const Matrix4 &mat
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.



---

# print

---

Print a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Matrix4 &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 `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 4x4 matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Matrix4 &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 `_VECTORMATH_DEBUG` is defined.

---

# select

---

Conditionally select between two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 select(
            const Matrix4 &mat0,
            const Matrix4 &mat1,
            vec_uint4 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.

---

# transpose

---

Transpose of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Matrix4 transpose(
            const Matrix4 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 absPerElem(
            const Transform3 &tfrm
        );
    }
}
```

### 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 appendScale (
            const Transform3 &tfrm,
            const Vector3 &scaleVec
        );
    }
}
```

## Arguments

---

*tfrm*          3x4 transformation matrix  
*scaleVec*    3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# inverse

---

Inverse of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 inverse(
            const Transform3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 mulPerElem(
            const Transform3 &tfrm0,
            const Transform3 &tfrm1
        );
    }
}
```

## Arguments

---

*tfrm0* 3x4 transformation matrix  
*tfrm1* 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.



---

# orthoInverse

---

Compute the inverse of a 3x4 transformation matrix, expected to have an orthogonal upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 orthoInverse (
            const Transform3 &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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 prependScale(
            const Vector3 &scaleVec,
            const Transform3 &tfrm
        );
    }
}
```

## Arguments

---

<i>scaleVec</i>	3-D 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 3-D vector.

## Notes

---

Faster than creating and multiplying a scale transformation matrix.

---

# print

---

Print a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Transform3 &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 `_VECTORMATH_DEBUG` is defined.

---

# print

---

Print a 3x4 transformation matrix and an associated string identifier.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline void print(
            const Transform3 &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 `_VECTORMATH_DEBUG` is defined.

---

# select

---

Conditionally select between two 3x4 transformation matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        inline const Transform3 select(
            const Transform3 &tfrm0,
            const Transform3 &tfrm1,
            vec_uint4 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.

---

# **Vectormath::Soa::Matrix3**

# Summary

## Vectormath::Soa::Matrix3

A set of four 3x3 matrices in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.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="#">get4Aos</a>	Extract four AoS 3x3 matrices.
<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 a 3x3 matrix.
<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>	Replicate an AoS 3x3 matrix.
<a href="#">Matrix3</a>	Insert four AoS 3x3 matrices.
<a href="#">operator *</a>	Multiply a 3x3 matrix by a scalar.
<a href="#">operator *</a>	Multiply a 3x3 matrix by a 3-D vector.
<a href="#">operator *</a>	Multiply two 3x3 matrices.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a 3x3 matrix.
<a href="#">operator +</a>	Add two 3x3 matrices.
<a href="#">operator +=</a>	Perform compound assignment and addition with a 3x3 matrix.
<a href="#">operator -</a>	Subtract a 3x3 matrix from another 3x3 matrix.
<a href="#">operator -</a>	Negate all elements of a 3x3 matrix.
<a href="#">operator -=</a>	Perform compound assignment and subtraction by a 3x3 matrix.
<a href="#">operator =</a>	Assign one 3x3 matrix to another.
<a href="#">operator []</a>	Subscripting operator to set or get a column.

---

Methods	Description
<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 3-D 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.
<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.



---

# Constructors and Destructors

## Matrix3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Matrix3

---

Copy a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

None

## Description

---

Construct a copy of a 3x3 matrix.

---

# Matrix3

---

Construct a 3x3 matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3(
                const Vector3 &col0,
                const Vector3 &col1,
                const Vector3 &col2
            );
        };
    }
}
```

## Arguments

---

*col0* 3-D vector  
*col1* 3-D vector  
*col2* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3x3 matrix containing the specified columns.

---

# Matrix3

---

Construct a 3x3 rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            explicit inline Matrix3(
                const Quat &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.

---

# Matrix3

---

Set all elements of a 3x3 matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            explicit inline Matrix3(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3x3 matrix with all elements set to the scalar value argument.

---

# Matrix3

---

Replicate an AoS 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3(
                const Aos::Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat*    AoS 3x3 matrix

## Return Values

---

None

## Description

---

Replicate an AoS 3x3 matrix in all four slots of an SoA 3x3 matrix.

---

# Matrix3

---

Insert four AoS 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3(
                const Aos::Matrix3 &mat0,
                const Aos::Matrix3 &mat1,
                const Aos::Matrix3 &mat2,
                const Aos::Matrix3 &mat3
            );
        };
    };
}
```

## Arguments

---

*mat0*    AoS 3x3 matrix  
*mat1*    AoS 3x3 matrix  
*mat2*    AoS 3x3 matrix  
*mat3*    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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Matrix3 operator *(
                vec_float4 scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 3x3 matrix and scalar

### Description

---

Multiply a 3x3 matrix by a scalar.



---

# operator \*

---

Multiply a 3x3 matrix by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 operator *(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Product of the specified 3x3 matrix and 3-D vector

## Description

---

Multiply a 3x3 matrix by a 3-D vector.

---

# operator \*

---

Multiply two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Matrix3 operator *(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Product of the specified 3x3 matrices

## Description

---

Multiply two 3x3 matrices.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &operator *=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &operator *=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and multiplication by a 3x3 matrix.

---

# operator+

---

Add two 3x3 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Matrix3 operator+(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Sum of the specified 3x3 matrices

## Description

---

Add two 3x3 matrices.

---

# operator+=

---

Perform compound assignment and addition with a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &operator+=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and addition with a 3x3 matrix.

---

# operator-

---

Subtract a 3x3 matrix from another 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Matrix3 operator-(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

Difference of the specified 3x3 matrices

## Description

---

Subtract a 3x3 matrix from another 3x3 matrix.

---

# operator-

---

Negate all elements of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Matrix3 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3x3 matrix containing negated elements of the specified 3x3 matrix

## Description

---

Negate all elements of a 3x3 matrix.



---

# operator-=

---

Perform compound assignment and subtraction by a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &operator-=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Perform compound assignment and subtraction by a 3x3 matrix.

---

# operator=

---

Assign one 3x3 matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &operator=(
                const Matrix3 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 3x3 matrix

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Assign one 3x3 matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Vector3 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-2

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Matrix3](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 operator[] (
                int col
            );
        }
    }
}
```

## 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static 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 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotation(
                vec_float4 radians,
                const Vector3 &unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 3x3 matrix

## Description

---

Construct a 3x3 matrix to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotation(
                const Quat &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.

---

# rotationX

---

Construct a 3x3 matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotationX(
                vec_float4 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.



---

# rotationY

---

Construct a 3x3 matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotationY(
                vec_float4 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.

---

# rotationZ

---

Construct a 3x3 matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotationZ(
                vec_float4 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.

---

# rotationZYX

---

Construct a 3x3 matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 rotationZYX(
                const Vector3 &radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to *rotationZ(radiansXYZ.getZ()) \* rotationY(radiansXYZ.getY()) \* rotationX(radiansXYZ.getX())*.

---

# scale

---

Construct a 3x3 matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            static inline const Matrix3 scale(
                const Vector3 &scaleVec
            );
        };
    }
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# Public Instance Methods

## get4Aos

---

Extract four AoS 3x3 matrices.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline void get4Aos(
                Aos::Matrix3 &result0,
                Aos::Matrix3 &result1,
                Aos::Matrix3 &result2,
                Aos::Matrix3 &result3
            );
        };
    }
}
```

### 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).

---

# getCol

---

Get the column of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 getCol(
                int col
            );
        }
    }
}
```

## 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.

---

# getCol0

---

Get column 0 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 3x3 matrix.

---

# getCol1

---

Get column 1 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 3x3 matrix.



---

# getCol2

---

Get column 2 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 3x3 matrix.

---

# getElem

---

Get the element of a 3x3 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline vec_float4 getElem(
                int col,
                int row
            );
        }
    }
}
```

## 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.

---

# getRow

---

Get the row of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline const Vector3 getRow(
                int row
            );
        }
    }
}
```

## 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.

---

# setCol

---

Set the column of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setCol(
                int col,
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-2
<i>vec</i>	3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set the column of a 3x3 matrix referred to by the specified index.

---

# setCol0

---

Set column 0 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setCol0(
                const Vector3 &col0
            );
        }
    }
}
```

## Arguments

---

*col0* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 0 of a 3x3 matrix.

---

# setCol1

---

Set column 1 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setCol1(
                const Vector3 &col1
            );
        }
    }
}
```

## Arguments

---

*col1* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 1 of a 3x3 matrix.

---

# setCol2

---

Set column 2 of a 3x3 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setCol2(
                const Vector3 &col2
            );
        }
    }
}
```

## Arguments

---

*col2* 3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set column 2 of a 3x3 matrix.

---

# setElem

---

Set the element of a 3x3 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setElem(
                int col,
                int row,
                vec_float4 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.



---

# setRow

---

Set the row of a 3x3 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix3 {
            inline Matrix3 &setRow(
                int row,
                const Vector3 &vec
            );
        };
    }
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-2
<i>vec</i>	3-D vector

## Return Values

---

A reference to the resulting 3x3 matrix

## Description

---

Set the row of a 3x3 matrix referred to by the specified index.

---

# **Vectormath::Soa::Matrix4**

# Summary

## Vectormath::Soa::Matrix4

A set of four 4x4 matrices in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.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 a 4x4 matrix.
<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 3-D vector.
<a href="#">Matrix4</a>	Construct a 4x4 matrix from a unit-length quaternion and a 3-D 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 4-D vector.
<a href="#">operator *</a>	Multiply a 4x4 matrix by a 3-D vector.
<a href="#">operator *</a>	Multiply a 4x4 matrix by a 3-D 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.

Methods	Description
<a href="#"><u>operator *=</u></a>	Perform compound assignment and multiplication by a 4x4 matrix.
<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>perspective</u></a>	Construct a perspective projection matrix.
<a href="#"><u>rotation</u></a>	Construct a 4x4 matrix to rotate around a unit-length 3-D vector.
<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>rotationY</u></a>	Construct a 4x4 matrix to rotate around the y axis.
<a href="#"><u>rotationZ</u></a>	Construct a 4x4 matrix to rotate around the z axis.
<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>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.

---

# Constructors and Destructors

## Matrix4

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Matrix4

---

Copy a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

None

## Description

---

Construct a copy of a 4x4 matrix.

---

# Matrix4

---

Construct a 4x4 matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Vector4 &col0,
                const Vector4 &col1,
                const Vector4 &col2,
                const Vector4 &col3
            );
        };
    };
}
```

## Arguments

---

*col0* 4-D vector  
*col1* 4-D vector  
*col2* 4-D vector  
*col3* 4-D vector

## Return Values

---

None

## Description

---

Construct a 4x4 matrix containing the specified columns.

---

# Matrix4

---

Construct a 4x4 matrix from a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            explicit inline Matrix4(
                const Transform3 &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).



---

# Matrix4

---

Construct a 4x4 matrix from a 3x3 matrix and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Matrix3 &mat,
                const Vector3 &translateVec
            );
        };
    }
}
```

## Arguments

---

<i>mat</i>	3x3 matrix
<i>translateVec</i>	3-D 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 3-D vector argument, and whose bottom row is (0,0,0,1).

---

# Matrix4

---

Construct a 4x4 matrix from a unit-length quaternion and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Quat &unitQuat,
                const Vector3 &translateVec
            );
        };
    }
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>translateVec</i>	3-D 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 3-D vector argument, and whose bottom row is (0,0,0,1).

---

# Matrix4

---

Set all elements of a 4x4 matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            explicit inline Matrix4(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

None

## Description

---

Construct a 4x4 matrix with all elements set to the scalar value argument.

---

# Matrix4

---

Replicate an AoS 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Aos::Matrix4 &mat
            );
        };
    };
}
```

## Arguments

---

*mat*    AoS 4x4 matrix

## Return Values

---

None

## Description

---

Replicate an AoS 4x4 matrix in all four slots of an SoA 4x4 matrix.

---

# Matrix4

---

Insert four AoS 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4(
                const Aos::Matrix4 &mat0,
                const Aos::Matrix4 &mat1,
                const Aos::Matrix4 &mat2,
                const Aos::Matrix4 &mat3
            );
        };
    };
}
```

## Arguments

---

*mat0*    AoS 4x4 matrix  
*mat1*    AoS 4x4 matrix  
*mat2*    AoS 4x4 matrix  
*mat3*    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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator *(
                vec_float4 scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 4x4 matrix and scalar

### Description

---

Multiply a 4x4 matrix by a scalar.

---

# operator \*

---

Multiply a 4x4 matrix by a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 operator *(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Product of the specified 4x4 matrix and 4-D vector

## Description

---

Multiply a 4x4 matrix by a 4-D vector.

---

# operator \*

---

Multiply a 4x4 matrix by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 operator *(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Product of the specified 4x4 matrix and 3-D vector

## Description

---

Multiply a 4x4 matrix by a 3-D vector treated as if it were a 4-D vector with the w element equal to 0.



---

# operator \*

---

Multiply a 4x4 matrix by a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 operator *(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Product of the specified 4x4 matrix and 3-D point

## Description

---

Multiply a 4x4 matrix by a 3-D point treated as if it were a 4-D vector with the w element equal to 1.

---

# operator \*

---

Multiply two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator *(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Product of the specified 4x4 matrices

## Description

---

Multiply two 4x4 matrices.

---

# operator \*

---

Multiply a 4x4 matrix by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator *(
                const Transform3 &tfrm
            );
        }
    }
}
```

## 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).

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator *=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator *=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and multiplication by a 4x4 matrix.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator *=(
                const Transform3 &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.

---

# operator+

---

Add two 4x4 matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator+(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Sum of the specified 4x4 matrices

## Description

---

Add two 4x4 matrices.

---

# operator+=

---

Perform compound assignment and addition with a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator+=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and addition with a 4x4 matrix.



---

# operator-

---

Subtract a 4x4 matrix from another 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator-(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

Difference of the specified 4x4 matrices

## Description

---

Subtract a 4x4 matrix from another 4x4 matrix.

---

# operator-

---

Negate all elements of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix4 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

4x4 matrix containing negated elements of the specified 4x4 matrix

## Description

---

Negate all elements of a 4x4 matrix.

---

# operator-=

---

Perform compound assignment and subtraction by a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator-=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Perform compound assignment and subtraction by a 4x4 matrix.

---

# operator=

---

Assign one 4x4 matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &operator=(
                const Matrix4 &mat
            );
        }
    }
}
```

## Arguments

---

*mat* 4x4 matrix

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Assign one 4x4 matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Vector4 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Matrix4](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

Indexed column

## Description

---

Subscripting operator invoked when applied to const [Matrix4](#).

## frustum

### Definition

$$\begin{array}{ccccccc} 2*zNear/(right-left) & 0 & (right+left)/(right-left) & 0 & & & \\ 0 & 2*zNear/(top-bottom) & (top+bottom)/(top-bottom) & 0 & & & \\ 0 & 0 & -(zFar+zNear)/(zFar-zNear) & & & & \\ -2*zFar*zNear/(zFar-zNear) & & & & & & \\ 0 & 0 & -1 & 0 & & & \end{array}$$

---

# identity

---

Construct an identity 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static 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.



---

# lookAt

---

Construct viewing matrix based on eye position, position looked at, and up direction.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 lookAt(
                const Point3 &eyePos,
                const Point3 &lookAtPos,
                const Vector3 &upVec
            );
        };
    }
}
```

## Arguments

---

<i>eyePos</i>	3-D point
<i>lookAtPos</i>	3-D point
<i>upVec</i>	3-D 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.

---

# orthographic

---

Construct an orthographic projection matrix.

## Definition

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 orthographic(
                vec_float4 left,
                vec_float4 right,
                vec_float4 bottom,
                vec_float4 top,
                vec_float4 zNear,
                vec_float4 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

$$\begin{array}{cccc} 2/(right-left) & 0 & 0 & -(right+left)/(right-left) \\ 0 & 2/(top-bottom) & 0 & -(top+bottom)/(top-bottom) \\ 0 & 0 & -2/(zFar-zNear) & -(zFar+zNear)/(zFar-zNear) \\ 0 & 0 & 0 & 1 \end{array}.$$

---

# perspective

---

Construct a perspective projection matrix.

## Definition

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 perspective(
                vec_float4 fovyRadians,
                vec_float4 aspect,
                vec_float4 zNear,
                vec_float4 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:

$$\begin{bmatrix} \cot(fovyRadians/2)/aspect & 0 & 0 & 0 \\ 0 & \cot(fovyRadians/2) & 0 & 0 \\ 0 & 0 & (zFar+zNear)/(zNear-zFar) & 2*zFar*zNear/(zNear-zFar) \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

---

# rotation

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotation(
                vec_float4 radians,
                const Vector3 &unitVec
            );
        };
    };
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed 4x4 matrix

## Description

---

Construct a 4x4 matrix to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotation(
                const Quat &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.

---

# rotationX

---

Construct a 4x4 matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotationX(
                vec_float4 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.

---

# rotationY

---

Construct a 4x4 matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotationY(
                vec_float4 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.

---

# rotationZ

---

Construct a 4x4 matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotationZ(
                vec_float4 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.



---

# rotationZYX

---

Construct a 4x4 matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 rotationZYX(
                const Vector3 &radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to *rotationZ(radiansXYZ.getZ()) \* rotationY(radiansXYZ.getY()) \* rotationX(radiansXYZ.getX())*.

---

# scale

---

Construct a 4x4 matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 scale(
                const Vector3 &scaleVec
            );
        }
    }
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# translation

---

Construct a 4x4 matrix to perform translation.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            static inline const Matrix4 translation(
                const Vector3 &translateVec
            );
        };
    };
}
```

## Arguments

---

*translateVec* 3-D 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*.

---

# Public Instance Methods

## get4Aos

---

Extract four AoS 4x4 matrices.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline void get4Aos(
                Aos::Matrix4 &result0,
                Aos::Matrix4 &result1,
                Aos::Matrix4 &result2,
                Aos::Matrix4 &result3
            );
        };
    }
}
```

### 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).

---

# getCol

---

Get the column of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getCol(
                int col
            );
        }
    }
}
```

## 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.

---

# getCol0

---

Get column 0 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 4x4 matrix.

---

# getCol1

---

Get column 1 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 4x4 matrix.

---

# getCol2

---

Get column 2 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 4x4 matrix.



---

# getCol3

---

Get column 3 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getCol3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 3

## Description

---

Get column 3 of a 4x4 matrix.

---

# getElem

---

Get the element of a 4x4 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline vec_float4 getElem(
                int col,
                int row
            );
        }
    }
}
```

## 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.

---

# getRow

---

Get the row of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector4 getRow(
                int row
            );
        }
    }
}
```

## 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.

---

# getTranslation

---

Get the translation component of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Vector3 getTranslation();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Translation component

## Description

---

Get the translation component of a 4x4 matrix.

---

# getUpper3x3

---

Get the upper-left 3x3 submatrix of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline const Matrix3 getUpper3x3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Upper-left 3x3 submatrix

## Description

---

Get the upper-left 3x3 submatrix of a 4x4 matrix.

---

# setCol

---

Set the column of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setCol(
                int col,
                const Vector4 &vec
            );
        };
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-3
<i>vec</i>	4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the column of a 4x4 matrix referred to by the specified index.

---

# setCol0

---

Set column 0 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setCol0(
                const Vector4 &col0
            );
        }
    }
}
```

## Arguments

---

*col0* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 0 of a 4x4 matrix.

---

# setCol1

---

Set column 1 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setCol1(
                const Vector4 &col1
            );
        }
    }
}
```

## Arguments

---

*col1* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 1 of a 4x4 matrix.



---

# setCol2

---

Set column 2 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setCol2(
                const Vector4 &col2
            );
        }
    }
}
```

## Arguments

---

*col2* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 2 of a 4x4 matrix.

---

# setCol3

---

Set column 3 of a 4x4 matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setCol3(
                const Vector4 &col3
            );
        }
    }
}
```

## Arguments

---

*col3* 4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set column 3 of a 4x4 matrix.

---

# setElem

---

Set the element of a 4x4 matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setElem(
                int col,
                int row,
                vec_float4 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.

---

# setRow

---

Set the row of a 4x4 matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setRow(
                int row,
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-3
<i>vec</i>	4-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the row of a 4x4 matrix referred to by the specified index.

---

# setTranslation

---

Set translation component.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setTranslation(
                const Vector3 &translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D vector

## Return Values

---

A reference to the resulting 4x4 matrix

## Description

---

Set the translation component of a 4x4 matrix equal to the specified 3-D vector.

## Notes

---

This function does not change the bottom row elements.

---

# setUpper3x3

---

Set the upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Matrix4 {
            inline Matrix4 &setUpper3x3(
                const Matrix3 &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.

---

## **Vectormath::Soa::Point3**

# Summary

## Vectormath::Soa::Point3

A set of four 3-D points in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.h>
class Point3;
```

### Description

A class representing a set of four 3-D points stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 3-D points.
<a href="#">getElem</a>	Get an x, y, or z element of a 3-D point by index.
<a href="#">getX</a>	Get the x element of a 3-D point.
<a href="#">getY</a>	Get the y element of a 3-D point.
<a href="#">getZ</a>	Get the z element of a 3-D point.
<a href="#">operator+</a>	Add a 3-D point to a 3-D vector.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3-D vector.
<a href="#">operator-</a>	Subtract a 3-D point from another 3-D point.
<a href="#">operator-</a>	Subtract a 3-D vector from a 3-D point.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3-D vector.
<a href="#">operator=</a>	Assign one 3-D 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="#">Point3</a>	Default constructor; does no initialization.
<a href="#">Point3</a>	Copy a 3-D point.
<a href="#">Point3</a>	Construct a 3-D point from x, y, and z elements.
<a href="#">Point3</a>	Copy elements from a 3-D vector into a 3-D point.
<a href="#">Point3</a>	Set all elements of a 3-D point to the same scalar value.
<a href="#">Point3</a>	Replicate an AoS 3-D point.
<a href="#">Point3</a>	Insert four AoS 3-D points.
<a href="#">setElem</a>	Set an x, y, or z element of a 3-D point by index.
<a href="#">setX</a>	Set the x element of a 3-D point.
<a href="#">setY</a>	Set the y element of a 3-D point.
<a href="#">setZ</a>	Set the z element of a 3-D point.



---

# Constructors and Destructors

## Point3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Point3

---

Copy a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

Construct a copy of a 3-D point.

---

# Point3

---

Construct a 3-D point from x, y, and z elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3(
                vec_float4 x,
                vec_float4 y,
                vec_float4 z
            );
        };
    };
}
```

## Arguments

---

*x* Scalar value  
*y* Scalar value  
*z* Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D point containing the specified x, y, and z elements.

---

# Point3

---

Copy elements from a 3-D vector into a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            explicit inline Point3(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3-D point containing the x, y, and z elements of the specified 3-D vector.

---

# Point3

---

Set all elements of a 3-D point to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            explicit inline Point3(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D point with all elements set to the scalar value argument.

---

# Point3

---

Replicate an AoS 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3(
                Aos::Point3 pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* AoS 3-D point

## Return Values

---

None

## Description

---

Replicate an AoS 3-D point in all four slots of an SoA 3-D point.

---

# Point3

---

Insert four AoS 3-D points.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3(
                Aos::Point3 pnt0,
                Aos::Point3 pnt1,
                Aos::Point3 pnt2,
                Aos::Point3 pnt3
            );
        };
    };
}
```

## Arguments

---

*pnt0*    AoS 3-D point  
*pnt1*    AoS 3-D point  
*pnt2*    AoS 3-D point  
*pnt3*    AoS 3-D point

## Return Values

---

None

## Description

---

Insert four AoS 3-D points into four slots of an SoA 3-D point (transpose the data format).

---

# Operator Methods

## operator+

---

Add a 3-D point to a 3-D vector.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline const Point3 operator+(
                const Vector3 &vec
            );
        }
    }
}
```

### Arguments

---

*vec* 3-D vector

### Return Values

---

Sum of the specified 3-D point and 3-D vector

### Description

---

Add a 3-D point to a 3-D vector.



---

# operator+=

---

Perform compound assignment and addition with a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &operator+=(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Perform compound assignment and addition with a 3-D vector.

---

# operator-

---

Subtract a 3-D point from another 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline const Vector3 operator-(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Difference of the specified 3-D points

## Description

---

Subtract a 3-D point from another 3-D point.

---

# operator-

---

Subtract a 3-D vector from a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline const Point3 operator-(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Difference of the specified 3-D point and 3-D vector

## Description

---

Subtract a 3-D vector from a 3-D point.

---

# operator-=

---

Perform compound assignment and subtraction by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &operator-=(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Perform compound assignment and subtraction by a 3-D vector.

---

# operator=

---

Assign one 3-D point to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &operator=(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Assign one 3-D point to another.

---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline vec_float4 operator[] (
                int idx
            );
        }
    }
}
```

## Arguments

---

*idx*     Index, expected in the range 0-2

## Return Values

---

Indexed element

## Description

---

Subscripting operator invoked when applied to const [Point3](#).

---

# Public Instance Methods

## get4Aos

---

Extract four AoS 3-D points.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline void get4Aos(
                Aos::Point3 &result0,
                Aos::Point3 &result1,
                Aos::Point3 &result2,
                Aos::Point3 &result3
            );
        };
    };
}
```

### Arguments

---

<i>result0</i>	An output AoS 3-D point
<i>result1</i>	An output AoS 3-D point
<i>result2</i>	An output AoS 3-D point
<i>result3</i>	An output AoS 3-D point

### Return Values

---

None

### Description

---

Extract four AoS 3-D points from four slots of an SoA 3-D point (transpose the data format).



---

# getElem

---

Get an x, y, or z element of a 3-D point by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline vec_float4 getElem(
                int idx
            );
        }
    }
}
```

## 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 3-D point by specifying an index of 0, 1, or 2, respectively.

---

# getX

---

Get the x element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline vec_float4 getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 3-D point

## Description

---

Get the x element of a 3-D point.

---

# getY

---

Get the y element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline vec_float4 getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 3-D point

## Description

---

Get the y element of a 3-D point.

---

# getZ

---

Get the z element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline vec_float4 getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 3-D point

## Description

---

Get the z element of a 3-D point.

---

# setElem

---

Set an x, y, or z element of a 3-D point by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &setElem(
                int idx,
                vec_float4 value
            );
        };
    };
}
```

## Arguments

---

*idx*      Index, expected in the range 0-2  
*value*    Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set an x, y, or z element of a 3-D point by specifying an index of 0, 1, or 2, respectively.

---

# setX

---

Set the x element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &setX(
                vec_float4 x
            );
        }
    }
}
```

## Arguments

---

$x$  Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the x element of a 3-D point to the specified scalar value.

---

# setY

---

Set the y element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &setY(
                vec_float4 y
            );
        }
    }
}
```

## Arguments

---

*y* Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the y element of a 3-D point to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Point3 {
            inline Point3 &setZ(
                vec_float4 z
            );
        }
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 3-D point

## Description

---

Set the z element of a 3-D point to the specified scalar value.



---

**Vectormath::Soa::Quat**

# Summary

## Vectormath::Soa::Quat

A set of four quaternions in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.h>
class Quat;
```

### Description

A class representing a set of four quaternions stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS quaternions.
<a href="#">getElem</a>	Get an x, y, z, or w element of a quaternion by index.
<a href="#">getW</a>	Get the w element of a quaternion.
<a href="#">getX</a>	Get the x element of a quaternion.
<a href="#">getXYZ</a>	Get the x, y, and z elements of a quaternion.
<a href="#">getY</a>	Get the y element of a quaternion.
<a href="#">getZ</a>	Get the z element of a quaternion.
<a href="#">identity</a>	Construct an identity quaternion.
<a href="#">operator *</a>	Multiply two quaternions.
<a href="#">operator *</a>	Multiply a quaternion by a scalar.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a quaternion.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator+</a>	Add two quaternions.
<a href="#">operator+=</a>	Perform compound assignment and addition with a quaternion.
<a href="#">operator-</a>	Subtract a quaternion from another quaternion.
<a href="#">operator-</a>	Negate all elements of a quaternion.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a quaternion.
<a href="#">operator/</a>	Divide a quaternion by a scalar.
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar.
<a href="#">operator=</a>	Assign one quaternion 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="#">Quat</a>	Default constructor; does no initialization.
<a href="#">Quat</a>	Copy a quaternion.
<a href="#">Quat</a>	Construct a quaternion from x, y, z, and w elements.
<a href="#">Quat</a>	Construct a quaternion from a 3-D vector and a scalar.
<a href="#">Quat</a>	Copy elements from a 4-D vector into a quaternion.
<a href="#">Quat</a>	Convert a rotation matrix to a unit-length quaternion.
<a href="#">Quat</a>	Set all elements of a quaternion to the same scalar value.
<a href="#">Quat</a>	Replicate an AoS quaternion.

---

Methods	Description
<a href="#"><u>Quat</u></a>	Insert four AoS quaternions.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate between two unit-length 3-D vectors.
<a href="#"><u>rotation</u></a>	Construct a quaternion to rotate around a unit-length 3-D 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 x, y, z, or w element of a quaternion by index.
<a href="#"><u>setW</u></a>	Set the w element of a quaternion.
<a href="#"><u>setX</u></a>	Set the x element 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>setZ</u></a>	Set the z element of a quaternion.

---

# Constructors and Destructors

## Quat

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Quat

---

Copy a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

None

## Description

---

Construct a copy of a quaternion.

---

# Quat

---

Construct a quaternion from x, y, z, and w elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat(
                vec_float4 x,
                vec_float4 y,
                vec_float4 z,
                vec_float4 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.

---

# Quat

---

Construct a quaternion from a 3-D vector and a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat(
                const Vector3 &xyz,
                vec_float4 w
            );
        };
    }
}
```

## Arguments

---

<i>xyz</i>	3-D vector
<i>w</i>	Scalar value

## Return Values

---

None

## Description

---

Construct a quaternion with the x, y, and z elements of the specified 3-D vector and with the w element set to the specified scalar.

---

# Quat

---

Copy elements from a 4-D vector into a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            explicit inline Quat(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

None

## Description

---

Construct a quaternion containing the x, y, z, and w elements of the specified 4-D vector.



---

# Quat

---

Convert a rotation matrix to a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            explicit inline Quat(
                const Matrix3 &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.

---

# Quat

---

Set all elements of a quaternion to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            explicit inline Quat(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

None

## Description

---

Construct a quaternion with all elements set to the scalar value argument.

---

# Quat

---

Replicate an AoS quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat(
                Aos::Quat quat
            );
        }
    }
}
```

## Arguments

---

*quat*    AoS quaternion

## Return Values

---

None

## Description

---

Replicate an AoS quaternion in all four slots of an SoA quaternion.

---

# Quat

---

Insert four AoS quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat(
                Aos::Quat quat0,
                Aos::Quat quat1,
                Aos::Quat quat2,
                Aos::Quat 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator *(
                const Quat &quat
            );
        };
    };
}
```

### Arguments

---

*quat* Quaternion

### Return Values

---

Product of the specified quaternions

### Description

---

Multiply two quaternions.

---

# operator \*

---

Multiply a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator *(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

Product of the specified quaternion and scalar

## Description

---

Multiply a quaternion by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator *=(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and multiplication by a quaternion.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator *=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and multiplication by a scalar.



---

# operator+

---

Add two quaternions.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator+(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Sum of the specified quaternions

## Description

---

Add two quaternions.

---

# operator+=

---

Perform compound assignment and addition with a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator+=(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and addition with a quaternion.

---

# operator-

---

Subtract a quaternion from another quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator-(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

Difference of the specified quaternions

## Description

---

Subtract a quaternion from another quaternion.

---

# operator-

---

Negate all elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Quaternion containing negated elements of the specified quaternion

## Description

---

Negate all elements of a quaternion.

---

# operator-=

---

Perform compound assignment and subtraction by a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator-=(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and subtraction by a quaternion.

---

# operator/

---

Divide a quaternion by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Quat operator/(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

Quotient of the specified quaternion and scalar

## Description

---

Divide a quaternion by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator/=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting quaternion

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one quaternion to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &operator=(
                const Quat &quat
            );
        }
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

A reference to the resulting quaternion

## Description

---

Assign one quaternion to another.



---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 operator[] (
                int idx
            );
        }
    }
}
```

## 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

## identity

---

Construct an identity quaternion.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat identity();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed quaternion

### Description

---

Construct an identity quaternion equal to (0,0,0,1).

---

# rotation

---

Construct a quaternion to rotate between two unit-length 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat rotation(
                const Vector3 &unitVec0,
                const Vector3 &unitVec1
            );
        };
    };
}
```

## Arguments

---

*unitVec0* 3-D vector, expected to be unit-length  
*unitVec1* 3-D vector, expected to be unit-length

## Return Values

---

The constructed quaternion

## Description

---

Construct a quaternion to rotate between two unit-length 3-D vectors.

## Notes

---

The result is unpredictable if *unitVec0* and *unitVec1* point in opposite directions.

---

# rotation

---

Construct a quaternion to rotate around a unit-length 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat rotation(
                vec_float4 radians,
                const Vector3 &unitVec
            );
        };
    }
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D vector, expected to be unit-length

## Return Values

---

The constructed quaternion

## Description

---

Construct a quaternion to rotate around a unit-length 3-D vector by the specified radians angle.

---

# rotationX

---

Construct a quaternion to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat rotationX(
                vec_float4 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.

---

# rotationY

---

Construct a quaternion to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat rotationY(
                vec_float4 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.

---

# rotationZ

---

Construct a quaternion to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            static inline const Quat rotationZ(
                vec_float4 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.



---

# Public Instance Methods

## get4Aos

---

Extract four AoS quaternions.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline void get4Aos(
                Aos::Quat &result0,
                Aos::Quat &result1,
                Aos::Quat &result2,
                Aos::Quat &result3
            );
        };
    }
}
```

### 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).

---

# getElem

---

Get an x, y, z, or w element of a quaternion by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 getElem(
                int idx
            );
        }
    }
}
```

## 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.

---

# getW

---

Get the w element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 getW();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

w element of a quaternion

## Description

---

Get the w element of a quaternion.

---

# getX

---

Get the x element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a quaternion

## Description

---

Get the x element of a quaternion.

---

# getXYZ

---

Get the x, y, and z elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline const Vector3 getXYZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing x, y, and z elements

## Description

---

Extract a quaternion's x, y, and z elements into a 3-D vector.

---

# getY

---

Get the y element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a quaternion

## Description

---

Get the y element of a quaternion.

---

# getZ

---

Get the z element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline vec_float4 getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a quaternion

## Description

---

Get the z element of a quaternion.

---

# setElem

---

Set an x, y, z, or w element of a quaternion by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setElem(
                int idx,
                vec_float4 value
            );
        };
    }
}
```

## Arguments

---

*idx*      Index, expected in the range 0-3  
*value*    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.



---

# setW

---

Set the w element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setW(
                vec_float4 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.

---

# setX

---

Set the x element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setX(
                vec_float4 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.

---

# setXYZ

---

Set the x, y, and z elements of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setXYZ(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting quaternion

## Description

---

Set the x, y, and z elements to those of the specified 3-D vector.

## Notes

---

This function does not change the w element.

---

# setY

---

Set the y element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setY(
                vec_float4 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.

---

# setZ

---

Set the z element of a quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Quat {
            inline Quat &setZ(
                vec_float4 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.

---

# **Vectormath::Soa::Transform3**

# Summary

## Vectormath::Soa::Transform3

A set of four 3x4 transformation matrices in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.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 3-D vector.
<a href="#">operator *</a>	Multiply a 3x4 transformation matrix by a 3-D 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 3-D 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.

Methods	Description
<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.
<a href="#">setCol3</a>	Set column 3 of a 3x4 transformation matrix.
<a href="#">setElem</a>	Set the element of a 3x4 transformation matrix referred to by column and row indices.
<a href="#">setRow</a>	Set the row of a 3x4 transformation matrix referred to by the specified index.
<a href="#">setTranslation</a>	Set translation component.
<a href="#">setUpper3x3</a>	Set the upper-left 3x3 submatrix.
<a href="#">Transform3</a>	Default constructor; does no initialization.
<a href="#">Transform3</a>	Copy a 3x4 transformation matrix.
<a href="#">Transform3</a>	Construct a 3x4 transformation matrix containing the specified columns.
<a href="#">Transform3</a>	Construct a 3x4 transformation matrix from a 3x3 matrix and a 3-D vector.
<a href="#">Transform3</a>	Construct a 3x4 transformation matrix from a unit-length quaternion and a 3-D vector.
<a href="#">Transform3</a>	Set all elements of a 3x4 transformation matrix to the same scalar value.
<a href="#">Transform3</a>	Replicate an AoS 3x4 transformation matrix.
<a href="#">Transform3</a>	Insert four AoS 3x4 transformation matrices.
<a href="#">translation</a>	Construct a 3x4 transformation matrix to perform translation.



---

# Constructors and Destructors

## Transform3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Transform3

---

Copy a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

None

## Description

---

Construct a copy of a 3x4 transformation matrix.

---

# Transform3

---

Construct a 3x4 transformation matrix containing the specified columns.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Vector3 &col0,
                const Vector3 &col1,
                const Vector3 &col2,
                const Vector3 &col3
            );
        };
    };
}
```

## Arguments

---

*col0* 3-D vector  
*col1* 3-D vector  
*col2* 3-D vector  
*col3* 3-D vector

## Return Values

---

None

## Description

---

Construct a 3x4 transformation matrix containing the specified columns.

---

# Transform3

---

Construct a 3x4 transformation matrix from a 3x3 matrix and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Matrix3 &tfrm,
                const Vector3 &translateVec
            );
        };
    };
}
```

## Arguments

---

<i>tfrm</i>	3x3 matrix
<i>translateVec</i>	3-D 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 3-D vector argument.

---

# Transform3

---

Construct a 3x4 transformation matrix from a unit-length quaternion and a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Quat &unitQuat,
                const Vector3 &translateVec
            );
        };
    }
}
```

## Arguments

---

<i>unitQuat</i>	Quaternion, expected to be unit-length
<i>translateVec</i>	3-D 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 3-D vector argument.

---

# Transform3

---

Set all elements of a 3x4 transformation matrix to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            explicit inline Transform3(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3x4 transformation matrix with all elements set to the scalar value argument.

---

# Transform3

---

Replicate an AoS 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Aos::Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm*    AoS 3x4 transformation matrix

## Return Values

---

None

## Description

---

Replicate an AoS 3x4 transformation matrix in all four slots of an SoA 3x4 transformation matrix.

---

# Transform3

---

Insert four AoS 3x4 transformation matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3(
                const Aos::Transform3 &tfrm0,
                const Aos::Transform3 &tfrm1,
                const Aos::Transform3 &tfrm2,
                const Aos::Transform3 &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 3-D vector.

### **Definition**

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 operator *(
                const Vector3 &vec
            );
        }
    }
}
```

### **Arguments**

---

*vec* 3-D vector

### **Return Values**

---

Product of the specified 3x4 transformation matrix and 3-D vector

### **Description**

---

Applies the 3x3 upper-left submatrix (but not the translation component) of a 3x4 transformation matrix to a 3-D vector.

---

# operator \*

---

Multiply a 3x4 transformation matrix by a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Point3 operator *(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Product of the specified 3x4 transformation matrix and 3-D point

## Description

---

Applies the 3x3 upper-left submatrix and the translation component of a 3x4 transformation matrix to a 3-D point.

---

# operator \*

---

Multiply two 3x4 transformation matrices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Transform3 operator *(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

Product of the specified 3x4 transformation matrices

## Description

---

Multiply two 3x4 transformation matrices.

---

# operator \*=

---

Perform compound assignment and multiplication by a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &operator *=(
                const Transform3 &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.

---

# operator=

---

Assign one 3x4 transformation matrix to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &operator=(
                const Transform3 &tfrm
            );
        }
    }
}
```

## Arguments

---

*tfrm* 3x4 transformation matrix

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Assign one 3x4 transformation matrix to another.

---

# operator[]

---

Subscripting operator to set or get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Vector3 &operator[] (
                int col
            );
        }
    }
}
```

## Arguments

---

*col*     Index, expected in the range 0-3

## Return Values

---

A reference to indexed column

## Description

---

Subscripting operator invoked when applied to non-const [Transform3](#).

---

# operator[]

---

Subscripting operator to get a column.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 operator[] (
                int col
            );
        }
    }
}
```

## 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static 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 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotation(
                vec_float4 radians,
                const Vector3 &unitVec
            );
        };
    };
}
```

## Arguments

---

*radians*    Scalar value  
*unitVec*    3-D 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 3-D vector by the specified radians angle.

---

# rotation

---

Construct a rotation matrix from a unit-length quaternion.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotation(
                const Quat &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.

---

# rotationX

---

Construct a 3x4 transformation matrix to rotate around the x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotationX(
                vec_float4 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.

---

# rotationY

---

Construct a 3x4 transformation matrix to rotate around the y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotationY(
                vec_float4 radians
            );
        }
    }
}
```

## Arguments

---

*radians*    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.

---

# rotationZ

---

Construct a 3x4 transformation matrix to rotate around the z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotationZ(
                vec_float4 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.

---

# rotationZYX

---

Construct a 3x4 transformation matrix to rotate around the x, y, and z axes.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 rotationZYX(
                const Vector3 &radiansXYZ
            );
        }
    }
}
```

## Arguments

---

*radiansXYZ* 3-D 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 3-D vector. Equivalent to *rotationZ(radiansXYZ.getZ()) \* rotationY(radiansXYZ.getY()) \* rotationX(radiansXYZ.getX())*.

---

# scale

---

Construct a 3x4 transformation matrix to perform scaling.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 scale(
                const Vector3 &scaleVec
            );
        }
    }
}
```

## Arguments

---

*scaleVec* 3-D 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*.

---

# translation

---

Construct a 3x4 transformation matrix to perform translation.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            static inline const Transform3 translation(
                const Vector3 &translateVec
            );
        };
    };
}
```

## Arguments

---

*translateVec* 3-D 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline void get4Aos(
                Aos::Transform3 &result0,
                Aos::Transform3 &result1,
                Aos::Transform3 &result2,
                Aos::Transform3 &result3
            );
        };
    }
}
```

### 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).

---

# getCol

---

Get the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getCol(
                int col
            );
        }
    }
}
```

## 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.

---

# getCol0

---

Get column 0 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getCol0();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 0

## Description

---

Get column 0 of a 3x4 transformation matrix.

---

# getCol1

---

Get column 1 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getCol1();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 1

## Description

---

Get column 1 of a 3x4 transformation matrix.

---

# getCol2

---

Get column 2 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getCol2();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Column 2

## Description

---

Get column 2 of a 3x4 transformation matrix.

---

# getCol3

---

Get column 3 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getCol3();
        }
    }
}
```

## 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline vec_float4 getElem(
                int col,
                int row
            );
        };
    }
}
```

## 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.

---

# getRow

---

Get the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector4 getRow(
                int row
            );
        }
    }
}
```

## 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.



---

# getTranslation

---

Get the translation component of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Vector3 getTranslation();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Translation component

## Description

---

Get the translation component of a 3x4 transformation matrix.

---

# getUpper3x3

---

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline const Matrix3 getUpper3x3();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

Upper-left 3x3 submatrix

## Description

---

Get the upper-left 3x3 submatrix of a 3x4 transformation matrix.

---

# setCol

---

Set the column of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setCol(
                int col,
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

<i>col</i>	Index, expected in the range 0-3
<i>vec</i>	3-D 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.

---

# setCol0

---

Set column 0 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setCol0(
                const Vector3 &col0
            );
        }
    }
}
```

## Arguments

---

*col0* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 0 of a 3x4 transformation matrix.

---

# setCol1

---

Set column 1 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setCol1(
                const Vector3 &col1
            );
        }
    }
}
```

## Arguments

---

*col1* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 1 of a 3x4 transformation matrix.

---

# setCol2

---

Set column 2 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setCol2(
                const Vector3 &col2
            );
        }
    }
}
```

## Arguments

---

*col2* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 2 of a 3x4 transformation matrix.

---

# setCol3

---

Set column 3 of a 3x4 transformation matrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setCol3(
                const Vector3 &col3
            );
        }
    }
}
```

## Arguments

---

*col3* 3-D vector

## Return Values

---

A reference to the resulting 3x4 transformation matrix

## Description

---

Set column 3 of a 3x4 transformation matrix.

---

# setElem

---

Set the element of a 3x4 transformation matrix referred to by column and row indices.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setElem(
                int col,
                int row,
                vec_float4 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.



---

# setRow

---

Set the row of a 3x4 transformation matrix referred to by the specified index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setRow(
                int row,
                const Vector4 &vec
            );
        };
    };
}
```

## Arguments

---

<i>row</i>	Index, expected in the range 0-2
<i>vec</i>	4-D 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.

---

# setTranslation

---

Set translation component.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setTranslation(
                const Vector3 &translateVec
            );
        }
    }
}
```

## Arguments

---

*translateVec* 3-D 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 3-D vector.

---

# setUpper3x3

---

Set the upper-left 3x3 submatrix.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Transform3 {
            inline Transform3 &setUpper3x3(
                const Matrix3 &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.

---

# **Vectormath::Soa::Vector3**

# Summary

## Vectormath::Soa::Vector3

A set of four 3-D vectors in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.h>
class Vector3;
```

### Description

A class representing a set of four 3-D vectors stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 3-D vectors.
<a href="#">getElem</a>	Get an x, y, or z element of a 3-D vector by index.
<a href="#">getX</a>	Get the x element of a 3-D vector.
<a href="#">getY</a>	Get the y element of a 3-D vector.
<a href="#">getZ</a>	Get the z element of a 3-D vector.
<a href="#">operator *</a>	Multiply a 3-D vector by a scalar.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator+</a>	Add two 3-D vectors.
<a href="#">operator+</a>	Add a 3-D vector to a 3-D point.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 3-D vector.
<a href="#">operator-</a>	Subtract a 3-D vector from another 3-D vector.
<a href="#">operator-</a>	Negate all elements of a 3-D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 3-D vector.
<a href="#">operator/</a>	Divide a 3-D vector by a scalar.
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar.
<a href="#">operator=</a>	Assign one 3-D 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 x, y, or z element of a 3-D vector by index.
<a href="#">setX</a>	Set the x element of a 3-D vector.
<a href="#">setY</a>	Set the y element of a 3-D vector.
<a href="#">setZ</a>	Set the z element of a 3-D vector.
<a href="#">Vector3</a>	Default constructor; does no initialization.
<a href="#">Vector3</a>	Copy a 3-D vector.
<a href="#">Vector3</a>	Construct a 3-D vector from x, y, and z elements.
<a href="#">Vector3</a>	Copy elements from a 3-D point into a 3-D vector.
<a href="#">Vector3</a>	Set all elements of a 3-D vector to the same scalar value.
<a href="#">Vector3</a>	Replicate an AoS 3-D vector.
<a href="#">Vector3</a>	Insert four AoS 3-D vectors.
<a href="#">xAxis</a>	Construct x axis.
<a href="#">yAxis</a>	Construct y axis.

---

Methods	Description
<a href="#">zAxis</a>	Construct z axis.

---

# Constructors and Destructors

## Vector3

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Vector3

---

Copy a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

Construct a copy of a 3-D vector.



---

# Vector3

---

Construct a 3-D vector from x, y, and z elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3(
                vec_float4 x,
                vec_float4 y,
                vec_float4 z
            );
        };
    }
}
```

## Arguments

---

*x*   Scalar value  
*y*   Scalar value  
*z*   Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D vector containing the specified x, y, and z elements.

---

# Vector3

---

Copy elements from a 3-D point into a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            explicit inline Vector3(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

Construct a 3-D vector containing the x, y, and z elements of the specified 3-D point.

---

# Vector3

---

Set all elements of a 3-D vector to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            explicit inline Vector3(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 3-D vector with all elements set to the scalar value argument.

---

# Vector3

---

Replicate an AoS 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3(
                Aos::Vector3 vec
            );
        }
    }
}
```

## Arguments

---

*vec*    AoS 3-D vector

## Return Values

---

None

## Description

---

Replicate an AoS 3-D vector in all four slots of an SoA 3-D vector.

---

# Vector3

---

Insert four AoS 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3(
                Aos::Vector3 vec0,
                Aos::Vector3 vec1,
                Aos::Vector3 vec2,
                Aos::Vector3 vec3
            );
        };
    };
}
```

## Arguments

---

*vec0*    AoS 3-D vector  
*vec1*    AoS 3-D vector  
*vec2*    AoS 3-D vector  
*vec3*    AoS 3-D vector

## Return Values

---

None

## Description

---

Insert four AoS 3-D vectors into four slots of an SoA 3-D vector (transpose the data format).

---

# Operator Methods

## operator \*

---

Multiply a 3-D vector by a scalar.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Vector3 operator *(
                vec_float4 scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 3-D vector and scalar

### Description

---

Multiply a 3-D vector by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &operator *=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator+

---

Add two 3-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Vector3 operator+(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Sum of the specified 3-D vectors

## Description

---

Add two 3-D vectors.



---

# operator+

---

Add a 3-D vector to a 3-D point.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Point3 operator+(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

Sum of the specified 3-D vector and 3-D point

## Description

---

Add a 3-D vector to a 3-D point.

---

# operator+=

---

Perform compound assignment and addition with a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &operator+=(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and addition with a 3-D vector.

---

# operator-

---

Subtract a 3-D vector from another 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Vector3 operator-(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

Difference of the specified 3-D vectors

## Description

---

Subtract a 3-D vector from another 3-D vector.

---

# operator-

---

Negate all elements of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Vector3 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing negated elements of the specified 3-D vector

## Description

---

Negate all elements of a 3-D vector.

---

# operator-=

---

Perform compound assignment and subtraction by a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &operator-=(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and subtraction by a 3-D vector.

---

# operator/

---

Divide a 3-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline const Vector3 operator/(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

Quotient of the specified 3-D vector and scalar

## Description

---

Divide a 3-D vector by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &operator/=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one 3-D vector to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &operator=(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Assign one 3-D vector to another.



---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline vec_float4 operator[] (
                int idx
            );
        }
    }
}
```

## 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            static inline const Vector3 xAxis();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed 3-D vector

### Description

---

Construct a 3-D vector equal to (1,0,0).

---

# yAxis

---

Construct y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            static inline const Vector3 yAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 3-D vector

## Description

---

Construct a 3-D vector equal to (0,1,0).

---

# zAxis

---

Construct z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            static inline const Vector3 zAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 3-D vector

## Description

---

Construct a 3-D vector equal to (0,0,1).

---

# Public Instance Methods

## get4Aos

---

Extract four AoS 3-D vectors.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline void get4Aos(
                Aos::Vector3 &result0,
                Aos::Vector3 &result1,
                Aos::Vector3 &result2,
                Aos::Vector3 &result3
            );
        };
    }
}
```

### Arguments

---

<i>result0</i>	An output AoS 3-D vector
<i>result1</i>	An output AoS 3-D vector
<i>result2</i>	An output AoS 3-D vector
<i>result3</i>	An output AoS 3-D vector

### Return Values

---

None

### Description

---

Extract four AoS 3-D vectors from four slots of an SoA 3-D vector (transpose the data format).

---

# getElem

---

Get an x, y, or z element of a 3-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline vec_float4 getElem(
                int idx
            );
        }
    }
}
```

## 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 3-D vector by specifying an index of 0, 1, or 2, respectively.

---

# getX

---

Get the x element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline vec_float4 getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 3-D vector

## Description

---

Get the x element of a 3-D vector.



---

# getY

---

Get the y element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline vec_float4 getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 3-D vector

## Description

---

Get the y element of a 3-D vector.

---

# getZ

---

Get the z element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline vec_float4 getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 3-D vector

## Description

---

Get the z element of a 3-D vector.

---

# setElem

---

Set an x, y, or z element of a 3-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &setElem(
                int idx,
                vec_float4 value
            );
        };
    }
}
```

## Arguments

---

*idx*      Index, expected in the range 0-2  
*value*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set an x, y, or z element of a 3-D vector by specifying an index of 0, 1, or 2, respectively.

---

# setX

---

Set the x element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &setX(
                vec_float4 x
            );
        }
    }
}
```

## Arguments

---

$x$  Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the x element of a 3-D vector to the specified scalar value.

---

# setY

---

Set the y element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &setY(
                vec_float4 y
            );
        }
    }
}
```

## Arguments

---

*y* Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the y element of a 3-D vector to the specified scalar value.

---

# setZ

---

Set the z element of a 3-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector3 {
            inline Vector3 &setZ(
                vec_float4 z
            );
        }
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 3-D vector

## Description

---

Set the z element of a 3-D vector to the specified scalar value.

---

# **Vectormath::Soa::Vector4**

# Summary

## Vectormath::Soa::Vector4

A set of four 4-D vectors in structure-of-arrays format.

### Definition

```
#include <vectormath/cpp/vectormath_soa.h>
class Vector4;
```

### Description

A class representing a set of four 4-D vectors stored in structure-of-arrays (SoA) format.

### Methods Summary

Methods	Description
<a href="#">get4Aos</a>	Extract four AoS 4-D vectors.
<a href="#">getElem</a>	Get an x, y, z, or w element of a 4-D vector by index.
<a href="#">getW</a>	Get the w element of a 4-D vector.
<a href="#">getX</a>	Get the x element of a 4-D vector.
<a href="#">getXYZ</a>	Get the x, y, and z elements of a 4-D vector.
<a href="#">getY</a>	Get the y element of a 4-D vector.
<a href="#">getZ</a>	Get the z element of a 4-D vector.
<a href="#">operator *</a>	Multiply a 4-D vector by a scalar.
<a href="#">operator *=</a>	Perform compound assignment and multiplication by a scalar.
<a href="#">operator+</a>	Add two 4-D vectors.
<a href="#">operator+=</a>	Perform compound assignment and addition with a 4-D vector.
<a href="#">operator-</a>	Subtract a 4-D vector from another 4-D vector.
<a href="#">operator-</a>	Negate all elements of a 4-D vector.
<a href="#">operator-=</a>	Perform compound assignment and subtraction by a 4-D vector.
<a href="#">operator/</a>	Divide a 4-D vector by a scalar.
<a href="#">operator/=</a>	Perform compound assignment and division by a scalar.
<a href="#">operator=</a>	Assign one 4-D 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 x, y, z, or w element of a 4-D vector by index.
<a href="#">setW</a>	Set the w element of a 4-D vector.
<a href="#">setX</a>	Set the x element of a 4-D vector.
<a href="#">setXYZ</a>	Set the x, y, and z elements of a 4-D vector.
<a href="#">setY</a>	Set the y element of a 4-D vector.
<a href="#">setZ</a>	Set the z element of a 4-D vector.
<a href="#">Vector4</a>	Default constructor; does no initialization.
<a href="#">Vector4</a>	Copy a 4-D vector.
<a href="#">Vector4</a>	Construct a 4-D vector from x, y, z, and w elements.
<a href="#">Vector4</a>	Construct a 4-D vector from a 3-D vector and a scalar.
<a href="#">Vector4</a>	Copy x, y, and z from a 3-D vector into a 4-D vector, and set w to 0.



---

Methods	Description
<a href="#">Vector4</a>	Copy x, y, and z from a 3-D point into a 4-D vector, and set w to 1.
<a href="#">Vector4</a>	Copy elements from a quaternion into a 4-D vector.
<a href="#">Vector4</a>	Set all elements of a 4-D vector to the same scalar value.
<a href="#">Vector4</a>	Replicate an AoS 4-D vector.
<a href="#">Vector4</a>	Insert four AoS 4-D 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.

---

# Constructors and Destructors

## Vector4

---

Default constructor; does no initialization.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4() ;
        }
    }
}
```

### Arguments

---

None

### Return Values

---

None

### Description

---

Default constructor; does no initialization.

---

# Vector4

---

Copy a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4(
                const Vector4 &vec
            );
        };
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

None

## Description

---

Construct a copy of a 4-D vector.

---

# Vector4

---

Construct a 4-D vector from x, y, z, and w elements.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4(
                vec_float4 x,
                vec_float4 y,
                vec_float4 z,
                vec_float4 w
            );
        };
    }
}
```

## Arguments

---

*x* Scalar value  
*y* Scalar value  
*z* Scalar value  
*w* Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector containing the specified x, y, z, and w elements.

---

# Vector4

---

Construct a 4-D vector from a 3-D vector and a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4(
                const Vector3 &xyz,
                vec_float4 w
            );
        };
    }
}
```

## Arguments

---

<i>xyz</i>	3-D vector
<i>w</i>	Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D vector and with the w element set to the specified scalar.

---

# Vector4

---

Copy x, y, and z from a 3-D vector into a 4-D vector, and set w to 0.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            explicit inline Vector4(
                const Vector3 &vec
            );
        };
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D vector and with the w element set to 0.

---

# Vector4

---

Copy x, y, and z from a 3-D point into a 4-D vector, and set w to 1.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            explicit inline Vector4(
                const Point3 &pnt
            );
        }
    }
}
```

## Arguments

---

*pnt* 3-D point

## Return Values

---

None

## Description

---

Construct a 4-D vector with the x, y, and z elements of the specified 3-D point and with the w element set to 1.

---

# Vector4

---

Copy elements from a quaternion into a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            explicit inline Vector4(
                const Quat &quat
            );
        };
    }
}
```

## Arguments

---

*quat* Quaternion

## Return Values

---

None

## Description

---

Construct a 4-D vector containing the x, y, z, and w elements of the specified quaternion.



---

# Vector4

---

Set all elements of a 4-D vector to the same scalar value.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            explicit inline Vector4(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

None

## Description

---

Construct a 4-D vector with all elements set to the scalar value argument.

---

# Vector4

---

Replicate an AoS 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4(
                Aos::Vector4 vec
            );
        }
    }
}
```

## Arguments

---

*vec*    AoS 4-D vector

## Return Values

---

None

## Description

---

Replicate an AoS 4-D vector in all four slots of an SoA 4-D vector.

---

# Vector4

---

Insert four AoS 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4(
                Aos::Vector4 vec0,
                Aos::Vector4 vec1,
                Aos::Vector4 vec2,
                Aos::Vector4 vec3
            );
        };
    };
}
```

## Arguments

---

*vec0*    AoS 4-D vector  
*vec1*    AoS 4-D vector  
*vec2*    AoS 4-D vector  
*vec3*    AoS 4-D vector

## Return Values

---

None

## Description

---

Insert four AoS 4-D vectors into four slots of an SoA 4-D vector (transpose the data format).

---

# Operator Methods

## operator \*

---

Multiply a 4-D vector by a scalar.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector4 operator *(
                vec_float4 scalar
            );
        }
    }
}
```

### Arguments

---

*scalar* Scalar value

### Return Values

---

Product of the specified 4-D vector and scalar

### Description

---

Multiply a 4-D vector by a scalar.

---

# operator \*=

---

Perform compound assignment and multiplication by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &operator *=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and multiplication by a scalar.

---

# operator+

---

Add two 4-D vectors.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector4 operator+(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Sum of the specified 4-D vectors

## Description

---

Add two 4-D vectors.

---

# operator+=

---

Perform compound assignment and addition with a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &operator+=(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and addition with a 4-D vector.

---

# operator-

---

Subtract a 4-D vector from another 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector4 operator-(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

Difference of the specified 4-D vectors

## Description

---

Subtract a 4-D vector from another 4-D vector.



---

# operator-

---

Negate all elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector4 operator-();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

4-D vector containing negated elements of the specified 4-D vector

## Description

---

Negate all elements of a 4-D vector.

---

# operator-=

---

Perform compound assignment and subtraction by a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &operator-=(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and subtraction by a 4-D vector.

---

# operator/

---

Divide a 4-D vector by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector4 operator/(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar*    Scalar value

## Return Values

---

Quotient of the specified 4-D vector and scalar

## Description

---

Divide a 4-D vector by a scalar.

---

# operator/=

---

Perform compound assignment and division by a scalar.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &operator/=(
                vec_float4 scalar
            );
        }
    }
}
```

## Arguments

---

*scalar* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Perform compound assignment and division by a scalar.

---

# operator=

---

Assign one 4-D vector to another.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &operator=(
                const Vector4 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 4-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Assign one 4-D vector to another.

---

# operator[]

---

Subscripting operator to set or get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            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](#).

---

# operator[]

---

Subscripting operator to get an element.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 operator[] (
                int idx
            );
        }
    }
}
```

## 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/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            static inline const Vector4 wAxis();
        }
    }
}
```

### Arguments

---

None

### Return Values

---

The constructed 4-D vector

### Description

---

Construct a 4-D vector equal to (0,0,0,1).



---

# xAxis

---

Construct x axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            static inline const Vector4 xAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (1,0,0,0).

---

# yAxis

---

Construct y axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            static inline const Vector4 yAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (0,1,0,0).

---

# zAxis

---

Construct z axis.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            static inline const Vector4 zAxis();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

The constructed 4-D vector

## Description

---

Construct a 4-D vector equal to (0,0,1,0).

---

# Public Instance Methods

## get4Aos

---

Extract four AoS 4-D vectors.

### Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline void get4Aos(
                Aos::Vector4 &result0,
                Aos::Vector4 &result1,
                Aos::Vector4 &result2,
                Aos::Vector4 &result3
            );
        };
    }
}
```

### Arguments

---

<i>result0</i>	An output AoS 4-D vector
<i>result1</i>	An output AoS 4-D vector
<i>result2</i>	An output AoS 4-D vector
<i>result3</i>	An output AoS 4-D vector

### Return Values

---

None

### Description

---

Extract four AoS 4-D vectors from four slots of an SoA 4-D vector (transpose the data format).

---

# getElem

---

Get an x, y, z, or w element of a 4-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 getElem(
                int idx
            );
        }
    }
}
```

## 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 4-D vector by specifying an index of 0, 1, 2, or 3, respectively.

---

# getW

---

Get the w element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 getW();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

w element of a 4-D vector

## Description

---

Get the w element of a 4-D vector.

---

# getX

---

Get the x element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 getX();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

x element of a 4-D vector

## Description

---

Get the x element of a 4-D vector.

---

# getXYZ

---

Get the x, y, and z elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline const Vector3 getXYZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

3-D vector containing x, y, and z elements

## Description

---

Extract a 4-D vector's x, y, and z elements into a 3-D vector.



---

# getY

---

Get the y element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 getY();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

y element of a 4-D vector

## Description

---

Get the y element of a 4-D vector.

---

# getZ

---

Get the z element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline vec_float4 getZ();
        }
    }
}
```

## Arguments

---

None

## Return Values

---

z element of a 4-D vector

## Description

---

Get the z element of a 4-D vector.

---

# setElem

---

Set an x, y, z, or w element of a 4-D vector by index.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setElem(
                int idx,
                vec_float4 value
            );
        };
    }
}
```

## Arguments

---

*idx*      Index, expected in the range 0-3  
*value*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set an x, y, z, or w element of a 4-D vector by specifying an index of 0, 1, 2, or 3, respectively.

---

# setW

---

Set the w element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setW(
                vec_float4 w
            );
        }
    }
}
```

## Arguments

---

*w* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the w element of a 4-D vector to the specified scalar value.

---

# setX

---

Set the x element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setX(
                vec_float4 x
            );
        }
    }
}
```

## Arguments

---

x    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the x element of a 4-D vector to the specified scalar value.

---

# setXYZ

---

Set the x, y, and z elements of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setXYZ(
                const Vector3 &vec
            );
        }
    }
}
```

## Arguments

---

*vec* 3-D vector

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the x, y, and z elements to those of the specified 3-D vector.

## Notes

---

This function does not change the w element.

---

# setY

---

Set the y element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setY(
                vec_float4 y
            );
        }
    }
}
```

## Arguments

---

*y* Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the y element of a 4-D vector to the specified scalar value.

---

# setZ

---

Set the z element of a 4-D vector.

## Definition

---

```
#include <vectormath/cpp/vectormath_soa.h>
namespace Vectormath {
    namespace Soa {
        class Vector4 {
            inline Vector4 &setZ(
                vec_float4 z
            );
        }
    }
}
```

## Arguments

---

*z*    Scalar value

## Return Values

---

A reference to the resulting 4-D vector

## Description

---

Set the z element of a 4-D vector to the specified scalar value.