Math Helper

1.0

Generated by Doxygen 1.8.3

Thu Jan 3 2013 11:22:54

Contents

Chapter 1

Namespace Index

1.1 Packages

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

Chapter 2

Class Index

2.1 Class List

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

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

minimum	The minimum vertex of the bounding box.
maximum	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

minimumX	The minimum x-coordinate of the bounding box.
minimumY	The minimum y-coordinate of the bounding box.
minimumZ	The minimum z-coordinate of the bounding box.
maximumX	The maximum x-coordinate of the bounding box.
maximumY	The maximum y-coordinate of the bounding box.
maximumZ	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 radious 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

center	The center of the sphere.
radius	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

centerX	The x-coordinate for the center of the sphere.
centerY	The y-coordinate for the center of the sphere.
centerZ	The z-coordinate for the center of the sphere.
radius	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 radious 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

box1	The first box to test.
box2	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

box	The box to test.
point	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

box	The box to test.
sphere	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

box	The box to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	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

box1	The first box to test.
box2	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

box	The box to test.
sphere	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

box	The box to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	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

box	The box to test.
point	The point to test.
result	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

plane	The plane to test.
point	The point to test.
result	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

point	The point to test.
segment1	The starting point of the segment to test.
segment2	The ending point of the segment to test.
result	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

sphere	
point	The point to test.
result	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

sphere1	The first sphere to test.
sphere2	The second sphere to test.
result	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.

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

point	The point to test.
vertex1	The first vertex to test.
vertex2	The second vertex to test.
vertex3	The third vertex to test.
result	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

box1	The first box to test.
box2	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

box	The box to test.
point	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

plane	The plane to test.
point	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

sphere	The sphere to test.
point	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

sphere1	The first sphere to test.
sphere2	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

plane	The plane to test.
box	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

plane1	The first plane to test.
plane2	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

plane1	The first plane to test.
plane2	The second plane to test.
line	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

plane	The plane to test.
point	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

plane	The plane to test.
sphere	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

plane	The plane to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	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

ray	The ray to test.
box	The box to test.
distance	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

ray	The ray to test.
box	The box to test.
point	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

ray	The ray to test.
plane	The plane to test.
distance	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

ray	The ray to test.
plane	The plane to test
point	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

ray	The ray to test.
point	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

ray1	The first ray to test.
ray2	The second ray to test.
point	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 BoundingSphere *sphere,* out float *distance*) [static]

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

Parameters

ray	The ray to test.
sphere	The sphere to test.
distance	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 BoundingSphere *sphere,* out Vector3 *point*) [static]

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

Parameters

ray	The ray to test.
sphere	The sphere to test.
point	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 BoundingSphere *sphere,* out Vector3 *point,* out Vector3 *normal*) [static]

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

Parameters

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

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

ray	The ray to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	The third vertex of the triangle to test.
distance	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

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

Returns

Whether the two objects intersected.

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

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

Parameters

sphere	The sphere to test.
box	The box to test.

Returns

The type of containment the two objects have.

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

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

Parameters

sphere	The sphere to test.
point	The point to test.

Returns

The type of containment the two objects have.

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

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

Parameters

sphere1	The first sphere to test.
sphere2	The second sphere to test.

Returns

The type of containment the two objects have.

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

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

Parameters

sphere	The sphere to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	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 BoundingSphere *sphere1,* ref BoundingSphere *sphere2*) [static]

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

Parameters

sphere1	First sphere to test.
sphere2	Second sphere to test.

Returns

Whether the two objects intersected.

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

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

Parameters

sphere	The sphere to test.
vertex1	The first vertex of the triangle to test.
vertex2	The second vertex of the triagnle to test.
vertex3	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

vertex1	The first vertex of the triangle.
vertex2	The second vertex of the triangle.
vertex3	The third vertex of the triangle
direction	The direction for which to build the supporting point.
result	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

box	The box to generate the supporting point for.
direction	The direction for which to build the supporting point.
result	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

sphere	The sphere to generate the supporting point for.
direction	The direction for which to build the supporting point.
result	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

points	The points that make up the polyhedra.
direction	The direction for which to build the supporting point.
result	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

value	The float to pack.

Returns

The half represntation of the float.

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

Unpacks the specified half.

Parameters

value	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 than 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 than 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 than 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

value	The vector to take the absolute value of.
result	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

value	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

left	The first vector to add.
right	The second vector to add.
result	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

left	The first vector to add.
right	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

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

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

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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	Weighting factor.
result	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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	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

value	The value to clamp.
min	The minimum value.
max	The maximum value.
result	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

value	The value to clamp.
min	The minimum value.
max	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

value	The vector to take the cosine of.
result	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

value	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

value1	The first vector.
value2	The second vector.
result	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

value1	The first vector.
value2	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

value1	The first vector.
value2	The second vector
result	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

value1	The first vector.
value2	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

left	First source vector.
right	Second source vector.
result	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

left	First source vector.
right	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

value	The value to take e raised to each component of.
result	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

value	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

value1	First source position vector.
tangent1	First source tangent vector.
value2	Second source position vector.
tangent2	Second source tangent vector.
amount	Weighting factor.
result	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

value1	First source position vector.
tangent1	First source tangent vector.
value2	Second source position vector.
tangent2	Second source tangent vector.
amount	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .
result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

left	The first vector to modulate.
right	The second vector to modulate.
result	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

left	The first vector to modulate.
right	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

value	The vector to negate.
result	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

value	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

value	The vector to normalize.
result	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

value	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

destination	The list of orthogonalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

Orthonormalizes a list of vectors.

Parameters

destination	The list of orthonormalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

value	The vector to base the perpendicular vector on.
result	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

value	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

left	First source vector.
right	Second source vector.
result	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

left	First source vector.
right	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

value	The vector to take the reciprocal of.
result	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

value	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal of.
result	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal 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

vector	The source vector.
normal	Normal of the surface.
result	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

vector	The source vector.
normal	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

vector	The source vector.
normal	Normal of the surface.
index	Index of refraction.
result	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

vector	The source vector.
normal	Normal of the surface.
index	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

value	The vector to take the sine of.
result	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

value	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 than the cosine of each component in the vector.

Parameters

value	The vector to take the sine and cosine of.
sinResult	When the method completes, contains the sine of each component in the input vector.
cosResult	When the method completes, contains the cpsome pf 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

start	Start vector.

end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .
result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value	The vector to take the square root of.
result	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

value	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

left	The first vector to subtract.
right	The second vector to subtract.
result	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

left	The first vector to subtract.
right	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

value	The vector to take the tangent of.
result	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

value	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 than 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 than 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 than 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

value	The vector to take the absolute value of.
result	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

value	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

left	The first vector to add.
right	The second vector to add.
result	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

left	The first vector to add.
right	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

value1	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 1 of the triangle.
value2	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 2 of the triangle.
value3	A UnityEngine.Vector3 containing the 3D Cartesian coordinates of vertex 3 of the triangle.
amount1	Barycentric coordinate b2, which expresses the weighting factor toward vertex 2 (specified in *value2*).
amount2	Barycentric coordinate b3, which expresses the weighting factor toward vertex 3 (specified in *value3*).
result	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

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

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

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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	Weighting factor.
result	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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	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

value	The value to clamp.
min	The minimum value.
max	The maximum value.
result	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

value	The value to clamp.
min	The minimum value.
max	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

value	The vector to take the cosine of.
result	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

value	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

left	First source vector.
right	Second source vector.
result	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

left	First source vector.
right	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

value1	The first vector.
value2	The second vector.
result	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

value1	The first vector.
value2	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

value1	The first vector.
value2	The second vector.
result	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

value1	The first vector.
value2	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

left	First source vector.
right	Second source vector.
result	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

left	First source vector.
right	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

value	The value to take e raised to each component of.
result	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

value	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

value1	First source position vector.
tangent1	First source tangent vector.
value2	Second source position vector.
tangent2	Second source tangent vector.
amount	Weighting factor.
result	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

value1	First source position vector.
tangent1	First source tangent vector.

value2	Second source position vector.
tangent2	Second source tangent vector.
amount	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .
result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

left	The first vector to modulate.
right	The second vector to modulate.
result	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

left	The first vector to modulate.
right	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

value	The vector to negate.
result	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

value	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

value	The vector to normalize.
result	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

value	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

destination	The list of orthogonalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

Orthonormalizes a list of vectors.

Parameters

destination	The list of orthonormalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

Takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the reciprocal of.
result	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

value	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal of.
result	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal 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

vector	The source vector.
normal	Normal of the surface.
result	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

vector	The source vector.
normal	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

vector	The source vector.
normal	Normal of the surface.
index	Index of refraction.
result	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

vector	The source vector.
normal	Normal of the surface.
index	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

value	The vector to take the sine of.
result	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

value	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 than the cosine of each component in the vector.

Parameters

value	The vector to take the sine and cosine of.
sinResult	When the method completes, contains the sine of each component in the input vector.
cosResult	When the method completes, contains the cpsome pf 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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .
result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value	The vector to take the square root of.
result	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

value	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

left	The first vector to subtract.
right	The second vector to subtract.
result	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

left	The first vector to subtract.
right	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

value	The vector to take the tangent of.
result	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

value	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

value1	First source vector.
value2	Second source vector.
value3	Third source vector.
result	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

value1	First source vector.
value2	Second source vector.
value3	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 than 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 than 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 than 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

value	The vector to take the absolute value of.
result	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

value	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

left	The first vector to add.
right	The second vector to add.
result	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

left	The first vector to add.
right	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

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

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

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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	Weighting factor.
result	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

value1	The first position in the interpolation.
value2	The second position in the interpolation.
value3	The third position in the interpolation.
value4	The fourth position in the interpolation.
amount	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

value	The value to clamp.
min	The minimum value.
max	The maximum value.
result	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

value	The value to clamp.
min	The minimum value.
max	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

value	The vector to take the cosine of.
result	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

value	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

value1	The first vector.
value2	The second vector.
result	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

value1	The first vector.
value2	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

value1	The first vector.
value2	The second vector.
result	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

value1	The first vector.
value2	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

left	First source vector
right	Second source vector.
result	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

left	First source vector.
right	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

value	The value to take e raised to each component of.
result	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

value	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

value1	First source position vector.
tangent1	First source tangent vector.
value2	Second source position vector.
tangent2	Second source tangent vector.
amount	Weighting factor.
result	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

value1	First source position vector.
tangent1	First source tangent vector.
value2	Second source position vector.
tangent2	Second source tangent vector.
amount	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .
result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

value1	The first source vector.
value2	The second source vector.
result	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

value1	The first source vector.
value2	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

left	The first vector to modulate.
right	The second vector to modulate.
result	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

left	The first vector to modulate.
right	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

value	The vector to scale.
scalar	The amount by which to scale the vector.
result	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

value	The vector to scale.
scalar	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

value	The vector to negate.
result	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

value	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

value	The vector to normalize.
result	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

value	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

destination	The list of orthogonalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

Orthonormalizes a list of vectors.

Parameters

destination	The list of orthonormalized vectors.
source	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

ArgumentNullException	Thrown when *source* or *destination* is `null`.
ArgumentOutOfRange-Exception	Thrown when *destination* is shorter in length than *source* .

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

Takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the reciprocal of.
result	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

value	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal of.
result	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 than takes the reciprocal of each component in the vector.

Parameters

value	The vector to take the square root and recpirocal 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

value	The vector to take the sine of.
result	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

value	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 than the cosine of each component in the vector.

Parameters

value	The vector to take the sine and cosine of.
sinResult	When the method completes, contains the sine of each component in the input vector.
cosResult	When the method completes, contains the cpsome pf 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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

result	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

start	Start vector.
end	End vector.
amount	Value between 0 and 1 indicating the weight of *end* .

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

value	The vector to take the square root of.
result	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

value	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

left	The first vector to subtract.
right	The second vector to subtract.
result	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

left	The first vector to subtract.
right	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

value	The vector to take the tangent of.
result	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

value	The vector to take the tangent of.

Returns

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

Index