

Math Helper

1.0

Generated by Doxygen 1.8.3

Thu Jan 3 2013 11:22:54

Contents

1	Namespace Index	1
1.1	Packages	1
2	Class Index	3
2.1	Class List	3
3	Namespace Documentation	5
3.1	Package MathHelper	5
3.1.1	Typedef Documentation	5
3.1.1.1	Plane	5
3.1.1.2	Ray	5
3.1.1.3	Vector3	6
3.1.2	Enumeration Type Documentation	6
3.1.2.1	ContainmentType	6
3.1.2.2	PlaneIntersectionType	6
4	Class Documentation	7
4.1	BoundingBox Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Constructor & Destructor Documentation	7
4.1.2.1	BoundingBox	7
4.1.2.2	BoundingBox	8
4.1.3	Member Function Documentation	8
4.1.3.1	GetCorners	8
4.1.4	Member Data Documentation	8
4.1.4.1	Maximum	8
4.1.4.2	Minimum	8
4.2	BoundingSphere Struct Reference	8
4.2.1	Detailed Description	9
4.2.2	Constructor & Destructor Documentation	9
4.2.2.1	BoundingSphere	9
4.2.2.2	BoundingSphere	9

4.2.3	Member Data Documentation	9
4.2.3.1	Center	9
4.2.3.2	Radius	9
4.3	CollisionHelper Class Reference	9
4.3.1	Detailed Description	12
4.3.2	Member Function Documentation	12
4.3.2.1	BoxContainsBox	12
4.3.2.2	BoxContainsPoint	12
4.3.2.3	BoxContainsSphere	12
4.3.2.4	BoxContainsTriangle	12
4.3.2.5	BoxIntersectsBox	13
4.3.2.6	BoxIntersectsSphere	13
4.3.2.7	BoxIntersectsTriangle	13
4.3.2.8	ClosestPointOnBoxToPoint	14
4.3.2.9	ClosestPointOnPlaneToPoint	14
4.3.2.10	ClosestPointOnSegmentToPoint	14
4.3.2.11	ClosestPointOnSphereToPoint	14
4.3.2.12	ClosestPointOnSphereToSphere	14
4.3.2.13	ClosestPointOnTriangleToPoint	15
4.3.2.14	DistanceBoxBox	15
4.3.2.15	DistanceBoxPoint	15
4.3.2.16	DistancePlanePoint	15
4.3.2.17	DistanceSpherePoint	16
4.3.2.18	DistanceSphereSphere	16
4.3.2.19	PlaneIntersectsBox	16
4.3.2.20	PlaneIntersectsPlane	16
4.3.2.21	PlaneIntersectsPlane	17
4.3.2.22	PlaneIntersectsPoint	17
4.3.2.23	PlaneIntersectsSphere	17
4.3.2.24	PlaneIntersectsTriangle	18
4.3.2.25	RayIntersectsBox	18
4.3.2.26	RayIntersectsBox	18
4.3.2.27	RayIntersectsPlane	18
4.3.2.28	RayIntersectsPlane	19
4.3.2.29	RayIntersectsPoint	19
4.3.2.30	RayIntersectsRay	19
4.3.2.31	RayIntersectsSphere	20
4.3.2.32	RayIntersectsSphere	20
4.3.2.33	RayIntersectsSphere	20
4.3.2.34	RayIntersectsSphere	21

4.3.2.35	RayIntersectsTriangle	21
4.3.2.36	RayIntersectsTriangle	21
4.3.2.37	SphereContainsBox	22
4.3.2.38	SphereContainsPoint	22
4.3.2.39	SphereContainsSphere	22
4.3.2.40	SphereContainsTriangle	23
4.3.2.41	SphereIntersectsSphere	23
4.3.2.42	SphereIntersectsTriangle	23
4.3.2.43	SupportPoint	23
4.3.2.44	SupportPoint	24
4.3.2.45	SupportPoint	24
4.3.2.46	SupportPoint	24
4.4	FloatHelper Class Reference	24
4.4.1	Detailed Description	25
4.4.2	Member Function Documentation	25
4.4.2.1	PackHalf	25
4.4.2.2	UnpackHalf	25
4.5	Vector2Helper Class Reference	25
4.5.1	Detailed Description	28
4.5.2	Member Function Documentation	28
4.5.2.1	Abs	28
4.5.2.2	Abs	28
4.5.2.3	Add	28
4.5.2.4	Add	29
4.5.2.5	Barycentric	29
4.5.2.6	Barycentric	29
4.5.2.7	CatmullRom	29
4.5.2.8	CatmullRom	30
4.5.2.9	Clamp	30
4.5.2.10	Clamp	30
4.5.2.11	Cos	30
4.5.2.12	Cos	31
4.5.2.13	Distance	31
4.5.2.14	Distance	31
4.5.2.15	DistanceSquared	31
4.5.2.16	DistanceSquared	32
4.5.2.17	Divide	32
4.5.2.18	Divide	32
4.5.2.19	Dot	32
4.5.2.20	Dot	33

4.5.2.21	Exp	33
4.5.2.22	Exp	33
4.5.2.23	Hermite	33
4.5.2.24	Hermite	34
4.5.2.25	Lerp	34
4.5.2.26	Lerp	34
4.5.2.27	Max	35
4.5.2.28	Max	35
4.5.2.29	Min	35
4.5.2.30	Min	35
4.5.2.31	Modulate	35
4.5.2.32	Modulate	36
4.5.2.33	Multiply	36
4.5.2.34	Multiply	36
4.5.2.35	Negate	36
4.5.2.36	Negate	37
4.5.2.37	Normalize	37
4.5.2.38	Normalize	37
4.5.2.39	Orthogonalize	37
4.5.2.40	Orthonormalize	38
4.5.2.41	Perp	38
4.5.2.42	Perp	38
4.5.2.43	PerpDot	38
4.5.2.44	PerpDot	39
4.5.2.45	Reciprocal	39
4.5.2.46	Reciprocal	39
4.5.2.47	ReciprocalSqrt	39
4.5.2.48	ReciprocalSqrt	40
4.5.2.49	Reflect	40
4.5.2.50	Reflect	40
4.5.2.51	Refract	40
4.5.2.52	Refract	41
4.5.2.53	Sin	41
4.5.2.54	Sin	41
4.5.2.55	SinCos	41
4.5.2.56	SmoothStep	41
4.5.2.57	SmoothStep	42
4.5.2.58	Sqrt	42
4.5.2.59	Sqrt	42
4.5.2.60	Subtract	42

4.5.2.61	Subtract	43
4.5.2.62	Tan	43
4.5.2.63	Tan	43
4.6	Vector3Helper Class Reference	43
4.6.1	Detailed Description	46
4.6.2	Member Function Documentation	46
4.6.2.1	Abs	46
4.6.2.2	Abs	46
4.6.2.3	Add	47
4.6.2.4	Add	47
4.6.2.5	Barycentric	47
4.6.2.6	Barycentric	47
4.6.2.7	CatmullRom	48
4.6.2.8	CatmullRom	48
4.6.2.9	Clamp	48
4.6.2.10	Clamp	49
4.6.2.11	Cos	49
4.6.2.12	Cos	49
4.6.2.13	Cross	49
4.6.2.14	Cross	49
4.6.2.15	Distance	50
4.6.2.16	Distance	50
4.6.2.17	DistanceSquared	50
4.6.2.18	DistanceSquared	50
4.6.2.19	Divide	51
4.6.2.20	Divide	51
4.6.2.21	Dot	51
4.6.2.22	Dot	51
4.6.2.23	Exp	52
4.6.2.24	Exp	52
4.6.2.25	Hermite	52
4.6.2.26	Hermite	52
4.6.2.27	Lerp	53
4.6.2.28	Lerp	53
4.6.2.29	Max	53
4.6.2.30	Max	54
4.6.2.31	Min	54
4.6.2.32	Min	54
4.6.2.33	Modulate	54
4.6.2.34	Modulate	54

4.6.2.35	Multiply	55
4.6.2.36	Multiply	55
4.6.2.37	Negate	55
4.6.2.38	Negate	55
4.6.2.39	Normalize	56
4.6.2.40	Normalize	56
4.6.2.41	Orthogonalize	56
4.6.2.42	Orthonormalize	56
4.6.2.43	Reciprocal	57
4.6.2.44	Reciprocal	57
4.6.2.45	ReciprocalSqrt	57
4.6.2.46	ReciprocalSqrt	57
4.6.2.47	Reflect	58
4.6.2.48	Reflect	58
4.6.2.49	Refract	58
4.6.2.50	Refract	58
4.6.2.51	Sin	59
4.6.2.52	Sin	59
4.6.2.53	SinCos	59
4.6.2.54	SmoothStep	59
4.6.2.55	SmoothStep	59
4.6.2.56	Sqrt	60
4.6.2.57	Sqrt	60
4.6.2.58	Subtract	60
4.6.2.59	Subtract	60
4.6.2.60	Tan	61
4.6.2.61	Tan	61
4.6.2.62	TripleProduct	61
4.6.2.63	TripleProduct	61
4.7	Vector4Helper Class Reference	61
4.7.1	Detailed Description	64
4.7.2	Member Function Documentation	64
4.7.2.1	Abs	64
4.7.2.2	Abs	64
4.7.2.3	Add	64
4.7.2.4	Add	65
4.7.2.5	Barycentric	65
4.7.2.6	Barycentric	65
4.7.2.7	CatmullRom	65
4.7.2.8	CatmullRom	66

4.7.2.9	Clamp	66
4.7.2.10	Clamp	66
4.7.2.11	Cos	66
4.7.2.12	Cos	67
4.7.2.13	Distance	67
4.7.2.14	Distance	67
4.7.2.15	DistanceSquared	67
4.7.2.16	DistanceSquared	68
4.7.2.17	Divide	68
4.7.2.18	Divide	68
4.7.2.19	Dot	68
4.7.2.20	Dot	69
4.7.2.21	Exp	69
4.7.2.22	Exp	69
4.7.2.23	Hermite	69
4.7.2.24	Hermite	70
4.7.2.25	Lerp	70
4.7.2.26	Lerp	70
4.7.2.27	Max	71
4.7.2.28	Max	71
4.7.2.29	Min	71
4.7.2.30	Min	71
4.7.2.31	Modulate	71
4.7.2.32	Modulate	72
4.7.2.33	Multiply	72
4.7.2.34	Multiply	72
4.7.2.35	Negate	72
4.7.2.36	Negate	73
4.7.2.37	Normalize	73
4.7.2.38	Normalize	73
4.7.2.39	Orthogonalize	73
4.7.2.40	Orthonormalize	74
4.7.2.41	Reciprocal	74
4.7.2.42	Reciprocal	74
4.7.2.43	ReciprocalSqrt	74
4.7.2.44	ReciprocalSqrt	75
4.7.2.45	Sin	75
4.7.2.46	Sin	75
4.7.2.47	SinCos	75
4.7.2.48	SmoothStep	75

4.7.2.49 SmoothStep	76
4.7.2.50 Sqrt	76
4.7.2.51 Sqrt	76
4.7.2.52 Subtract	76
4.7.2.53 Subtract	76
4.7.2.54 Tan	77
4.7.2.55 Tan	77

Index**77**

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

MathHelper	5
-----------------------------	---

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BoundingBox	
Represents an axis-aligned bounding box in three dimensional space.	7
BoundingSphere	
Represents a bounding sphere in three dimensional space.	8
CollisionHelper	
Contains static methods to help in determining intersections, containment, etc.	9
FloatHelper	
Helper class to convert float values	24
Vector2Helper	
Helper methods for UnityEngine.Vector2	25
Vector3Helper	
Helper methods for UnityEngine.Vector3	43
Vector4Helper	
Helper methods for UnityEngine.Vector4	61

Chapter 3

Namespace Documentation

3.1 Package MathHelper

Classes

- struct **BoundingBox**
Represents an axis-aligned bounding box in three dimensional space.
- struct **BoundingSphere**
Represents a bounding sphere in three dimensional space.
- class **CollisionHelper**
Contains static methods to help in determining intersections, containment, etc.
- class **FloatHelper**
Helper class to convert float values
- class **Vector2Helper**
Helper methods for UnityEngine.Vector2
- class **Vector3Helper**
Helper methods for UnityEngine.Vector3
- class **Vector4Helper**
Helper methods for UnityEngine.Vector4

Typedefs

- using **Vector3** = UnityEngine.Vector3
- using **Ray** = UnityEngine.Ray
- using **Plane** = UnityEngine.Plane

Enumerations

- enum **PlaneIntersectionType** { **Back**, **Front**, **Intersecting** }
- enum **ContainmentType** { **Disjoint**, **Contains**, **Intersects** }

3.1.1 Typedef Documentation

3.1.1.1 using **Plane** = UnityEngine.Plane

3.1.1.2 using **Ray** = UnityEngine.Ray

3.1.1.3 `typedef UnityEngine.Vector3 Vector3`

3.1.2 Enumeration Type Documentation

3.1.2.1 `enum ContainmentType`

Enumerator

Disjoint The two bounding volumes don't intersect at all.

Contains One bounding volume completely contains another.

Intersects The two bounding volumes overlap.

3.1.2.2 `enum PlaneIntersectionType`

Enumerator

Back The object is behind the plane.

Front The object is in front of the plane.

Intersecting The object is intersecting the plane.

Chapter 4

Class Documentation

4.1 BoundingBox Struct Reference

Represents an axis-aligned bounding box in three dimensional space.

Public Member Functions

- **BoundingBox** (**Vector3** minimum, **Vector3** maximum)
*Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.*
- **BoundingBox** (float minimumX, float minimumY, float minimumZ, float maximumX, float maximumY, float maximumZ)
*Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.*
- **Vector3[] GetCorners** ()
Retrieves the eight corners of the bounding box.

Public Attributes

- **Vector3 Minimum**
The minimum point of the box.
- **Vector3 Maximum**
The maximum point of the box.

4.1.1 Detailed Description

Represents an axis-aligned bounding box in three dimensional space.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BoundingBox (**Vector3** minimum, **Vector3** maximum)

Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.

Parameters

<i>minimum</i>	The minimum vertex of the bounding box.
<i>maximum</i>	The maximum vertex of the bounding box.

4.1.2.2 BoundingBox (float *minimumX*, float *minimumY*, float *minimumZ*, float *maximumX*, float *maximumY*, float *maximumZ*)

Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.

Parameters

<i>minimumX</i>	The minimum x-coordinate of the bounding box.
<i>minimumY</i>	The minimum y-coordinate of the bounding box.
<i>minimumZ</i>	The minimum z-coordinate of the bounding box.
<i>maximumX</i>	The maximum x-coordinate of the bounding box.
<i>maximumY</i>	The maximum y-coordinate of the bounding box.
<i>maximumZ</i>	The maximum z-coordinate of the bounding box.

4.1.3 Member Function Documentation

4.1.3.1 Vector3 [] GetCorners ()

Retrieves the eight corners of the bounding box.

Returns

An array of points representing the eight corners of the bounding box.

4.1.4 Member Data Documentation

4.1.4.1 Vector3 Maximum

The maximum point of the box.

4.1.4.2 Vector3 Minimum

The minimum point of the box.

4.2 BoundingSphere Struct Reference

Represents a bounding sphere in three dimensional space.

Public Member Functions

- **BoundingSphere** (Vector3 center, float radius)
*Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.*
- **BoundingSphere** (float centerX, float centerY, float centerZ, float radius)
*Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.*

Public Attributes

- **Vector3 Center**
The center of the sphere in three dimensional space.
- float **Radius**
The radius of the sphere.

4.2.1 Detailed Description

Represents a bounding sphere in three dimensional space.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BoundingSphere (Vector3 center, float radius)

Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.

Parameters

<i>center</i>	The center of the sphere.
<i>radius</i>	The radius of the sphere.

4.2.2.2 BoundingSphere (float centerX, float centerY, float centerZ, float radius)

Initializes a new instance of the **MathHelper.BoundingBox** (p. 7) struct.

Parameters

<i>centerX</i>	The x-coordinate for the center of the sphere.
<i>centerY</i>	The y-coordinate for the center of the sphere.
<i>centerZ</i>	The z-coordinate for the center of the sphere.
<i>radius</i>	The radius of the sphere.

4.2.3 Member Data Documentation

4.2.3.1 Vector3 Center

The center of the sphere in three dimensional space.

4.2.3.2 float Radius

The radius of the sphere.

4.3 CollisionHelper Class Reference

Contains static methods to help in determining intersections, containment, etc.

Static Public Member Functions

- static void **ClosestPointOnSegmentToPoint** (ref **Vector3** segment1, ref **Vector3** segment2, ref **Vector3** point, out **Vector3** result)
Determines the closest point between a point and a segment.
- static void **ClosestPointOnPlaneToPoint** (ref **Plane** plane, ref **Vector3** point, out **Vector3** result)
*Determines the closest point between a *UnityEngine.Plane* and a point.*
- static void **ClosestPointOnTriangleToPoint** (ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3, ref **Vector3** point, out **Vector3** result)
Determines the closest point between a point and a triangle.
- static void **ClosestPointOnBoxToPoint** (ref **BoundingBox** box, ref **Vector3** point, out **Vector3** result)

- Determines the closest point between a **MathHelper.BoundingBox** (p. 7) and a point.*
- static void **ClosestPointOnSphereToPoint** (ref **BoundingSphere** sphere, ref **Vector3** point, out **Vector3** result)
- Determines the closest point between a **MathHelper.BoundingSphere** (p. 8) and a point.*
- static void **ClosestPointOnSphereToSphere** (ref **BoundingSphere** sphere1, ref **BoundingSphere** sphere2, out **Vector3** result)
- Determines the closest point between a **MathHelper.BoundingSphere** (p. 8) and a **MathHelper.BoundingSphere** (p. 8).*
- static float **DistancePlanePoint** (ref **Plane** plane, ref **Vector3** point)
- Determines the distance between a **UnityEngine.Plane** and a point.*
- static float **DistanceBoxPoint** (ref **BoundingBox** box, ref **Vector3** point)
- Determines the distance between a **MathHelper.BoundingBox** (p. 7) and a point.*
- static float **DistanceBoxBox** (ref **BoundingBox** box1, ref **BoundingBox** box2)
- Determines the distance between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingBox** (p. 7).*
- static float **DistanceSpherePoint** (ref **BoundingSphere** sphere, ref **Vector3** point)
- Determines the distance between a **MathHelper.BoundingSphere** (p. 8) and a point.*
- static float **DistanceSphereSphere** (ref **BoundingSphere** sphere1, ref **BoundingSphere** sphere2)
- Determines the distance between a **MathHelper.BoundingSphere** (p. 8) and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **RayIntersectsPoint** (ref **Ray** ray, ref **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a point.*
- static bool **RayIntersectsRay** (ref **Ray** ray1, ref **Ray** ray2, out **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **UnityEngine.Ray**.*
- static bool **RayIntersectsPlane** (ref **Ray** ray, ref **Plane** plane, out float distance)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **UnityEngine.Plane**.*
- static bool **RayIntersectsPlane** (ref **Ray** ray, ref **Plane** plane, out **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **UnityEngine.Plane**.*
- static bool **RayIntersectsTriangle** (ref **Ray** ray, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3, out float distance)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a triangle.*
- static bool **RayIntersectsTriangle** (ref **Ray** ray, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3, out **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a triangle.*
- static bool **RayIntersectsBox** (ref **Ray** ray, ref **BoundingBox** box, out float distance)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **MathHelper.BoundingBox** (p. 7).*
- static bool **RayIntersectsBox** (ref **Ray** ray, ref **BoundingBox** box, out **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **UnityEngine.Plane**.*
- static bool **RayIntersectsSphere** (ref **Ray** ray, ref **BoundingSphere** sphere, out float distance)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **RayIntersectsSphere** (ref **Ray** ray, ref **BoundingSphere** sphere, out **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **RayIntersectsSphere** (ref **Ray** ray, ref **BoundingSphere** sphere, out **Vector3** point, out **Vector3** normal)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **RayIntersectsSphere** (ref **Ray** ray, ref **BoundingSphere** sphere, out **Vector3** entrancePoint, out **Vector3** entranceNormal, out **Vector3** exitPoint, out **Vector3** exitNormal)
- Determines whether there is an intersection between a **UnityEngine.Ray** and a **MathHelper.BoundingSphere** (p. 8).*
- static **PlaneIntersectionType** **PlaneIntersectsPoint** (ref **Plane** plane, ref **Vector3** point)
- Determines whether there is an intersection between a **UnityEngine.Plane** and a point.*
- static bool **PlaneIntersectsPlane** (ref **Plane** plane1, ref **Plane** plane2)
- Determines whether there is an intersection between a **UnityEngine.Plane** and a **UnityEngine.Plane**.*
- static bool **PlaneIntersectsPlane** (ref **Plane** plane1, ref **Plane** plane2, out **Ray** line)
- Determines whether there is an intersection between a **UnityEngine.Plane** and a **UnityEngine.Plane**.*

- static **PlaneIntersectionType** **PlaneIntersectsTriangle** (ref **Plane** plane, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3)
*Determines whether there is an intersection between a **UnityEngine.Plane** and a triangle.*
- static **PlaneIntersectionType** **PlaneIntersectsBox** (ref **Plane** plane, ref **BoundingBox** box)
*Determines whether there is an intersection between a **UnityEngine.Plane** and a **MathHelper.BoundingBox** (p. 7).*
- static **PlaneIntersectionType** **PlaneIntersectsSphere** (ref **Plane** plane, ref **BoundingSphere** sphere)
*Determines whether there is an intersection between a **UnityEngine.Plane** and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **BoxIntersectsTriangle** (ref **BoundingBox** box, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3)
*Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a triangle.*
- static bool **BoxIntersectsBox** (ref **BoundingBox** box1, ref **BoundingBox** box2)
*Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingBox** (p. 7).*
- static bool **BoxIntersectsSphere** (ref **BoundingBox** box, ref **BoundingSphere** sphere)
*Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingSphere** (p. 8).*
- static bool **SphereIntersectsTriangle** (ref **BoundingSphere** sphere, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3)
*Determines whether there is an intersection between a **MathHelper.BoundingSphere** (p. 8) and a triangle.*
- static bool **SphereIntersectsSphere** (ref **BoundingSphere** sphere1, ref **BoundingSphere** sphere2)
*Determines whether there is an intersection between a **MathHelper.BoundingSphere** (p. 8) and a **MathHelper.BoundingSphere** (p. 8).*
- static **ContainmentType** **BoxContainsPoint** (ref **BoundingBox** box, ref **Vector3** point)
*Determines whether a **MathHelper.BoundingBox** (p. 7) contains a point.*
- static **ContainmentType** **BoxContainsTriangle** (ref **BoundingBox** box, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3)
*Determines whether a **MathHelper.BoundingBox** (p. 7) contains a triangle.*
- static **ContainmentType** **BoxContainsBox** (ref **BoundingBox** box1, ref **BoundingBox** box2)
*Determines whether a **MathHelper.BoundingBox** (p. 7) contains a **MathHelper.BoundingBox** (p. 7).*
- static **ContainmentType** **BoxContainsSphere** (ref **BoundingBox** box, ref **BoundingSphere** sphere)
*Determines whether a **MathHelper.BoundingBox** (p. 7) contains a **MathHelper.BoundingSphere** (p. 8).*
- static **ContainmentType** **SphereContainsPoint** (ref **BoundingSphere** sphere, ref **Vector3** point)
*Determines whether a **MathHelper.BoundingSphere** (p. 8) contains a point.*
- static **ContainmentType** **SphereContainsTriangle** (ref **BoundingSphere** sphere, ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3)
*Determines whether a **MathHelper.BoundingSphere** (p. 8) contains a triangle.*
- static **ContainmentType** **SphereContainsBox** (ref **BoundingSphere** sphere, ref **BoundingBox** box)
*Determines whether a **MathHelper.BoundingSphere** (p. 8) contains a **MathHelper.BoundingBox** (p. 7).*
- static **ContainmentType** **SphereContainsSphere** (ref **BoundingSphere** sphere1, ref **BoundingSphere** sphere2)
*Determines whether a **MathHelper.BoundingSphere** (p. 8) contains a **MathHelper.BoundingSphere** (p. 8).*
- static void **SupportPoint** (ref **Vector3** vertex1, ref **Vector3** vertex2, ref **Vector3** vertex3, ref **Vector3** direction, out **Vector3** result)
Generates a supporting point from a specific triangle.
- static void **SupportPoint** (ref **BoundingBox** box, ref **Vector3** direction, out **Vector3** result)
*Generates a supporting point from a specific **MathHelper.BoundingBox** (p. 7).*
- static void **SupportPoint** (ref **BoundingSphere** sphere, ref **Vector3** direction, out **Vector3** result)
*Generates a supporting point from a specific **MathHelper.BoundingSphere** (p. 8).*
- static void **SupportPoint** (IEnumerable< **Vector3** > points, ref **Vector3** direction, out **Vector3** result)
Generates a supporting point from a polyhedra.

4.3.1 Detailed Description

Contains static methods to help in determining intersections, containment, etc.

4.3.2 Member Function Documentation

4.3.2.1 static ContainmentType BoxContainsBox (ref BoundingBox box1, ref BoundingBox box2) [static]

Determines whether a **MathHelper.BoundingBox** (p. 7) contains a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>box1</i>	The first box to test.
<i>box2</i>	The second box to test.

Returns

The type of containment the two objects have.

4.3.2.2 static ContainmentType BoxContainsPoint (ref BoundingBox box, ref Vector3 point) [static]

Determines whether a **MathHelper.BoundingBox** (p. 7) contains a point.

Parameters

<i>box</i>	The box to test.
<i>point</i>	The point to test.

Returns

The type of containment the two objects have.

4.3.2.3 static ContainmentType BoxContainsSphere (ref BoundingBox box, ref BoundingSphere sphere) [static]

Determines whether a **MathHelper.BoundingBox** (p. 7) contains a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>box</i>	The box to test.
<i>sphere</i>	The sphere to test.

Returns

The type of containment the two objects have.

4.3.2.4 static ContainmentType BoxContainsTriangle (ref BoundingBox box, ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3) [static]

Determines whether a **MathHelper.BoundingBox** (p. 7) contains a triangle.

Parameters

<i>box</i>	The box to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.

Returns

The type of containment the two objects have.

4.3.2.5 static bool BoxIntersectsBox (ref BoundingBox box1, ref BoundingBox box2) [static]

Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>box1</i>	The first box to test.
<i>box2</i>	The second box to test.

Returns

Whether the two objects intersected.

4.3.2.6 static bool BoxIntersectsSphere (ref BoundingBox box, ref BoundingSphere sphere) [static]

Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>box</i>	The box to test.
<i>sphere</i>	The sphere to test.

Returns

Whether the two objects intersected.

4.3.2.7 static bool BoxIntersectsTriangle (ref BoundingBox box, ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3) [static]

Determines whether there is an intersection between a **MathHelper.BoundingBox** (p. 7) and a triangle.

Parameters

<i>box</i>	The box to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.

Returns

Whether the two objects intersected.

4.3.2.8 static void ClosestPointOnBoxToPoint (ref **BoundingBox** *box*, ref **Vector3** *point*, out **Vector3** *result*)
[static]

Determines the closest point between a **MathHelper.BoundingBox** (p. 7) and a point.

Parameters

<i>box</i>	The box to test.
<i>point</i>	The point to test.
<i>result</i>	When the method completes, contains the closest point between the two objects.

4.3.2.9 static void ClosestPointOnPlaneToPoint (ref **Plane** *plane*, ref **Vector3** *point*, out **Vector3** *result*) [static]

Determines the closest point between a **UnityEngine.Plane** and a point.

Parameters

<i>plane</i>	The plane to test.
<i>point</i>	The point to test.
<i>result</i>	When the method completes, contains the closest point between the two objects.

4.3.2.10 static void ClosestPointOnSegmentToPoint (ref **Vector3** *segment1*, ref **Vector3** *segment2*, ref **Vector3** *point*, out **Vector3** *result*) [static]

Determines the closest point between a point and a segment.

Parameters

<i>point</i>	The point to test.
<i>segment1</i>	The starting point of the segment to test.
<i>segment2</i>	The ending point of the segment to test.
<i>result</i>	When the method completes, contains the closest point between the two objects.

4.3.2.11 static void ClosestPointOnSphereToPoint (ref **BoundingSphere** *sphere*, ref **Vector3** *point*, out **Vector3** *result*)
[static]

Determines the closest point between a **MathHelper.BoundingSphere** (p. 8) and a point.

Parameters

<i>sphere</i>	
<i>point</i>	The point to test.
<i>result</i>	When the method completes, contains the closest point between the two objects; or, if the point is directly in the center of the sphere, contains UnityEngine.Vector3.zero .

4.3.2.12 static void ClosestPointOnSphereToSphere (ref **BoundingSphere** *sphere1*, ref **BoundingSphere** *sphere2*, out **Vector3** *result*) [static]

Determines the closest point between a **MathHelper.BoundingSphere** (p. 8) and a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>sphere1</i>	The first sphere to test.
<i>sphere2</i>	The second sphere to test.
<i>result</i>	When the method completes, contains the closest point between the two objects; or, if the point is directly in the center of the sphere, contains <code>UnityEngine.Vector3.zero</code> .

If the two spheres are overlapping, but not directly ontop of each other, the closest point is the 'closest' point of intersection. This can also be considered is the deepest point of intersection.

4.3.2.13 `static void ClosestPointOnTriangleToPoint (ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3, ref Vector3 point, out Vector3 result) [static]`

Determines the closest point between a point and a triangle.

Parameters

<i>point</i>	The point to test.
<i>vertex1</i>	The first vertex to test.
<i>vertex2</i>	The second vertex to test.
<i>vertex3</i>	The third vertex to test.
<i>result</i>	When the method completes, contains the closest point between the two objects.

4.3.2.14 `static float DistanceBoxBox (ref BoundingBox box1, ref BoundingBox box2) [static]`

Determines the distance between a **MathHelper.BoundingBox** (p. 7) and a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>box1</i>	The first box to test.
<i>box2</i>	The second box to test.

Returns

The distance between the two objects.

4.3.2.15 `static float DistanceBoxPoint (ref BoundingBox box, ref Vector3 point) [static]`

Determines the distance between a **MathHelper.BoundingBox** (p. 7) and a point.

Parameters

<i>box</i>	The box to test.
<i>point</i>	The point to test.

Returns

The distance between the two objects.

4.3.2.16 `static float DistancePlanePoint (ref Plane plane, ref Vector3 point) [static]`

Determines the distance between a `UnityEngine.Plane` and a point.

Parameters

<i>plane</i>	The plane to test.
<i>point</i>	The point to test.

Returns

The distance between the two objects.

4.3.2.17 `static float DistanceSpherePoint (ref BoundingSphere sphere, ref Vector3 point) [static]`

Determines the distance between a **MathHelper.BoundingSphere** (p. 8) and a point.

Parameters

<i>sphere</i>	The sphere to test.
<i>point</i>	The point to test.

Returns

The distance between the two objects.

4.3.2.18 `static float DistanceSphereSphere (ref BoundingSphere sphere1, ref BoundingSphere sphere2) [static]`

Determines the distance between a **MathHelper.BoundingSphere** (p. 8) and a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>sphere1</i>	The first sphere to test.
<i>sphere2</i>	The second sphere to test.

Returns

The distance between the two objects.

4.3.2.19 `static PlaneIntersectionType PlaneIntersectsBox (ref Plane plane, ref BoundingBox box) [static]`

Determines whether there is an intersection between a UnityEngine.Plane and a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>plane</i>	The plane to test.
<i>box</i>	The box to test.

Returns

Whether the two objects intersected.

4.3.2.20 `static bool PlaneIntersectsPlane (ref Plane plane1, ref Plane plane2) [static]`

Determines whether there is an intersection between a UnityEngine.Plane and a UnityEngine.Plane.

Parameters

<i>plane1</i>	The first plane to test.
<i>plane2</i>	The second plane to test.

Returns

Whether the two objects intersected.

4.3.2.21 static bool PlaneIntersectsPlane (ref Plane *plane1*, ref Plane *plane2*, out Ray *line*) [static]

Determines whether there is an intersection between a UnityEngine.Plane and a UnityEngine.Plane.

Parameters

<i>plane1</i>	The first plane to test.
<i>plane2</i>	The second plane to test.
<i>line</i>	When the method completes, contains the line of intersection as a UnityEngine.Ray, or a zero ray if there was no intersection.

Returns

Whether the two objects intersected.

Although a ray is set to have an origin, the ray returned by this method is really a line in three dimensions which has no real origin. The ray is considered valid when both the positive direction is used and when the negative direction is used.

4.3.2.22 static PlaneIntersectionType PlaneIntersectsPoint (ref Plane *plane*, ref Vector3 *point*) [static]

Determines whether there is an intersection between a UnityEngine.Plane and a point.

Parameters

<i>plane</i>	The plane to test.
<i>point</i>	The point to test.

Returns

Whether the two objects intersected.

4.3.2.23 static PlaneIntersectionType PlaneIntersectsSphere (ref Plane *plane*, ref BoundingSphere *sphere*) [static]

Determines whether there is an intersection between a UnityEngine.Plane and a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>plane</i>	The plane to test.
<i>sphere</i>	The sphere to test.

Returns

Whether the two objects intersected.

4.3.2.24 `static PlaneIntersectionType PlaneIntersectsTriangle (ref Plane plane, ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3)` [static]

Determines whether there is an intersection between a UnityEngine.Plane and a triangle.

Parameters

<i>plane</i>	The plane to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.

Returns

Whether the two objects intersected.

4.3.2.25 `static bool RayIntersectsBox (ref Ray ray, ref BoundingBox box, out float distance)` [static]

Determines whether there is an intersection between a UnityEngine.Ray and a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>ray</i>	The ray to test.
<i>box</i>	The box to test.
<i>distance</i>	When the method completes, contains the distance of the intersection, or 0 if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.26 `static bool RayIntersectsBox (ref Ray ray, ref BoundingBox box, out Vector3 point)` [static]

Determines whether there is an intersection between a UnityEngine.Ray and a UnityEngine.Plane.

Parameters

<i>ray</i>	The ray to test.
<i>box</i>	The box to test.
<i>point</i>	When the method completes, contains the point of intersection, or UnityEngine.Vector3.zero if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.27 `static bool RayIntersectsPlane (ref Ray ray, ref Plane plane, out float distance)` [static]

Determines whether there is an intersection between a UnityEngine.Ray and a UnityEngine.Plane.

Parameters

<i>ray</i>	The ray to test.
<i>plane</i>	The plane to test.
<i>distance</i>	When the method completes, contains the distance of the intersection, or 0 if there was no intersection.

Returns

Whether the two objects intersect.

4.3.2.28 `static bool RayIntersectsPlane (ref Ray ray, ref Plane plane, out Vector3 point) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a UnityEngine.Plane.

Parameters

<i>ray</i>	The ray to test.
<i>plane</i>	The plane to test
<i>point</i>	When the method completes, contains the point of intersection, or UnityEngine.Vector3.zero if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.29 `static bool RayIntersectsPoint (ref Ray ray, ref Vector3 point) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a point.

Parameters

<i>ray</i>	The ray to test.
<i>point</i>	The point to test.

Returns

Whether the two objects intersect.

4.3.2.30 `static bool RayIntersectsRay (ref Ray ray1, ref Ray ray2, out Vector3 point) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a UnityEngine.Ray.

Parameters

<i>ray1</i>	The first ray to test.
<i>ray2</i>	The second ray to test.
<i>point</i>	When the method completes, contains the point of intersection, or UnityEngine.Vector3.zero if there was no intersection.

Returns

Whether the two objects intersect.

4.3.2.31 static bool RayIntersectsSphere (ref Ray *ray*, ref BoundingBoxSphere *sphere*, out float *distance*) [static]

Determines whether there is an intersection between a UnityEngine.Ray and a **MathHelper.BoundingBoxSphere** (p. 8).

Parameters

<i>ray</i>	The ray to test.
<i>sphere</i>	The sphere to test.
<i>distance</i>	When the method completes, contains the distance of the intersection, or 0 if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.32 static bool RayIntersectsSphere (ref Ray *ray*, ref BoundingBoxSphere *sphere*, out Vector3 *point*) [static]

Determines whether there is an intersection between a UnityEngine.Ray and a **MathHelper.BoundingBoxSphere** (p. 8).

Parameters

<i>ray</i>	The ray to test.
<i>sphere</i>	The sphere to test.
<i>point</i>	When the method completes, contains the point of intersection, or UnityEngine.Vector3.zero if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.33 static bool RayIntersectsSphere (ref Ray *ray*, ref BoundingBoxSphere *sphere*, out Vector3 *point*, out Vector3 *normal*) [static]

Determines whether there is an intersection between a UnityEngine.Ray and a **MathHelper.BoundingBoxSphere** (p. 8).

Parameters

<i>ray</i>	The ray to test.
<i>sphere</i>	The sphere to test.
<i>point</i>	When the method completes, contains the point of intersection, or UnityEngine.Vector3.zero if there was no intersection.
<i>normal</i>	When the method completes, contains the normal vector on the sphere at the point of intersection.

Returns

Whether the two objects intersected.

4.3.2.34 `static bool RayIntersectsSphere (ref Ray ray, ref BoundingSphere sphere, out Vector3 entrancePoint, out Vector3 entranceNormal, out Vector3 exitPoint, out Vector3 exitNormal) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>ray</i>	The ray to test.
<i>sphere</i>	The sphere to test.
<i>entrancePoint</i>	When the method completes, contains the closest point of intersection, or UnityEngine.Vector3.zero if there was no intersection.
<i>entranceNormal</i>	When the method completes, contains the normal vector on the sphere at the point of closest intersection.
<i>exitPoint</i>	When the method completes, contains the farthest point of intersection, or UnityEngine.Vector3.zero if there was no intersection.
<i>exitNormal</i>	When the method completes, contains the normal vector on the sphere at the point of farthest intersection.

Returns

Whether the two objects intersected.

4.3.2.35 `static bool RayIntersectsTriangle (ref Ray ray, ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3, out float distance) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a triangle.

Parameters

<i>ray</i>	The ray to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.
<i>distance</i>	When the method completes, contains the distance of the intersection, or 0 if there was no intersection.

Returns

Whether the two objects intersected.

This method tests if the ray intersects either the front or back of the triangle. If the ray is parallel to the triangle's plane, no intersection is assumed to have happened. If the intersection of the ray and the triangle is behind the origin of the ray, no intersection is assumed to have happened. In both cases of assumptions, this method returns false.

4.3.2.36 `static bool RayIntersectsTriangle (ref Ray ray, ref Vector3 vertex1, ref Vector3 vertex2, ref Vector3 vertex3, out Vector3 point) [static]`

Determines whether there is an intersection between a UnityEngine.Ray and a triangle.

Parameters

<i>ray</i>	The ray to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.
<i>point</i>	When the method completes, contains the point of intersection, or <code>UnityEngine.Vector3.zero</code> if there was no intersection.

Returns

Whether the two objects intersected.

4.3.2.37 static ContainmentType SphereContainsBox (ref BoundingBox sphere, ref BoundingBox box)
[static]

Determines whether a **MathHelper.BoundingBox** (p. 8) contains a **MathHelper.BoundingBox** (p. 7).

Parameters

<i>sphere</i>	The sphere to test.
<i>box</i>	The box to test.

Returns

The type of containment the two objects have.

4.3.2.38 static ContainmentType SphereContainsPoint (ref BoundingBox sphere, ref Vector3 point)
[static]

Determines whether a **MathHelper.BoundingBox** (p. 8) contains a point.

Parameters

<i>sphere</i>	The sphere to test.
<i>point</i>	The point to test.

Returns

The type of containment the two objects have.

4.3.2.39 static ContainmentType SphereContainsSphere (ref BoundingBox sphere1, ref BoundingBox sphere2) [static]

Determines whether a **MathHelper.BoundingBox** (p. 8) contains a **MathHelper.BoundingBox** (p. 8).

Parameters

<i>sphere1</i>	The first sphere to test.
<i>sphere2</i>	The second sphere to test.

Returns

The type of containment the two objects have.

4.3.2.40 **static** **ContainmentType** SphereContainsTriangle (**ref** **BoundingBoxSphere** *sphere*, **ref** **Vector3** *vertex1*, **ref** **Vector3** *vertex2*, **ref** **Vector3** *vertex3*) *[static]*

Determines whether a **MathHelper.BoundingBoxSphere** (p. 8) contains a triangle.

Parameters

<i>sphere</i>	The sphere to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.

Returns

The type of containment the two objects have.

4.3.2.41 **static** **bool** SphereIntersectsSphere (**ref** **BoundingBoxSphere** *sphere1*, **ref** **BoundingBoxSphere** *sphere2*) *[static]*

Determines whether there is an intersection between a **MathHelper.BoundingBoxSphere** (p. 8) and a **MathHelper.BoundingBoxSphere** (p. 8).

Parameters

<i>sphere1</i>	First sphere to test.
<i>sphere2</i>	Second sphere to test.

Returns

Whether the two objects intersected.

4.3.2.42 **static** **bool** SphereIntersectsTriangle (**ref** **BoundingBoxSphere** *sphere*, **ref** **Vector3** *vertex1*, **ref** **Vector3** *vertex2*, **ref** **Vector3** *vertex3*) *[static]*

Determines whether there is an intersection between a **MathHelper.BoundingBoxSphere** (p. 8) and a triangle.

Parameters

<i>sphere</i>	The sphere to test.
<i>vertex1</i>	The first vertex of the triangle to test.
<i>vertex2</i>	The second vertex of the triangle to test.
<i>vertex3</i>	The third vertex of the triangle to test.

Returns

Whether the two objects intersected.

4.3.2.43 **static** **void** SupportPoint (**ref** **Vector3** *vertex1*, **ref** **Vector3** *vertex2*, **ref** **Vector3** *vertex3*, **ref** **Vector3** *direction*, **out** **Vector3** *result*) *[static]*

Generates a supporting point from a specific triangle.

Parameters

<i>vertex1</i>	The first vertex of the triangle.
<i>vertex2</i>	The second vertex of the triangle.
<i>vertex3</i>	The third vertex of the triangle
<i>direction</i>	The direction for which to build the supporting point.
<i>result</i>	When the method completes, contains the supporting point.

4.3.2.44 `static void SupportPoint (ref BoundingBox box, ref Vector3 direction, out Vector3 result)` [static]

Generates a supporting point from a specific **MathHelper.BoundingBox** (p. 7).

Parameters

<i>box</i>	The box to generate the supporting point for.
<i>direction</i>	The direction for which to build the supporting point.
<i>result</i>	When the method completes, contains the supporting point.

4.3.2.45 `static void SupportPoint (ref BoundingSphere sphere, ref Vector3 direction, out Vector3 result)`
[static]

Generates a supporting point from a specific **MathHelper.BoundingSphere** (p. 8).

Parameters

<i>sphere</i>	The sphere to generate the supporting point for.
<i>direction</i>	The direction for which to build the supporting point.
<i>result</i>	When the method completes, contains the supporting point.

4.3.2.46 `static void SupportPoint (IEnumerable< Vector3 > points, ref Vector3 direction, out Vector3 result)`
[static]

Generates a supporting point from a polyhedra.

Parameters

<i>points</i>	The points that make up the polyhedra.
<i>direction</i>	The direction for which to build the supporting point.
<i>result</i>	When the method completes, contains the supporting point.

4.4 FloatHelper Class Reference

Helper class to convert float values

Static Public Member Functions

- static float **UnpackHalf** (ushort value)

Unpacks the specified half.

- static ushort **PackHalf** (float value)

Packs the specified float.

4.4.1 Detailed Description

Helper class to convert float values

4.4.2 Member Function Documentation

4.4.2.1 static ushort PackHalf (float *value*) [static]

Packs the specified float.

Parameters

<i>value</i>	The float to pack.
--------------	--------------------

Returns

The half representation of the float.

4.4.2.2 static float UnpackHalf (ushort *value*) [static]

Unpacks the specified half.

Parameters

<i>value</i>	The half to unpack.
--------------	---------------------

Returns

The floating point representation of the half.

4.5 Vector2Helper Class Reference

Helper methods for UnityEngine.Vector2

Static Public Member Functions

- static void **Sqrt** (ref Vector2 value, out Vector2 result)
Takes the square root of each component in the vector.
- static Vector2 **Sqrt** (Vector2 value)
Takes the square root of each component in the vector.
- static void **Reciprocal** (ref Vector2 value, out Vector2 result)
Takes the reciprocal of each component in the vector.
- static Vector2 **Reciprocal** (Vector2 value)
Takes the reciprocal of each component in the vector.
- static void **ReciprocalSqrt** (ref Vector2 value, out Vector2 result)
Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.
- static Vector2 **ReciprocalSqrt** (Vector2 value)
Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.
- static void **Exp** (ref Vector2 value, out Vector2 result)
Takes e raised to the component in the vector.
- static Vector2 **Exp** (Vector2 value)

- Takes e raised to the component in the vector.*

 - static void **SinCos** (ref Vector2 value, out Vector2 sinResult, out Vector2 cosResult)

Takes the sine and then the cosine of each component in the vector.
- static void **Sin** (ref Vector2 value, out Vector2 result)

Takes the sine of each component in the vector.
- static Vector2 **Sin** (Vector2 value)

Takes the sine of each component in the vector.
- static void **Cos** (ref Vector2 value, out Vector2 result)

Takes the cosine of each component in the vector.
- static Vector2 **Cos** (Vector2 value)

Takes the cosine of each component in the vector.
- static void **Tan** (ref Vector2 value, out Vector2 result)

Takes the tangent of each component in the vector.
- static Vector2 **Tan** (Vector2 value)

Takes the tangent of each component in the vector.
- static void **Add** (ref Vector2 left, ref Vector2 right, out Vector2 result)

Adds two vectors.
- static Vector2 **Add** (Vector2 left, Vector2 right)

Adds two vectors.
- static void **Subtract** (ref Vector2 left, ref Vector2 right, out Vector2 result)

Subtracts two vectors.
- static Vector2 **Subtract** (Vector2 left, Vector2 right)

Subtracts two vectors.
- static void **Multiply** (ref Vector2 value, float scalar, out Vector2 result)

Scales a vector by the given value.
- static Vector2 **Multiply** (Vector2 value, float scalar)

Scales a vector by the given value.
- static void **Modulate** (ref Vector2 left, ref Vector2 right, out Vector2 result)

Modulates a vector with another by performing component-wise multiplication.
- static Vector2 **Modulate** (Vector2 left, Vector2 right)

Modulates a vector with another by performing component-wise multiplication.
- static void **Divide** (ref Vector2 value, float scalar, out Vector2 result)

Scales a vector by the given value.
- static Vector2 **Divide** (Vector2 value, float scalar)

Scales a vector by the given value.
- static void **Negate** (ref Vector2 value, out Vector2 result)

Reverses the direction of a given vector.
- static Vector2 **Negate** (Vector2 value)

Reverses the direction of a given vector.
- static void **Abs** (ref Vector2 value, out Vector2 result)

Takes the absolute value of each component.
- static Vector2 **Abs** (Vector2 value)

Takes the absolute value of each component.
- static void **Barycentric** (ref Vector2 value1, ref Vector2 value2, ref Vector2 value3, float amount1, float amount2, out Vector2 result)

Returns a UnityEngine.Vector2 containing the 2D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 2D triangle.
- static Vector2 **Barycentric** (Vector2 value1, Vector2 value2, Vector2 value3, float amount1, float amount2)

Returns a UnityEngine.Vector2 containing the 2D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 2D triangle.
- static void **Clamp** (ref Vector2 value, ref Vector2 min, ref Vector2 max, out Vector2 result)

- Restricts a value to be within a specified range.*
- static Vector2 **Clamp** (Vector2 value, Vector2 min, Vector2 max)
 - Restricts a value to be within a specified range.*
- static void **Distance** (ref Vector2 value1, ref Vector2 value2, out float result)
 - Calculates the distance between two vectors.*
- static float **Distance** (Vector2 value1, Vector2 value2)
 - Calculates the distance between two vectors.*
- static void **DistanceSquared** (ref Vector2 value1, ref Vector2 value2, out float result)
 - Calculates the squared distance between two vectors.*
- static float **DistanceSquared** (Vector2 value1, Vector2 value2)
 - Calculates the squared distance between two vectors.*
- static void **Dot** (ref Vector2 left, ref Vector2 right, out float result)
 - Calculates the dot product of two vectors.*
- static float **Dot** (Vector2 left, Vector2 right)
 - Calculates the dot product of two vectors.*
- static void **Perp** (ref Vector2 value, out Vector2 result)
 - Calculates a vector that is perpendicular to the given vector.*
- static Vector2 **Perp** (Vector2 value)
 - Calculates a vector that is perpendicular to the given vector.*
- static void **PerpDot** (ref Vector2 left, ref Vector2 right, out float result)
 - Calculates the perp dot product.*
- static float **PerpDot** (Vector2 left, Vector2 right)
 - Calculates the perp dot product.*
- static void **Normalize** (ref Vector2 value, out Vector2 result)
 - Converts the vector into a unit vector.*
- static Vector2 **Normalize** (Vector2 value)
 - Converts the vector into a unit vector.*
- static void **Lerp** (ref Vector2 start, ref Vector2 end, float amount, out Vector2 result)
 - Performs a linear interpolation between two vectors.*
- static Vector2 **Lerp** (Vector2 start, Vector2 end, float amount)
 - Performs a linear interpolation between two vectors.*
- static void **SmoothStep** (ref Vector2 start, ref Vector2 end, float amount, out Vector2 result)
 - Performs a cubic interpolation between two vectors.*
- static Vector2 **SmoothStep** (Vector2 start, Vector2 end, float amount)
 - Performs a cubic interpolation between two vectors.*
- static void **Hermite** (ref Vector2 value1, ref Vector2 tangent1, ref Vector2 value2, ref Vector2 tangent2, float amount, out Vector2 result)
 - Performs a Hermite spline interpolation.*
- static Vector2 **Hermite** (Vector2 value1, Vector2 tangent1, Vector2 value2, Vector2 tangent2, float amount)
 - Performs a Hermite spline interpolation.*
- static void **CatmullRom** (ref Vector2 value1, ref Vector2 value2, ref Vector2 value3, ref Vector2 value4, float amount, out Vector2 result)
 - Performs a Catmull-Rom interpolation using the specified positions.*
- static Vector2 **CatmullRom** (Vector2 value1, Vector2 value2, Vector2 value3, Vector2 value4, float amount)
 - Performs a Catmull-Rom interpolation using the specified positions.*
- static void **Max** (ref Vector2 value1, ref Vector2 value2, out Vector2 result)
 - Returns a vector containing the largest components of the specified vectors.*
- static Vector2 **Max** (Vector2 value1, Vector2 value2)
 - Returns a vector containing the largest components of the specified vectors.*
- static void **Min** (ref Vector2 value1, ref Vector2 value2, out Vector2 result)
 - Returns a vector containing the smallest components of the specified vectors.*

- static Vector2 **Min** (Vector2 value1, Vector2 value2)
Returns a vector containing the smallest components of the specified vectors.
- static void **Reflect** (ref Vector2 vector, ref Vector2 normal, out Vector2 result)
Returns the reflection of a vector off a surface that has the specified normal.
- static Vector2 **Reflect** (Vector2 vector, Vector2 normal)
Returns the reflection of a vector off a surface that has the specified normal.
- static void **Refract** (ref Vector2 vector, ref Vector2 normal, float index, out Vector2 result)
Returns the fraction of a vector off a surface that has the specified normal and index.
- static Vector2 **Refract** (Vector2 vector, Vector2 normal, float index)
Returns the fraction of a vector off a surface that has the specified normal and index.
- static void **Orthogonalize** (Vector2[] destination, params Vector2[] source)
Orthogonalizes a list of vectors.
- static void **Orthonormalize** (Vector2[] destination, params Vector2[] source)
Orthonormalizes a list of vectors.

4.5.1 Detailed Description

Helper methods for UnityEngine.Vector2

4.5.2 Member Function Documentation

4.5.2.1 static void Abs (ref Vector2 value, out Vector2 result) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
<i>result</i>	When the method completes, contains a vector that has all positive components.

4.5.2.2 static Vector2 Abs (Vector2 value) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
--------------	---

Returns

A vector that has all positive components.

4.5.2.3 static void Add (ref Vector2 left, ref Vector2 right, out Vector2 result) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.
<i>result</i>	When the method completes, contains the sum of the two vectors.

4.5.2.4 static Vector2 Add (Vector2 left, Vector2 right) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.

Returns

The sum of the two vectors.

4.5.2.5 static void Barycentric (ref Vector2 value1, ref Vector2 value2, ref Vector2 value3, float amount1, float amount2, out Vector2 result) [static]

Returns a UnityEngine.Vector2 containing the 2D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 2D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 2 of the triangle.
<i>value3</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).
<i>result</i>	When the method completes, contains the 2D Cartesian coordinates of the specified point.

4.5.2.6 static Vector2 Barycentric (Vector2 value1, Vector2 value2, Vector2 value3, float amount1, float amount2) [static]

Returns a UnityEngine.Vector2 containing the 2D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 2D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 2 of the triangle.
<i>value3</i>	A UnityEngine.Vector2 containing the 2D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).

Returns

A new UnityEngine.Vector2 containing the 2D Cartesian coordinates of the specified point.

4.5.2.7 static void CatmullRom (ref Vector2 value1, ref Vector2 value2, ref Vector2 value3, ref Vector2 value4, float amount, out Vector2 result) [static]

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Catmull-Rom interpolation.

4.5.2.8 `static Vector2 CatmullRom (Vector2 value1, Vector2 value2, Vector2 value3, Vector2 value4, float amount)` `[static]`

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.

Returns

A vector that is the result of the Catmull-Rom interpolation.

4.5.2.9 `static void Clamp (ref Vector2 value, ref Vector2 min, ref Vector2 max, out Vector2 result)` `[static]`

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.
<i>result</i>	When the method completes, contains the clamped value.

4.5.2.10 `static Vector2 Clamp (Vector2 value, Vector2 min, Vector2 max)` `[static]`

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.

Returns

The clamped value.

4.5.2.11 `static void Cos (ref Vector2 value, out Vector2 result)` `[static]`

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
<i>result</i>	When the method completes, contains a vector that contains the cosine of each component in the input vector.

4.5.2.12 static Vector2 Cos (Vector2 *value*) [static]

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
--------------	-----------------------------------

Returns

A vector that contains the cosine of each component in the input vector.

4.5.2.13 static void Distance (ref Vector2 *value1*, ref Vector2 *value2*, out float *result*) [static]

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.
<i>result</i>	When the method completes, contains the distance between the two vectors.

UnityEngine.Vector2.DistanceSquared(ref Vector2, ref Vector2, out float) may be preferred when only the relative distance is needed and speed is of the essence.

4.5.2.14 static float Distance (Vector2 *value1*, Vector2 *value2*) [static]

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The distance between the two vectors.

UnityEngine.Vector2.DistanceSquared(Vector2, Vector2) may be preferred when only the relative distance is needed and speed is of the essence.

4.5.2.15 static void DistanceSquared (ref Vector2 *value1*, ref Vector2 *value2*, out float *result*) [static]

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector
<i>result</i>	When the method completes, contains the squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.5.2.16 static float DistanceSquared (Vector2 *value1*, Vector2 *value2*) [static]

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.5.2.17 static void Divide (ref Vector2 *value*, float *scalar*, out Vector2 *result*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.5.2.18 static Vector2 Divide (Vector2 *value*, float *scalar*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.5.2.19 static void Dot (ref Vector2 *left*, ref Vector2 *right*, out float *result*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.
<i>result</i>	When the method completes, contains the dot product of the two vectors.

4.5.2.20 static float Dot (Vector2 *left*, Vector2 *right*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.

Returns

The dot product of the two vectors.

4.5.2.21 static void Exp (ref Vector2 *value*, out Vector2 *result*) [static]

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
<i>result</i>	When the method completes, contains a vector that has e raised to each of the components in the input vector.

4.5.2.22 static Vector2 Exp (Vector2 *value*) [static]

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
--------------	--

Returns

A vector that has e raised to each of the components in the input vector.

4.5.2.23 static void Hermite (ref Vector2 *value1*, ref Vector2 *tangent1*, ref Vector2 *value2*, ref Vector2 *tangent2*, float *amount*, out Vector2 *result*) [static]

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.
<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Hermite spline interpolation.

4.5.2.24 `static Vector2 Hermite (Vector2 value1, Vector2 tangent1, Vector2 value2, Vector2 tangent2, float amount)`
`[static]`

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.
<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.

Returns

The result of the Hermite spline interpolation.

4.5.2.25 `static void Lerp (ref Vector2 start, ref Vector2 end, float amount, out Vector2 result)` `[static]`

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .
<i>result</i>	When the method completes, contains the linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.5.2.26 `static Vector2 Lerp (Vector2 start, Vector2 end, float amount)` `[static]`

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.5.2.27 `static void Max (ref Vector2 value1, ref Vector2 value2, out Vector2 result) [static]`

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the largest components of the source vectors.

4.5.2.28 `static Vector2 Max (Vector2 value1, Vector2 value2) [static]`

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the largest components of the source vectors.

4.5.2.29 `static void Min (ref Vector2 value1, ref Vector2 value2, out Vector2 result) [static]`

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the smallest components of the source vectors.

4.5.2.30 `static Vector2 Min (Vector2 value1, Vector2 value2) [static]`

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the smallest components of the source vectors.

4.5.2.31 `static void Modulate (ref Vector2 left, ref Vector2 right, out Vector2 result) [static]`

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.
<i>result</i>	When the method completes, contains the modulated vector.

4.5.2.32 static Vector2 Modulate (Vector2 left, Vector2 right) [static]

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.

Returns

The modulated vector.

4.5.2.33 static void Multiply (ref Vector2 value, float scalar, out Vector2 result) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.5.2.34 static Vector2 Multiply (Vector2 value, float scalar) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.5.2.35 static void Negate (ref Vector2 value, out Vector2 result) [static]

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
<i>result</i>	When the method completes, contains a vector facing in the opposite direction.

4.5.2.36 static Vector2 Negate (Vector2 value) [static]

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
--------------	-----------------------

Returns

A vector facing in the opposite direction.

4.5.2.37 static void Normalize (ref Vector2 value, out Vector2 result) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
<i>result</i>	When the method completes, contains the normalized vector.

4.5.2.38 static Vector2 Normalize (Vector2 value) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
--------------	--------------------------

Returns

The normalized vector.

4.5.2.39 static void Orthogonalize (Vector2[] destination, params Vector2[] source) [static]

Orthogonalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthogonalized vectors.
<i>source</i>	The list of vectors to orthogonalize.

Orthogonalization is the process of making all vectors orthogonal to each other. This means that any given vector in the list will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.5.2.40 static void Orthonormalize (Vector2[] *destination*, params Vector2[] *source*) [static]

Orthonormalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthonormalized vectors.
<i>source</i>	The list of vectors to orthonormalize.

Orthonormalization is the process of making all vectors orthogonal to each other and making all vectors of unit length. This means that any given vector will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.5.2.41 static void Perp (ref Vector2 *value*, out Vector2 *result*) [static]

Calculates a vector that is perpendicular to the given vector.

Parameters

<i>value</i>	The vector to base the perpendicular vector on.
<i>result</i>	When the method completes, contains the perpendicular vector.

This method finds the perpendicular vector using a 90 degree counterclockwise rotation.

4.5.2.42 static Vector2 Perp (Vector2 *value*) [static]

Calculates a vector that is perpendicular to the given vector.

Parameters

<i>value</i>	The vector to base the perpendicular vector on.
--------------	---

Returns

The perpendicular vector.

This method finds the perpendicular vector using a 90 degree counterclockwise rotation.

4.5.2.43 static void PerpDot (ref Vector2 *left*, ref Vector2 *right*, out float *result*) [static]

Calculates the perp dot product.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.
<i>result</i>	When the method completes, contains the perp dot product of the two vectors.

The perp dot product is defined as taking the dot product of the perpendicular vector of the left vector with the right vector.

4.5.2.44 static float PerpDot (Vector2 *left*, Vector2 *right*) [static]

Calculates the perp dot product.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.

Returns

The perp dot product of the two vectors.

The perp dot product is defined as taking the dot product of the perpendicular vector of the left vector with the right vector.

4.5.2.45 static void Reciprocal (ref Vector2 *value*, out Vector2 *result*) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the reciprocal of the input vector.

4.5.2.46 static Vector2 Reciprocal (Vector2 *value*) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
--------------	---------------------------------------

Returns

A vector that is the reciprocal of the input vector.

4.5.2.47 static void ReciprocalSqrt (ref Vector2 *value*, out Vector2 *result*) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the square root and reciprocal of the input vector.

4.5.2.48 static Vector2 ReciprocalSqrt (Vector2 value) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
--------------	---

Returns

A vector that is the square root and reciprocal of the input vector.

4.5.2.49 static void Reflect (ref Vector2 vector, ref Vector2 normal, out Vector2 result) [static]

Returns the reflection of a vector off a surface that has the specified normal.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>result</i>	When the method completes, contains the reflected vector.

Reflect only gives the direction of a reflection off a surface, it does not determine whether the original vector was close enough to the surface to hit it.

4.5.2.50 static Vector2 Reflect (Vector2 vector, Vector2 normal) [static]

Returns the reflection of a vector off a surface that has the specified normal.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.

Returns

The reflected vector.

Reflect only gives the direction of a reflection off a surface, it does not determine whether the original vector was close enough to the surface to hit it.

4.5.2.51 static void Refract (ref Vector2 vector, ref Vector2 normal, float index, out Vector2 result) [static]

Returns the fraction of a vector off a surface that has the specified normal and index.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>index</i>	Index of refraction.
<i>result</i>	When the method completes, contains the refracted vector.

4.5.2.52 static Vector2 Refract (Vector2 *vector*, Vector2 *normal*, float *index*) [static]

Returns the fraction of a vector off a surface that has the specified normal and index.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>index</i>	Index of refraction.

Returns

The refracted vector.

4.5.2.53 static void Sin (ref Vector2 *value*, out Vector2 *result*) [static]

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
<i>result</i>	When the method completes, a vector that contains the sine of each component in the input vector.

4.5.2.54 static Vector2 Sin (Vector2 *value*) [static]

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
--------------	---------------------------------

Returns

A vector that contains the sine of each component in the input vector.

4.5.2.55 static void SinCos (ref Vector2 *value*, out Vector2 *sinResult*, out Vector2 *cosResult*) [static]

Takes the sine and then the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine and cosine of.
<i>sinResult</i>	When the method completes, contains the sine of each component in the input vector.
<i>cosResult</i>	When the method completes, contains the cosine of each component in the input vector.

4.5.2.56 static void SmoothStep (ref Vector2 *start*, ref Vector2 *end*, float *amount*, out Vector2 *result*) [static]

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
--------------	---------------

<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .
<i>result</i>	When the method completes, contains the cubic interpolation of the two vectors.

4.5.2.57 `static Vector2 SmoothStep (Vector2 start, Vector2 end, float amount)` `[static]`

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The cubic interpolation of the two vectors.

4.5.2.58 `static void Sqrt (ref Vector2 value, out Vector2 result)` `[static]`

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
<i>result</i>	When the method completes, contains a vector that is the square root of the input vector.

4.5.2.59 `static Vector2 Sqrt (Vector2 value)` `[static]`

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
--------------	--

Returns

A vector that is the square root of the input vector.

4.5.2.60 `static void Subtract (ref Vector2 left, ref Vector2 right, out Vector2 result)` `[static]`

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.
<i>result</i>	When the method completes, contains the difference of the two vectors.

4.5.2.61 static Vector2 Subtract (Vector2 left, Vector2 right) [static]

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.

Returns

The difference of the two vectors.

4.5.2.62 static void Tan (ref Vector2 value, out Vector2 result) [static]

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
<i>result</i>	When the method completes, contains a vector that contains the tangent of each component in the input vector.

4.5.2.63 static Vector2 Tan (Vector2 value) [static]

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
--------------	------------------------------------

Returns

A vector that contains the tangent of each component in the input vector.

4.6 Vector3Helper Class Reference

Helper methods for UnityEngine.Vector3

Static Public Member Functions

- static void **Sqrt** (ref **Vector3** value, out **Vector3** result)
Takes the square root of each component in the vector.
- static **Vector3 Sqrt** (**Vector3** value)
Takes the square root of each component in the vector.
- static void **Reciprocal** (ref **Vector3** value, out **Vector3** result)
Takes the reciprocal of each component in the vector.
- static **Vector3 Reciprocal** (**Vector3** value)
Takes the reciprocal of each component in the vector.
- static void **ReciprocalSqrt** (ref **Vector3** value, out **Vector3** result)
Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.
- static **Vector3 ReciprocalSqrt** (**Vector3** value)

- Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.*

 - static void **Exp** (ref **Vector3** value, out **Vector3** result)

Takes e raised to the component in the vector.
- static **Vector3 Exp** (**Vector3** value)

Takes e raised to the component in the vector.
- static void **SinCos** (ref **Vector3** value, out **Vector3** sinResult, out **Vector3** cosResult)

Takes the sine and then the cosine of each component in the vector.
- static void **Sin** (ref **Vector3** value, out **Vector3** result)

Takes the sine of each component in the vector.
- static **Vector3 Sin** (**Vector3** value)

Takes the sine of each component in the vector.
- static void **Cos** (ref **Vector3** value, out **Vector3** result)

Takes the cosine of each component in the vector.
- static **Vector3 Cos** (**Vector3** value)

Takes the cosine of each component in the vector.
- static void **Tan** (ref **Vector3** value, out **Vector3** result)

Takes the tangent of each component in the vector.
- static **Vector3 Tan** (**Vector3** value)

Takes the tangent of each component in the vector.
- static void **Add** (ref **Vector3** left, ref **Vector3** right, out **Vector3** result)

Adds two vectors.
- static **Vector3 Add** (**Vector3** left, **Vector3** right)

Adds two vectors.
- static void **Subtract** (ref **Vector3** left, ref **Vector3** right, out **Vector3** result)

Subtracts two vectors.
- static **Vector3 Subtract** (**Vector3** left, **Vector3** right)

Subtracts two vectors.
- static void **Multiply** (ref **Vector3** value, float scalar, out **Vector3** result)

Scales a vector by the given value.
- static **Vector3 Multiply** (**Vector3** value, float scalar)

Scales a vector by the given value.
- static void **Modulate** (ref **Vector3** left, ref **Vector3** right, out **Vector3** result)

Modulates a vector with another by performing component-wise multiplication.
- static **Vector3 Modulate** (**Vector3** left, **Vector3** right)

Modulates a vector with another by performing component-wise multiplication.
- static void **Divide** (ref **Vector3** value, float scalar, out **Vector3** result)

Scales a vector by the given value.
- static **Vector3 Divide** (**Vector3** value, float scalar)

Scales a vector by the given value.
- static void **Negate** (ref **Vector3** value, out **Vector3** result)

Reverses the direction of a given vector.
- static **Vector3 Negate** (**Vector3** value)

Reverses the direction of a given vector.
- static void **Abs** (ref **Vector3** value, out **Vector3** result)

Takes the absolute value of each component.
- static **Vector3 Abs** (**Vector3** value)

Takes the absolute value of each component.
- static void **Barycentric** (ref **Vector3** value1, ref **Vector3** value2, ref **Vector3** value3, float amount1, float amount2, out **Vector3** result)

Returns a UnityEngine.Vector3 containing the 3D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 3D triangle.

- static **Vector3 Barycentric** (**Vector3** value1, **Vector3** value2, **Vector3** value3, float amount1, float amount2)
Returns a UnityEngine.Vector3 containing the 3D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 3D triangle.
- static void **Clamp** (ref **Vector3** value, ref **Vector3** min, ref **Vector3** max, out **Vector3** result)
Restricts a value to be within a specified range.
- static **Vector3 Clamp** (**Vector3** value, **Vector3** min, **Vector3** max)
Restricts a value to be within a specified range.
- static void **Cross** (ref **Vector3** left, ref **Vector3** right, out **Vector3** result)
Calculates the cross product of two vectors.
- static **Vector3 Cross** (**Vector3** left, **Vector3** right)
Calculates the cross product of two vectors.
- static void **TripleProduct** (ref **Vector3** value1, ref **Vector3** value2, ref **Vector3** value3, out float result)
Calculates the tripple cross product of three vectors.
- static float **TripleProduct** (**Vector3** value1, **Vector3** value2, **Vector3** value3)
Calculates the tripple cross product of three vectors.
- static void **Distance** (ref **Vector3** value1, ref **Vector3** value2, out float result)
Calculates the distance between two vectors.
- static float **Distance** (**Vector3** value1, **Vector3** value2)
Calculates the distance between two vectors.
- static void **DistanceSquared** (ref **Vector3** value1, ref **Vector3** value2, out float result)
Calculates the squared distance between two vectors.
- static float **DistanceSquared** (**Vector3** value1, **Vector3** value2)
Calculates the squared distance between two vectors.
- static void **Dot** (ref **Vector3** left, ref **Vector3** right, out float result)
Calculates the dot product of two vectors.
- static float **Dot** (**Vector3** left, **Vector3** right)
Calculates the dot product of two vectors.
- static void **Normalize** (ref **Vector3** value, out **Vector3** result)
Converts the vector into a unit vector.
- static **Vector3 Normalize** (**Vector3** value)
Converts the vector into a unit vector.
- static void **Lerp** (ref **Vector3** start, ref **Vector3** end, float amount, out **Vector3** result)
Performs a linear interpolation between two vectors.
- static **Vector3 Lerp** (**Vector3** start, **Vector3** end, float amount)
Performs a linear interpolation between two vectors.
- static void **SmoothStep** (ref **Vector3** start, ref **Vector3** end, float amount, out **Vector3** result)
Performs a cubic interpolation between two vectors.
- static **Vector3 SmoothStep** (**Vector3** start, **Vector3** end, float amount)
Performs a cubic interpolation between two vectors.
- static void **Hermite** (ref **Vector3** value1, ref **Vector3** tangent1, ref **Vector3** value2, ref **Vector3** tangent2, float amount, out **Vector3** result)
Performs a Hermite spline interpolation.
- static **Vector3 Hermite** (**Vector3** value1, **Vector3** tangent1, **Vector3** value2, **Vector3** tangent2, float amount)
Performs a Hermite spline interpolation.
- static void **CatmullRom** (ref **Vector3** value1, ref **Vector3** value2, ref **Vector3** value3, ref **Vector3** value4, float amount, out **Vector3** result)
Performs a Catmull-Rom interpolation using the specified positions.
- static **Vector3 CatmullRom** (**Vector3** value1, **Vector3** value2, **Vector3** value3, **Vector3** value4, float amount)

- Performs a Catmull-Rom interpolation using the specified positions.*
- static void **Max** (ref **Vector3** value1, ref **Vector3** value2, out **Vector3** result)
Returns a vector containing the largest components of the specified vectors.
- static **Vector3 Max** (**Vector3** value1, **Vector3** value2)
Returns a vector containing the largest components of the specified vectors.
- static void **Min** (ref **Vector3** value1, ref **Vector3** value2, out **Vector3** result)
Returns a vector containing the smallest components of the specified vectors.
- static **Vector3 Min** (**Vector3** value1, **Vector3** value2)
Returns a vector containing the smallest components of the specified vectors.
- static void **Reflect** (ref **Vector3** vector, ref **Vector3** normal, out **Vector3** result)
Returns the reflection of a vector off a surface that has the specified normal.
- static **Vector3 Reflect** (**Vector3** vector, **Vector3** normal)
Returns the reflection of a vector off a surface that has the specified normal.
- static void **Refract** (ref **Vector3** vector, ref **Vector3** normal, float index, out **Vector3** result)
Returns the fraction of a vector off a surface that has the specified normal and index.
- static **Vector3 Refract** (**Vector3** vector, **Vector3** normal, float index)
Returns the fraction of a vector off a surface that has the specified normal and index.
- static void **Orthogonalize** (**Vector3**[] destination, params **Vector3**[] source)
Orthogonalizes a list of vectors.
- static void **Orthonormalize** (**Vector3**[] destination, params **Vector3**[] source)
Orthonormalizes a list of vectors.

4.6.1 Detailed Description

Helper methods for UnityEngine.Vector3

4.6.2 Member Function Documentation

4.6.2.1 static void Abs (ref **Vector3** value, out **Vector3** result) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
<i>result</i>	When the method completes, contains a vector that has all positive components.

4.6.2.2 static **Vector3** Abs (**Vector3** value) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
--------------	---

Returns

A vector that has all positive components.

4.6.2.3 static void Add (ref Vector3 left, ref Vector3 right, out Vector3 result) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.
<i>result</i>	When the method completes, contains the sum of the two vectors.

4.6.2.4 static Vector3 Add (Vector3 left, Vector3 right) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.

Returns

The sum of the two vectors.

4.6.2.5 static void Barycentric (ref Vector3 value1, ref Vector3 value2, ref Vector3 value3, float amount1, float amount2, out Vector3 result) [static]

Returns a UnityEngine.Vector3 containing the 3D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 3D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 2 of the triangle.
<i>value3</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).
<i>result</i>	When the method completes, contains the 3D Cartesian coordinates of the specified point.

4.6.2.6 static Vector3 Barycentric (Vector3 value1, Vector3 value2, Vector3 value3, float amount1, float amount2) [static]

Returns a UnityEngine.Vector3 containing the 3D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 3D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 2 of the triangle.

<i>value3</i>	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).

Returns

A new UnityEngine.Vector3 containing the 3D Cartesian coordinates of the specified point.

4.6.2.7 `static void CatmullRom (ref Vector3 value1, ref Vector3 value2, ref Vector3 value3, ref Vector3 value4, float amount, out Vector3 result) [static]`

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Catmull-Rom interpolation.

4.6.2.8 `static Vector3 CatmullRom (Vector3 value1, Vector3 value2, Vector3 value3, Vector3 value4, float amount) [static]`

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.

Returns

A vector that is the result of the Catmull-Rom interpolation.

4.6.2.9 `static void Clamp (ref Vector3 value, ref Vector3 min, ref Vector3 max, out Vector3 result) [static]`

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.
<i>result</i>	When the method completes, contains the clamped value.

4.6.2.10 static Vector3 Clamp (Vector3 value, Vector3 min, Vector3 max) [static]

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.

Returns

The clamped value.

4.6.2.11 static void Cos (ref Vector3 value, out Vector3 result) [static]

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
<i>result</i>	When the method completes, contains a vector that contains the cosine of each component in the input vector.

4.6.2.12 static Vector3 Cos (Vector3 value) [static]

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
--------------	-----------------------------------

Returns

A vector that contains the cosine of each component in the input vector.

4.6.2.13 static void Cross (ref Vector3 left, ref Vector3 right, out Vector3 result) [static]

Calculates the cross product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.
<i>result</i>	When the method completes, contains the cross product of the two vectors.

4.6.2.14 static Vector3 Cross (Vector3 left, Vector3 right) [static]

Calculates the cross product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.

Returns

The cross product of the two vectors.

4.6.2.15 `static void Distance (ref Vector3 value1, ref Vector3 value2, out float result)` `[static]`

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.
<i>result</i>	When the method completes, contains the distance between the two vectors.

UnityEngine.Vector3.DistanceSquared(ref Vector3, ref Vector3, out float) may be preferred when only the relative distance is needed and speed is of the essence.

4.6.2.16 `static float Distance (Vector3 value1, Vector3 value2)` `[static]`

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The distance between the two vectors.

UnityEngine.Vector3.DistanceSquared(Vector3, Vector3) may be preferred when only the relative distance is needed and speed is of the essence.

4.6.2.17 `static void DistanceSquared (ref Vector3 value1, ref Vector3 value2, out float result)` `[static]`

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.
<i>result</i>	When the method completes, contains the squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.6.2.18 `static float DistanceSquared (Vector3 value1, Vector3 value2)` `[static]`

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.6.2.19 static void Divide (ref Vector3 *value*, float *scalar*, out Vector3 *result*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.6.2.20 static Vector3 Divide (Vector3 *value*, float *scalar*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.6.2.21 static void Dot (ref Vector3 *left*, ref Vector3 *right*, out float *result*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.
<i>result</i>	When the method completes, contains the dot product of the two vectors.

4.6.2.22 static float Dot (Vector3 *left*, Vector3 *right*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.

Returns

The dot product of the two vectors.

4.6.2.23 `static void Exp (ref Vector3 value, out Vector3 result) [static]`

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
<i>result</i>	When the method completes, contains a vector that has e raised to each of the components in the input vector.

4.6.2.24 `static Vector3 Exp (Vector3 value) [static]`

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
--------------	--

Returns

A vector that has e raised to each of the components in the input vector.

4.6.2.25 `static void Hermite (ref Vector3 value1, ref Vector3 tangent1, ref Vector3 value2, ref Vector3 tangent2, float amount, out Vector3 result) [static]`

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.
<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Hermite spline interpolation.

4.6.2.26 `static Vector3 Hermite (Vector3 value1, Vector3 tangent1, Vector3 value2, Vector3 tangent2, float amount) [static]`

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.

<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.

Returns

The result of the Hermite spline interpolation.

4.6.2.27 static void Lerp (ref Vector3 start, ref Vector3 end, float amount, out Vector3 result) [static]

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .
<i>result</i>	When the method completes, contains the linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.6.2.28 static Vector3 Lerp (Vector3 start, Vector3 end, float amount) [static]

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.6.2.29 static void Max (ref Vector3 value1, ref Vector3 value2, out Vector3 result) [static]

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the largest components of the source vectors.

4.6.2.30 static Vector3 Max (Vector3 value1, Vector3 value2) [static]

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the largest components of the source vectors.

4.6.2.31 static void Min (ref Vector3 value1, ref Vector3 value2, out Vector3 result) [static]

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the smallest components of the source vectors.

4.6.2.32 static Vector3 Min (Vector3 value1, Vector3 value2) [static]

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the smallest components of the source vectors.

4.6.2.33 static void Modulate (ref Vector3 left, ref Vector3 right, out Vector3 result) [static]

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.
<i>result</i>	When the method completes, contains the modulated vector.

4.6.2.34 static Vector3 Modulate (Vector3 left, Vector3 right) [static]

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.

Returns

The modulated vector.

4.6.2.35 `static void Multiply (ref Vector3 value, float scalar, out Vector3 result) [static]`

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.6.2.36 `static Vector3 Multiply (Vector3 value, float scalar) [static]`

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.6.2.37 `static void Negate (ref Vector3 value, out Vector3 result) [static]`

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
<i>result</i>	When the method completes, contains a vector facing in the opposite direction.

4.6.2.38 `static Vector3 Negate (Vector3 value) [static]`

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
--------------	-----------------------

Returns

A vector facing in the opposite direction.

4.6.2.39 static void Normalize (ref Vector3 value, out Vector3 result) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
<i>result</i>	When the method completes, contains the normalized vector.

4.6.2.40 static Vector3 Normalize (Vector3 value) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
--------------	--------------------------

Returns

The normalized vector.

4.6.2.41 static void Orthogonalize (Vector3[] destination, params Vector3[] source) [static]

Orthogonalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthogonalized vectors.
<i>source</i>	The list of vectors to orthogonalize.

Orthogonalization is the process of making all vectors orthogonal to each other. This means that any given vector in the list will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.6.2.42 static void Orthonormalize (Vector3[] destination, params Vector3[] source) [static]

Orthonormalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthonormalized vectors.
<i>source</i>	The list of vectors to orthonormalize.

Orthonormalization is the process of making all vectors orthogonal to each other and making all vectors of unit length. This means that any given vector will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.6.2.43 static void Reciprocal (ref Vector3 value, out Vector3 result) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the reciprocal of the input vector.

4.6.2.44 static Vector3 Reciprocal (Vector3 value) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
--------------	---------------------------------------

Returns

A vector that is the reciprocal of the input vector.

4.6.2.45 static void ReciprocalSqrt (ref Vector3 value, out Vector3 result) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the square root and reciprocal of the input vector.

4.6.2.46 static Vector3 ReciprocalSqrt (Vector3 value) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
--------------	---

Returns

A vector that is the square root and reciprocal of the input vector.

4.6.2.47 `static void Reflect (ref Vector3 vector, ref Vector3 normal, out Vector3 result)` `[static]`

Returns the reflection of a vector off a surface that has the specified normal.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>result</i>	When the method completes, contains the reflected vector.

Reflect only gives the direction of a reflection off a surface, it does not determine whether the original vector was close enough to the surface to hit it.

4.6.2.48 `static Vector3 Reflect (Vector3 vector, Vector3 normal)` `[static]`

Returns the reflection of a vector off a surface that has the specified normal.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.

Returns

The reflected vector.

Reflect only gives the direction of a reflection off a surface, it does not determine whether the original vector was close enough to the surface to hit it.

4.6.2.49 `static void Refract (ref Vector3 vector, ref Vector3 normal, float index, out Vector3 result)` `[static]`

Returns the fraction of a vector off a surface that has the specified normal and index.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>index</i>	Index of refraction.
<i>result</i>	When the method completes, contains the refracted vector.

4.6.2.50 `static Vector3 Refract (Vector3 vector, Vector3 normal, float index)` `[static]`

Returns the fraction of a vector off a surface that has the specified normal and index.

Parameters

<i>vector</i>	The source vector.
<i>normal</i>	Normal of the surface.
<i>index</i>	Index of refraction.

Returns

The refracted vector.

4.6.2.51 `static void Sin (ref Vector3 value, out Vector3 result) [static]`

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
<i>result</i>	When the method completes, a vector that contains the sine of each component in the input vector.

4.6.2.52 `static Vector3 Sin (Vector3 value) [static]`

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
--------------	---------------------------------

Returns

A vector that contains the sine of each component in the input vector.

4.6.2.53 `static void SinCos (ref Vector3 value, out Vector3 sinResult, out Vector3 cosResult) [static]`

Takes the sine and then the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine and cosine of.
<i>sinResult</i>	When the method completes, contains the sine of each component in the input vector.
<i>cosResult</i>	When the method completes, contains the cosine of each component in the input vector.

4.6.2.54 `static void SmoothStep (ref Vector3 start, ref Vector3 end, float amount, out Vector3 result) [static]`

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .
<i>result</i>	When the method completes, contains the cubic interpolation of the two vectors.

4.6.2.55 `static Vector3 SmoothStep (Vector3 start, Vector3 end, float amount) [static]`

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The cubic interpolation of the two vectors.

4.6.2.56 `static void Sqrt (ref Vector3 value, out Vector3 result) [static]`

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
<i>result</i>	When the method completes, contains a vector that is the square root of the input vector.

4.6.2.57 `static Vector3 Sqrt (Vector3 value) [static]`

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
--------------	--

Returns

A vector that is the square root of the input vector.

4.6.2.58 `static void Subtract (ref Vector3 left, ref Vector3 right, out Vector3 result) [static]`

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.
<i>result</i>	When the method completes, contains the difference of the two vectors.

4.6.2.59 `static Vector3 Subtract (Vector3 left, Vector3 right) [static]`

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.

Returns

The difference of the two vectors.

4.6.2.60 `static void Tan (ref Vector3 value, out Vector3 result) [static]`

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
<i>result</i>	When the method completes, contains a vector that contains the tangent of each component in the input vector.

4.6.2.61 `static Vector3 Tan (Vector3 value) [static]`

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
--------------	------------------------------------

Returns

A vector that contains the tangent of each component in the input vector.

4.6.2.62 `static void TripleProduct (ref Vector3 value1, ref Vector3 value2, ref Vector3 value3, out float result) [static]`

Calculates the tripple cross product of three vectors.

Parameters

<i>value1</i>	First source vector.
<i>value2</i>	Second source vector.
<i>value3</i>	Third source vector.
<i>result</i>	When the method completes, contains the tripple cross product of the three vectors.

4.6.2.63 `static float TripleProduct (Vector3 value1, Vector3 value2, Vector3 value3) [static]`

Calculates the tripple cross product of three vectors.

Parameters

<i>value1</i>	First source vector.
<i>value2</i>	Second source vector.
<i>value3</i>	Third source vector.

Returns

The tripple cross product of the three vectors.

4.7 Vector4Helper Class Reference

Helper methods for UnityEngine.Vector4

Static Public Member Functions

- static void **Sqrt** (ref Vector4 value, out Vector4 result)
Takes the square root of each component in the vector.
- static Vector4 **Sqrt** (Vector4 value)
Takes the square root of each component in the vector.
- static void **Reciprocal** (ref Vector4 value, out Vector4 result)
Takes the reciprocal of each component in the vector.
- static Vector4 **Reciprocal** (Vector4 value)
Takes the reciprocal of each component in the vector.
- static void **ReciprocalSqrt** (ref Vector4 value, out Vector4 result)
Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.
- static Vector4 **ReciprocalSqrt** (Vector4 value)
Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.
- static void **Exp** (ref Vector4 value, out Vector4 result)
Takes e raised to the component in the vector.
- static Vector4 **Exp** (Vector4 value)
Takes e raised to the component in the vector.
- static void **SinCos** (ref Vector4 value, out Vector4 sinResult, out Vector4 cosResult)
Takes the sine and then the cosine of each component in the vector.
- static void **Sin** (ref Vector4 value, out Vector4 result)
Takes the sine of each component in the vector.
- static Vector4 **Sin** (Vector4 value)
Takes the sine of each component in the vector.
- static void **Cos** (ref Vector4 value, out Vector4 result)
Takes the cosine of each component in the vector.
- static Vector4 **Cos** (Vector4 value)
Takes the cosine of each component in the vector.
- static void **Tan** (ref Vector4 value, out Vector4 result)
Takes the tangent of each component in the vector.
- static Vector4 **Tan** (Vector4 value)
Takes the tangent of each component in the vector.
- static void **Add** (ref Vector4 left, ref Vector4 right, out Vector4 result)
Adds two vectors.
- static Vector4 **Add** (Vector4 left, Vector4 right)
Adds two vectors.
- static void **Subtract** (ref Vector4 left, ref Vector4 right, out Vector4 result)
Subtracts two vectors.
- static Vector4 **Subtract** (Vector4 left, Vector4 right)
Subtracts two vectors.
- static void **Multiply** (ref Vector4 value, float scalar, out Vector4 result)
Scales a vector by the given value.
- static Vector4 **Multiply** (Vector4 value, float scalar)
Scales a vector by the given value.
- static void **Modulate** (ref Vector4 left, ref Vector4 right, out Vector4 result)
Modulates a vector with another by performing component-wise multiplication.
- static Vector4 **Modulate** (Vector4 left, Vector4 right)
Modulates a vector with another by performing component-wise multiplication.
- static void **Divide** (ref Vector4 value, float scalar, out Vector4 result)
Scales a vector by the given value.
- static Vector4 **Divide** (Vector4 value, float scalar)

- Scales a vector by the given value.*

 - static void **Negate** (ref Vector4 value, out Vector4 result)
- Reverses the direction of a given vector.*

 - static Vector4 **Negate** (Vector4 value)
- Reverses the direction of a given vector.*

 - static void **Abs** (ref Vector4 value, out Vector4 result)
- Takes the absolute value of each component.*

 - static Vector4 **Abs** (Vector4 value)
- Takes the absolute value of each component.*

 - static void **Barycentric** (ref Vector4 value1, ref Vector4 value2, ref Vector4 value3, float amount1, float amount2, out Vector4 result)
- Returns a UnityEngine.Vector4 containing the 4D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 4D triangle.*

 - static Vector4 **Barycentric** (Vector4 value1, Vector4 value2, Vector4 value3, float amount1, float amount2)
- Returns a UnityEngine.Vector4 containing the 4D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 4D triangle.*

 - static void **Clamp** (ref Vector4 value, ref Vector4 min, ref Vector4 max, out Vector4 result)
- Restricts a value to be within a specified range.*

 - static Vector4 **Clamp** (Vector4 value, Vector4 min, Vector4 max)
- Restricts a value to be within a specified range.*

 - static void **Distance** (ref Vector4 value1, ref Vector4 value2, out float result)
- Calculates the distance between two vectors.*

 - static float **Distance** (Vector4 value1, Vector4 value2)
- Calculates the distance between two vectors.*

 - static void **DistanceSquared** (ref Vector4 value1, ref Vector4 value2, out float result)
- Calculates the squared distance between two vectors.*

 - static float **DistanceSquared** (Vector4 value1, Vector4 value2)
- Calculates the squared distance between two vectors.*

 - static void **Dot** (ref Vector4 left, ref Vector4 right, out float result)
- Calculates the dot product of two vectors.*

 - static float **Dot** (Vector4 left, Vector4 right)
- Calculates the dot product of two vectors.*

 - static void **Normalize** (ref Vector4 value, out Vector4 result)
- Converts the vector into a unit vector.*

 - static Vector4 **Normalize** (Vector4 value)
- Converts the vector into a unit vector.*

 - static void **Lerp** (ref Vector4 start, ref Vector4 end, float amount, out Vector4 result)
- Performs a linear interpolation between two vectors.*

 - static Vector4 **Lerp** (Vector4 start, Vector4 end, float amount)
- Performs a linear interpolation between two vectors.*

 - static void **SmoothStep** (ref Vector4 start, ref Vector4 end, float amount, out Vector4 result)
- Performs a cubic interpolation between two vectors.*

 - static Vector4 **SmoothStep** (Vector4 start, Vector4 end, float amount)
- Performs a cubic interpolation between two vectors.*

 - static void **Hermite** (ref Vector4 value1, ref Vector4 tangent1, ref Vector4 value2, ref Vector4 tangent2, float amount, out Vector4 result)
- Performs a Hermite spline interpolation.*

 - static Vector4 **Hermite** (Vector4 value1, Vector4 tangent1, Vector4 value2, Vector4 tangent2, float amount)
- Performs a Hermite spline interpolation.*

 - static void **CatmullRom** (ref Vector4 value1, ref Vector4 value2, ref Vector4 value3, ref Vector4 value4, float amount, out Vector4 result)

- Performs a Catmull-Rom interpolation using the specified positions.*
- static Vector4 **CatmullRom** (Vector4 value1, Vector4 value2, Vector4 value3, Vector4 value4, float amount)
Performs a Catmull-Rom interpolation using the specified positions.
- static void **Max** (ref Vector4 value1, ref Vector4 value2, out Vector4 result)
Returns a vector containing the largest components of the specified vectors.
- static Vector4 **Max** (Vector4 value1, Vector4 value2)
Returns a vector containing the largest components of the specified vectors.
- static void **Min** (ref Vector4 value1, ref Vector4 value2, out Vector4 result)
Returns a vector containing the smallest components of the specified vectors.
- static Vector4 **Min** (Vector4 value1, Vector4 value2)
Returns a vector containing the smallest components of the specified vectors.
- static void **Orthogonalize** (Vector4[] destination, params Vector4[] source)
Orthogonalizes a list of vectors.
- static void **Orthonormalize** (Vector4[] destination, params Vector4[] source)
Orthonormalizes a list of vectors.

4.7.1 Detailed Description

Helper methods for UnityEngine.Vector4

4.7.2 Member Function Documentation

4.7.2.1 static void Abs (ref Vector4 value, out Vector4 result) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
<i>result</i>	When the method completes, contains a vector that has all positive components.

4.7.2.2 static Vector4 Abs (Vector4 value) [static]

Takes the absolute value of each component.

Parameters

<i>value</i>	The vector to take the absolute value of.
--------------	---

Returns

A vector that has all positive components.

4.7.2.3 static void Add (ref Vector4 left, ref Vector4 right, out Vector4 result) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.
<i>result</i>	When the method completes, contains the sum of the two vectors.

4.7.2.4 static Vector4 Add (Vector4 left, Vector4 right) [static]

Adds two vectors.

Parameters

<i>left</i>	The first vector to add.
<i>right</i>	The second vector to add.

Returns

The sum of the two vectors.

4.7.2.5 static void Barycentric (ref Vector4 value1, ref Vector4 value2, ref Vector4 value3, float amount1, float amount2, out Vector4 result) [static]

Returns a UnityEngine.Vector4 containing the 4D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 4D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 2 of the triangle.
<i>value3</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).
<i>result</i>	When the method completes, contains the 4D Cartesian coordinates of the specified point.

4.7.2.6 static Vector4 Barycentric (Vector4 value1, Vector4 value2, Vector4 value3, float amount1, float amount2) [static]

Returns a UnityEngine.Vector4 containing the 4D Cartesian coordinates of a point specified in Barycentric coordinates relative to a 4D triangle.

Parameters

<i>value1</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 1 of the triangle.
<i>value2</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 2 of the triangle.
<i>value3</i>	A UnityEngine.Vector4 containing the 4D Cartesian coordinates of vertex 3 of the triangle.
<i>amount1</i>	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in <i>value2</i>).
<i>amount2</i>	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in <i>value3</i>).

Returns

A new UnityEngine.Vector4 containing the 4D Cartesian coordinates of the specified point.

4.7.2.7 static void CatmullRom (ref Vector4 value1, ref Vector4 value2, ref Vector4 value3, ref Vector4 value4, float amount, out Vector4 result) [static]

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Catmull-Rom interpolation.

4.7.2.8 `static Vector4 CatmullRom (Vector4 value1, Vector4 value2, Vector4 value3, Vector4 value4, float amount)` `[static]`

Performs a Catmull-Rom interpolation using the specified positions.

Parameters

<i>value1</i>	The first position in the interpolation.
<i>value2</i>	The second position in the interpolation.
<i>value3</i>	The third position in the interpolation.
<i>value4</i>	The fourth position in the interpolation.
<i>amount</i>	Weighting factor.

Returns

A vector that is the result of the Catmull-Rom interpolation.

4.7.2.9 `static void Clamp (ref Vector4 value, ref Vector4 min, ref Vector4 max, out Vector4 result)` `[static]`

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.
<i>result</i>	When the method completes, contains the clamped value.

4.7.2.10 `static Vector4 Clamp (Vector4 value, Vector4 min, Vector4 max)` `[static]`

Restricts a value to be within a specified range.

Parameters

<i>value</i>	The value to clamp.
<i>min</i>	The minimum value.
<i>max</i>	The maximum value.

Returns

The clamped value.

4.7.2.11 `static void Cos (ref Vector4 value, out Vector4 result)` `[static]`

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
<i>result</i>	When the method completes, contains a vector that contains the cosine of each component in the input vector.

4.7.2.12 static Vector4 Cos (Vector4 *value*) [static]

Takes the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the cosine of.
--------------	-----------------------------------

Returns

A vector that contains the cosine of each component in the input vector.

4.7.2.13 static void Distance (ref Vector4 *value1*, ref Vector4 *value2*, out float *result*) [static]

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.
<i>result</i>	When the method completes, contains the distance between the two vectors.

UnityEngine.Vector4.DistanceSquared(ref Vector4, ref Vector4, out float) may be preferred when only the relative distance is needed and speed is of the essence.

4.7.2.14 static float Distance (Vector4 *value1*, Vector4 *value2*) [static]

Calculates the distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The distance between the two vectors.

UnityEngine.Vector4.DistanceSquared(Vector4, Vector4) may be preferred when only the relative distance is needed and speed is of the essence.

4.7.2.15 static void DistanceSquared (ref Vector4 *value1*, ref Vector4 *value2*, out float *result*) [static]

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.
<i>result</i>	When the method completes, contains the squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.7.2.16 static float DistanceSquared (Vector4 *value1*, Vector4 *value2*) [static]

Calculates the squared distance between two vectors.

Parameters

<i>value1</i>	The first vector.
<i>value2</i>	The second vector.

Returns

The squared distance between the two vectors.

Distance squared is the value before taking the square root. Distance squared can often be used in place of distance if relative comparisons are being made. For example, consider three points A, B, and C. To determine whether B or C is further from A, compare the distance between A and B to the distance between A and C. Calculating the two distances involves two square roots, which are computationally expensive. However, using distance squared provides the same information and avoids calculating two square roots.

4.7.2.17 static void Divide (ref Vector4 *value*, float *scalar*, out Vector4 *result*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.7.2.18 static Vector4 Divide (Vector4 *value*, float *scalar*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.7.2.19 static void Dot (ref Vector4 *left*, ref Vector4 *right*, out float *result*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector
<i>right</i>	Second source vector.
<i>result</i>	When the method completes, contains the dot product of the two vectors.

4.7.2.20 static float Dot (Vector4 *left*, Vector4 *right*) [static]

Calculates the dot product of two vectors.

Parameters

<i>left</i>	First source vector.
<i>right</i>	Second source vector.

Returns

The dot product of the two vectors.

4.7.2.21 static void Exp (ref Vector4 *value*, out Vector4 *result*) [static]

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
<i>result</i>	When the method completes, contains a vector that has e raised to each of the components in the input vector.

4.7.2.22 static Vector4 Exp (Vector4 *value*) [static]

Takes e raised to the component in the vector.

Parameters

<i>value</i>	The value to take e raised to each component of.
--------------	--

Returns

A vector that has e raised to each of the components in the input vector.

4.7.2.23 static void Hermite (ref Vector4 *value1*, ref Vector4 *tangent1*, ref Vector4 *value2*, ref Vector4 *tangent2*, float *amount*, out Vector4 *result*) [static]

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.
<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.
<i>result</i>	When the method completes, contains the result of the Hermite spline interpolation.

4.7.2.24 `static Vector4 Hermite (Vector4 value1, Vector4 tangent1, Vector4 value2, Vector4 tangent2, float amount)`
`[static]`

Performs a Hermite spline interpolation.

Parameters

<i>value1</i>	First source position vector.
<i>tangent1</i>	First source tangent vector.
<i>value2</i>	Second source position vector.
<i>tangent2</i>	Second source tangent vector.
<i>amount</i>	Weighting factor.

Returns

The result of the Hermite spline interpolation.

4.7.2.25 `static void Lerp (ref Vector4 start, ref Vector4 end, float amount, out Vector4 result)` `[static]`

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .
<i>result</i>	When the method completes, contains the linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.7.2.26 `static Vector4 Lerp (Vector4 start, Vector4 end, float amount)` `[static]`

Performs a linear interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The linear interpolation of the two vectors.

This method performs the linear interpolation based on the following formula.

```
start + (end - start) * amount
```

Passing *amount* a value of 0 will cause *start* to be returned; a value of 1 will cause *end* to be returned.

4.7.2.27 `static void Max (ref Vector4 value1, ref Vector4 value2, out Vector4 result) [static]`

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the largest components of the source vectors.

4.7.2.28 `static Vector4 Max (Vector4 value1, Vector4 value2) [static]`

Returns a vector containing the largest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the largest components of the source vectors.

4.7.2.29 `static void Min (ref Vector4 value1, ref Vector4 value2, out Vector4 result) [static]`

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.
<i>result</i>	When the method completes, contains an new vector composed of the smallest components of the source vectors.

4.7.2.30 `static Vector4 Min (Vector4 value1, Vector4 value2) [static]`

Returns a vector containing the smallest components of the specified vectors.

Parameters

<i>value1</i>	The first source vector.
<i>value2</i>	The second source vector.

Returns

A vector containing the smallest components of the source vectors.

4.7.2.31 `static void Modulate (ref Vector4 left, ref Vector4 right, out Vector4 result) [static]`

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.
<i>result</i>	When the method completes, contains the modulated vector.

4.7.2.32 static Vector4 Modulate (Vector4 *left*, Vector4 *right*) [static]

Modulates a vector with another by performing component-wise multiplication.

Parameters

<i>left</i>	The first vector to modulate.
<i>right</i>	The second vector to modulate.

Returns

The modulated vector.

4.7.2.33 static void Multiply (ref Vector4 *value*, float *scalar*, out Vector4 *result*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.
<i>result</i>	When the method completes, contains the scaled vector.

4.7.2.34 static Vector4 Multiply (Vector4 *value*, float *scalar*) [static]

Scales a vector by the given value.

Parameters

<i>value</i>	The vector to scale.
<i>scalar</i>	The amount by which to scale the vector.

Returns

The scaled vector.

4.7.2.35 static void Negate (ref Vector4 *value*, out Vector4 *result*) [static]

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
<i>result</i>	When the method completes, contains a vector facing in the opposite direction.

4.7.2.36 static Vector4 Negate (Vector4 value) [static]

Reverses the direction of a given vector.

Parameters

<i>value</i>	The vector to negate.
--------------	-----------------------

Returns

A vector facing in the opposite direction.

4.7.2.37 static void Normalize (ref Vector4 value, out Vector4 result) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
<i>result</i>	When the method completes, contains the normalized vector.

4.7.2.38 static Vector4 Normalize (Vector4 value) [static]

Converts the vector into a unit vector.

Parameters

<i>value</i>	The vector to normalize.
--------------	--------------------------

Returns

The normalized vector.

4.7.2.39 static void Orthogonalize (Vector4[] destination, params Vector4[] source) [static]

Orthogonalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthogonalized vectors.
<i>source</i>	The list of vectors to orthogonalize.

Orthogonalization is the process of making all vectors orthogonal to each other. This means that any given vector in the list will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.7.2.40 static void Orthonormalize (Vector4[] *destination*, params Vector4[] *source*) [static]

Orthonormalizes a list of vectors.

Parameters

<i>destination</i>	The list of orthonormalized vectors.
<i>source</i>	The list of vectors to orthonormalize.

Orthonormalization is the process of making all vectors orthogonal to each other and making all vectors of unit length. This means that any given vector will be orthogonal to any other given vector in the list.

Because this method uses the modified Gram-Schmidt process, the resulting vectors tend to be numerically unstable. The numeric stability decreases according to the vectors position in the list so that the first vector is the most stable and the last vector is the least stable.

Exceptions

<i>ArgumentNullException</i>	Thrown when <i>source</i> or <i>destination</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	Thrown when <i>destination</i> is shorter in length than <i>source</i> .

4.7.2.41 static void Reciprocal (ref Vector4 *value*, out Vector4 *result*) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the reciprocal of the input vector.

4.7.2.42 static Vector4 Reciprocal (Vector4 *value*) [static]

Takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the reciprocal of.
--------------	---------------------------------------

Returns

A vector that is the reciprocal of the input vector.

4.7.2.43 static void ReciprocalSqrt (ref Vector4 *value*, out Vector4 *result*) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
<i>result</i>	When the method completes, contains a vector that is the square root and reciprocal of the input vector.

4.7.2.44 static Vector4 ReciprocalSqrt (Vector4 value) [static]

Takes the square root of each component in the vector and then takes the reciprocal of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root and reciprocal of.
--------------	---

Returns

A vector that is the square root and reciprocal of the input vector.

4.7.2.45 static void Sin (ref Vector4 value, out Vector4 result) [static]

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
<i>result</i>	When the method completes, a vector that contains the sine of each component in the input vector.

4.7.2.46 static Vector4 Sin (Vector4 value) [static]

Takes the sine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine of.
--------------	---------------------------------

Returns

A vector that contains the sine of each component in the input vector.

4.7.2.47 static void SinCos (ref Vector4 value, out Vector4 sinResult, out Vector4 cosResult) [static]

Takes the sine and then the cosine of each component in the vector.

Parameters

<i>value</i>	The vector to take the sine and cosine of.
<i>sinResult</i>	When the method completes, contains the sine of each component in the input vector.
<i>cosResult</i>	When the method completes, contains the cosine of each component in the input vector.

4.7.2.48 static void SmoothStep (ref Vector4 start, ref Vector4 end, float amount, out Vector4 result) [static]

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

<i>result</i>	When the method completes, contains the cubic interpolation of the two vectors.
---------------	---

4.7.2.49 static Vector4 SmoothStep (Vector4 *start*, Vector4 *end*, float *amount*) [static]

Performs a cubic interpolation between two vectors.

Parameters

<i>start</i>	Start vector.
<i>end</i>	End vector.
<i>amount</i>	Value between 0 and 1 indicating the weight of <i>end</i> .

Returns

The cubic interpolation of the two vectors.

4.7.2.50 static void Sqrt (ref Vector4 *value*, out Vector4 *result*) [static]

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
<i>result</i>	When the method completes, contains a vector that is the square root of the input vector.

4.7.2.51 static Vector4 Sqrt (Vector4 *value*) [static]

Takes the square root of each component in the vector.

Parameters

<i>value</i>	The vector to take the square root of.
--------------	--

Returns

A vector that is the square root of the input vector.

4.7.2.52 static void Subtract (ref Vector4 *left*, ref Vector4 *right*, out Vector4 *result*) [static]

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.
<i>result</i>	When the method completes, contains the difference of the two vectors.

4.7.2.53 static Vector4 Subtract (Vector4 *left*, Vector4 *right*) [static]

Subtracts two vectors.

Parameters

<i>left</i>	The first vector to subtract.
<i>right</i>	The second vector to subtract.

Returns

The difference of the two vectors.

4.7.2.54 `static void Tan (ref Vector4 value, out Vector4 result) [static]`

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
<i>result</i>	When the method completes, contains a vector that contains the tangent of each component in the input vector.

4.7.2.55 `static Vector4 Tan (Vector4 value) [static]`

Takes the tangent of each component in the vector.

Parameters

<i>value</i>	The vector to take the tangent of.
--------------	------------------------------------

Returns

A vector that contains the tangent of each component in the input vector.

Index

Abs

- MathHelper::Vector2Helper, 28
- MathHelper::Vector3Helper, 46
- MathHelper::Vector4Helper, 64

Add

- MathHelper::Vector2Helper, 28
- MathHelper::Vector3Helper, 47
- MathHelper::Vector4Helper, 64, 65

Back

- MathHelper, 6

Barycentric

- MathHelper::Vector2Helper, 29
- MathHelper::Vector3Helper, 47
- MathHelper::Vector4Helper, 65

BoundingBox, 7

- MathHelper::BoundingBox, 7

BoundingSphere, 8

- MathHelper::BoundingSphere, 9

BoxContainsBox

- MathHelper::CollisionHelper, 12

BoxContainsPoint

- MathHelper::CollisionHelper, 12

BoxContainsSphere

- MathHelper::CollisionHelper, 12

BoxContainsTriangle

- MathHelper::CollisionHelper, 12

BoxIntersectsBox

- MathHelper::CollisionHelper, 13

BoxIntersectsSphere

- MathHelper::CollisionHelper, 13

BoxIntersectsTriangle

- MathHelper::CollisionHelper, 13

CatmullRom

- MathHelper::Vector2Helper, 29, 30
- MathHelper::Vector3Helper, 48
- MathHelper::Vector4Helper, 65, 66

Center

- MathHelper::BoundingSphere, 9

Clamp

- MathHelper::Vector2Helper, 30
- MathHelper::Vector3Helper, 48
- MathHelper::Vector4Helper, 66

ClosestPointOnBoxToPoint

- MathHelper::CollisionHelper, 13

ClosestPointOnPlaneToPoint

- MathHelper::CollisionHelper, 14

ClosestPointOnSegmentToPoint

- MathHelper::CollisionHelper, 14

ClosestPointOnSphereToPoint

- MathHelper::CollisionHelper, 14

ClosestPointOnSphereToSphere

- MathHelper::CollisionHelper, 14

ClosestPointOnTriangleToPoint

- MathHelper::CollisionHelper, 15

CollisionHelper, 9

ContainmentType

- MathHelper, 6

Contains

- MathHelper, 6

Cos

- MathHelper::Vector2Helper, 30, 31
- MathHelper::Vector3Helper, 49
- MathHelper::Vector4Helper, 66, 67

Cross

- MathHelper::Vector3Helper, 49

Disjoint

- MathHelper, 6

Distance

- MathHelper::Vector2Helper, 31
- MathHelper::Vector3Helper, 50
- MathHelper::Vector4Helper, 67

DistanceBoxBox

- MathHelper::CollisionHelper, 15

DistanceBoxPoint

- MathHelper::CollisionHelper, 15

DistancePlanePoint

- MathHelper::CollisionHelper, 15

DistanceSpherePoint

- MathHelper::CollisionHelper, 16

DistanceSphereSphere

- MathHelper::CollisionHelper, 16

DistanceSquared

- MathHelper::Vector2Helper, 31, 32
- MathHelper::Vector3Helper, 50
- MathHelper::Vector4Helper, 67, 68

Divide

- MathHelper::Vector2Helper, 32
- MathHelper::Vector3Helper, 51
- MathHelper::Vector4Helper, 68

Dot

- MathHelper::Vector2Helper, 32, 33
- MathHelper::Vector3Helper, 51
- MathHelper::Vector4Helper, 68, 69

Exp

- MathHelper::Vector2Helper, 33
- MathHelper::Vector3Helper, 52

- MathHelper::Vector4Helper, 69
- FloatHelper, 24
- Front
 - MathHelper, 6
- GetCorners
 - MathHelper::BoundingBox, 8
- Hermite
 - MathHelper::Vector2Helper, 33, 34
 - MathHelper::Vector3Helper, 52
 - MathHelper::Vector4Helper, 69, 70
- Intersecting
 - MathHelper, 6
- Intersects
 - MathHelper, 6
- Lerp
 - MathHelper::Vector2Helper, 34
 - MathHelper::Vector3Helper, 53
 - MathHelper::Vector4Helper, 70
- MathHelper, 5
 - Back, 6
 - ContainmentType, 6
 - Contains, 6
 - Disjoint, 6
 - Front, 6
 - Intersecting, 6
 - Intersects, 6
 - Plane, 5
 - PlaneIntersectionType, 6
 - Ray, 5
 - Vector3, 5
- MathHelper::BoundingBox
 - BoundingBox, 7
 - GetCorners, 8
 - Maximum, 8
 - Minimum, 8
- MathHelper::BoundingSphere
 - BoundingSphere, 9
 - Center, 9
 - Radius, 9
- MathHelper::CollisionHelper
 - BoxContainsBox, 12
 - BoxContainsPoint, 12
 - BoxContainsSphere, 12
 - BoxContainsTriangle, 12
 - BoxIntersectsBox, 13
 - BoxIntersectsSphere, 13
 - BoxIntersectsTriangle, 13
 - ClosestPointOnBoxToPoint, 13
 - ClosestPointOnPlaneToPoint, 14
 - ClosestPointOnSegmentToPoint, 14
 - ClosestPointOnSphereToPoint, 14
 - ClosestPointOnSphereToSphere, 14
 - ClosestPointOnTriangleToPoint, 15
 - DistanceBoxBox, 15
 - DistanceBoxPoint, 15
 - DistancePlanePoint, 15
 - DistanceSpherePoint, 16
 - DistanceSphereSphere, 16
 - PlaneIntersectsBox, 16
 - PlaneIntersectsPlane, 16, 17
 - PlaneIntersectsPoint, 17
 - PlaneIntersectsSphere, 17
 - PlaneIntersectsTriangle, 18
 - RayIntersectsBox, 18
 - RayIntersectsPlane, 18, 19
 - RayIntersectsPoint, 19
 - RayIntersectsRay, 19
 - RayIntersectsSphere, 19–21
 - RayIntersectsTriangle, 21
 - SphereContainsBox, 22
 - SphereContainsPoint, 22
 - SphereContainsSphere, 22
 - SphereContainsTriangle, 22
 - SphereIntersectsSphere, 23
 - SphereIntersectsTriangle, 23
 - SupportPoint, 23, 24
- MathHelper::FloatHelper
 - PackHalf, 25
 - UnpackHalf, 25
- MathHelper::Vector2Helper
 - Abs, 28
 - Add, 28
 - Barycentric, 29
 - CatmullRom, 29, 30
 - Clamp, 30
 - Cos, 30, 31
 - Distance, 31
 - DistanceSquared, 31, 32
 - Divide, 32
 - Dot, 32, 33
 - Exp, 33
 - Hermite, 33, 34
 - Lerp, 34
 - Max, 34, 35
 - Min, 35
 - Modulate, 35, 36
 - Multiply, 36
 - Negate, 36
 - Normalize, 37
 - Orthogonalize, 37
 - Orthonormalize, 37
 - Perp, 38
 - PerpDot, 38, 39
 - Reciprocal, 39
 - ReciprocalSqrt, 39
 - Reflect, 40
 - Refract, 40
 - Sin, 41
 - SinCos, 41
 - SmoothStep, 41, 42
 - Sqrt, 42
 - Subtract, 42

- Tan, 43
- MathHelper::Vector3Helper
 - Abs, 46
 - Add, 47
 - Barycentric, 47
 - CatmullRom, 48
 - Clamp, 48
 - Cos, 49
 - Cross, 49
 - Distance, 50
 - DistanceSquared, 50
 - Divide, 51
 - Dot, 51
 - Exp, 52
 - Hermite, 52
 - Lerp, 53
 - Max, 53, 54
 - Min, 54
 - Modulate, 54
 - Multiply, 55
 - Negate, 55
 - Normalize, 55, 56
 - Orthogonalize, 56
 - Orthonormalize, 56
 - Reciprocal, 57
 - ReciprocalSqrt, 57
 - Reflect, 58
 - Refract, 58
 - Sin, 59
 - SinCos, 59
 - SmoothStep, 59
 - Sqrt, 60
 - Subtract, 60
 - Tan, 60, 61
 - TripleProduct, 61
- MathHelper::Vector4Helper
 - Abs, 64
 - Add, 64, 65
 - Barycentric, 65
 - CatmullRom, 65, 66
 - Clamp, 66
 - Cos, 66, 67
 - Distance, 67
 - DistanceSquared, 67, 68
 - Divide, 68
 - Dot, 68, 69
 - Exp, 69
 - Hermite, 69, 70
 - Lerp, 70
 - Max, 70, 71
 - Min, 71
 - Modulate, 71, 72
 - Multiply, 72
 - Negate, 72
 - Normalize, 73
 - Orthogonalize, 73
 - Orthonormalize, 73
 - Reciprocal, 74
 - ReciprocalSqrt, 74
 - Sin, 75
 - SinCos, 75
 - SmoothStep, 75, 76
 - Sqrt, 76
 - Subtract, 76
 - Tan, 77
- Max
 - MathHelper::Vector2Helper, 34, 35
 - MathHelper::Vector3Helper, 53, 54
 - MathHelper::Vector4Helper, 70, 71
- Maximum
 - MathHelper::BoundingBox, 8
- Min
 - MathHelper::Vector2Helper, 35
 - MathHelper::Vector3Helper, 54
 - MathHelper::Vector4Helper, 71
- Minimum
 - MathHelper::BoundingBox, 8
- Modulate
 - MathHelper::Vector2Helper, 35, 36
 - MathHelper::Vector3Helper, 54
 - MathHelper::Vector4Helper, 71, 72
- Multiply
 - MathHelper::Vector2Helper, 36
 - MathHelper::Vector3Helper, 55
 - MathHelper::Vector4Helper, 72
- Negate
 - MathHelper::Vector2Helper, 36
 - MathHelper::Vector3Helper, 55
 - MathHelper::Vector4Helper, 72
- Normalize
 - MathHelper::Vector2Helper, 37
 - MathHelper::Vector3Helper, 55, 56
 - MathHelper::Vector4Helper, 73
- Orthogonalize
 - MathHelper::Vector2Helper, 37
 - MathHelper::Vector3Helper, 56
 - MathHelper::Vector4Helper, 73
- Orthonormalize
 - MathHelper::Vector2Helper, 37
 - MathHelper::Vector3Helper, 56
 - MathHelper::Vector4Helper, 73
- PackHalf
 - MathHelper::FloatHelper, 25
- Perp
 - MathHelper::Vector2Helper, 38
- PerpDot
 - MathHelper::Vector2Helper, 38, 39
- Plane
 - MathHelper, 5
- PlaneIntersectionType
 - MathHelper, 6
- PlaneIntersectsBox
 - MathHelper::CollisionHelper, 16
- PlaneIntersectsPlane

- MathHelper::CollisionHelper, 16, 17
- PlaneIntersectsPoint
 - MathHelper::CollisionHelper, 17
- PlaneIntersectsSphere
 - MathHelper::CollisionHelper, 17
- PlaneIntersectsTriangle
 - MathHelper::CollisionHelper, 18
- Radius
 - MathHelper::BoundingSphere, 9
- Ray
 - MathHelper, 5
- RayIntersectsBox
 - MathHelper::CollisionHelper, 18
- RayIntersectsPlane
 - MathHelper::CollisionHelper, 18, 19
- RayIntersectsPoint
 - MathHelper::CollisionHelper, 19
- RayIntersectsRay
 - MathHelper::CollisionHelper, 19
- RayIntersectsSphere
 - MathHelper::CollisionHelper, 19–21
- RayIntersectsTriangle
 - MathHelper::CollisionHelper, 21
- Reciprocal
 - MathHelper::Vector2Helper, 39
 - MathHelper::Vector3Helper, 57
 - MathHelper::Vector4Helper, 74
- ReciprocalSqrt
 - MathHelper::Vector2Helper, 39
 - MathHelper::Vector3Helper, 57
 - MathHelper::Vector4Helper, 74
- Reflect
 - MathHelper::Vector2Helper, 40
 - MathHelper::Vector3Helper, 58
- Refract
 - MathHelper::Vector2Helper, 40
 - MathHelper::Vector3Helper, 58
- Sin
 - MathHelper::Vector2Helper, 41
 - MathHelper::Vector3Helper, 59
 - MathHelper::Vector4Helper, 75
- SinCos
 - MathHelper::Vector2Helper, 41
 - MathHelper::Vector3Helper, 59
 - MathHelper::Vector4Helper, 75
- SmoothStep
 - MathHelper::Vector2Helper, 41, 42
 - MathHelper::Vector3Helper, 59
 - MathHelper::Vector4Helper, 75, 76
- SphereContainsBox
 - MathHelper::CollisionHelper, 22
- SphereContainsPoint
 - MathHelper::CollisionHelper, 22
- SphereContainsSphere
 - MathHelper::CollisionHelper, 22
- SphereContainsTriangle
 - MathHelper::CollisionHelper, 22
- SphereIntersectsSphere
 - MathHelper::CollisionHelper, 23
- SphereIntersectsTriangle
 - MathHelper::CollisionHelper, 23
- Sqrt
 - MathHelper::Vector2Helper, 42
 - MathHelper::Vector3Helper, 60
 - MathHelper::Vector4Helper, 76
- Subtract
 - MathHelper::Vector2Helper, 42
 - MathHelper::Vector3Helper, 60
 - MathHelper::Vector4Helper, 76
- SupportPoint
 - MathHelper::CollisionHelper, 23, 24
- Tan
 - MathHelper::Vector2Helper, 43
 - MathHelper::Vector3Helper, 60, 61
 - MathHelper::Vector4Helper, 77
- TripleProduct
 - MathHelper::Vector3Helper, 61
- UnpackHalf
 - MathHelper::FloatHelper, 25
- Vector2Helper, 25
- Vector3
 - MathHelper, 5
- Vector3Helper, 43
- Vector4Helper, 61