computeṙhino3d Documentation

Release 0.14.0

Robert McNeel & Associates

Mar 03, 2022

Contents:

CHAPTER 1

RhinoCompute.AreaMassProperties

RhinoCompute.AreaMassProperties.**compute**(*closedPlanarCurve*, *multiple=false*)

Computes an AreaMassProperties for a closed planar curve.

Arguments

- **closedPlanarCurve** (*rhino3dm.Curve*) – Curve to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given curve or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute1**(*closedPlanarCurve*, *planarTolerance*, *multiple=false*)

Computes an AreaMassProperties for a closed planar curve.

Arguments

- **closedPlanarCurve** (*rhino3dm.Curve*) – Curve to measure.
- **planarTolerance** (*float*) – absolute tolerance used to insure the closed curve is planar
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given curve or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute2**(*hatch*, *multiple=false*)

Computes an AreaMassProperties for a hatch.

Arguments

- **hatch** (*Hatch*) – Hatch to measure.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given hatch or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute3**(*mesh*, *multiple=false*)

Computes an AreaMassProperties for a mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Mesh or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute4**(*mesh*, *area*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the AreaMassProperties for a single Mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to measure.
- **area** (*bool*) – True to calculate area.
- **firstMoments** (*bool*) – True to calculate area first moments, area, and area centroid.
- **secondMoments** (*bool*) – True to calculate area second moments.
- **productMoments** (*bool*) – True to calculate area product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Mesh or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute5**(*brep*, *multiple=false*)

Computes an AreaMassProperties for a brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Brep or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute6**(*brep*, *area*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the AreaMassProperties for a single Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **area** (*bool*) – True to calculate area.
- **firstMoments** (*bool*) – True to calculate area first moments, area, and area centroid.

- **secondMoments** (*bool*) – True to calculate area second moments.
- **productMoments** (*bool*) – True to calculate area product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Brep or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute7**(*brep*, *area*, *firstMoments*, *secondMoments*, *productMoments*, *relativeTolerance*, *absoluteTolerance*, *multiple=false*)

Compute the AreaMassProperties for a single Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **area** (*bool*) – True to calculate area.
- **firstMoments** (*bool*) – True to calculate area first moments, area, and area centroid.
- **secondMoments** (*bool*) – True to calculate area second moments.
- **productMoments** (*bool*) – True to calculate area product moments.
- **relativeTolerance** (*float*) – The relative tolerance used for the calculation. In overloads of this function where tolerances are not specified, 1.0e-6 is used.
- **absoluteTolerance** (*float*) – The absolute tolerancwe used for the calculation. In overloads of this function where tolerances are not specified, 1.0e-6 is used.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Brep or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute8**(*surface*, *multiple=false*)

Computes an AreaMassProperties for a surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Surface or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute9**(*surface*, *area*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the AreaMassProperties for a single Surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to measure.
- **area** (*bool*) – True to calculate area.
- **firstMoments** (*bool*) – True to calculate area first moments, area, and area centroid.
- **secondMoments** (*bool*) – True to calculate area second moments.

- **productMoments** (*bool*) – True to calculate area product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the given Surface or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute10**(*geometry*, *multiple=false*)

Computes the Area properties for a collection of geometric objects. At present only Breps, Surfaces, Meshes and Planar Closed Curves are supported.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – Objects to include in the area computation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The Area properties for the entire collection or None on failure.

Return type AreaMassProperties

RhinoCompute.AreaMassProperties.**compute11**(*geometry*, *area*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Computes the AreaMassProperties for a collection of geometric objects. At present only Breps, Surfaces, Meshes and Planar Closed Curves are supported.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – Objects to include in the area computation.
- **area** (*bool*) – True to calculate area.
- **firstMoments** (*bool*) – True to calculate area first moments, area, and area centroid.
- **secondMoments** (*bool*) – True to calculate area second moments.
- **productMoments** (*bool*) – True to calculate area product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The AreaMassProperties for the entire collection or None on failure.

Return type AreaMassProperties

CHAPTER 2

RhinoCompute.BezierCurve

RhinoCompute.BezierCurve.**createCubicBeziers**(*sourceCurve*, *distanceTolerance*, *kinkTolerance*, *multiple=false*)

Constructs an array of cubic, non-rational Beziers that fit a curve to a tolerance.

Arguments

- **sourceCurve** (*rhino3dm.Curve*) – A curve to approximate.
- **distanceTolerance** (*float*) – The max fitting error. Use RhinoMath.SqrtEpsilon as a minimum.
- **kinkTolerance** (*float*) – If the input curve has a g1-discontinuity with angle radian measure greater than kinkTolerance at some point P, the list of beziers will also have a kink at P.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of bezier curves. The array can be empty and might contain None items.

Return type rhino3dm.BezierCurve[]

RhinoCompute.BezierCurve.**createBeziers**(*sourceCurve*, *multiple=false*)

Create an array of Bezier curves that fit to an existing curve. Please note, these Beziers can be of any order and may be rational.

Arguments

- **sourceCurve** (*rhino3dm.Curve*) – The curve to fit Beziers to
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Bezier curves

Return type rhino3dm.BezierCurve[]

CHAPTER 3

RhinoCompute.Brep

RhinoCompute.Brep.**isBox**(*thisBrep*, *multiple=false*)

Verifies a Brep is in the form of a solid box.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the Brep is a solid box, False otherwise.

Return type bool

RhinoCompute.Brep.**isBox1**(*thisBrep*, *tolerance*, *multiple=false*)

Verifies a Brep is in the form of a solid box.

Arguments

- **tolerance** (*float*) – The tolerance used to determine if faces are planar and to compare face normals.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the Brep is a solid box, False otherwise.

Return type bool

RhinoCompute.Brep.**changeSeam**(*face*, *direction*, *parameter*, *tolerance*, *multiple=false*)

Change the seam of a closed trimmed surface.

Arguments

- **face** (*rhino3dm.BrepFace*) – A Brep face with a closed underlying surface.
- **direction** (*int*) – The parameter direction (0 = U, 1 = V). The face's underlying surface must be closed in this direction.
- **parameter** (*float*) – The parameter at which to place the seam.
- **tolerance** (*float*) – Tolerance used to cut up surface.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new Brep that has the same geometry as the face with a relocated seam if successful, or None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**copyTrimCurves**(*trimSource*, *surfaceSource*, *tolerance*, *multiple=false*)

Copy all trims from a Brep face onto a surface.

Arguments

- **trimSource** (*rhino3dm.BrepFace*) – Brep face which defines the trimming curves.
- **surfaceSource** (*rhino3dm.Surface*) – The surface to trim.
- **tolerance** (*float*) – Tolerance to use for rebuilding 3D trim curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep with the shape of surfaceSource and the trims of trimSource or None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createBaseballSphere**(*center*, *radius*, *tolerance*, *multiple=false*)

Creates a brep representation of the sphere with two similar trimmed NURBS surfaces, and no singularities.

Arguments

- **center** (*rhino3dm.Point3d*) – The center of the sphere.
- **radius** (*float*) – The radius of the sphere.
- **tolerance** (*float*) – Used in computing 2d trimming curves. If >= 0.0, then the max of ON_0.0001 * radius and RhinoMath.ZeroTolerance will be used.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new brep, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createDevelopableLoft**(*crv0*, *crv1*, *reverse0*, *reverse1*, *density*, *multiple=false*)

Creates a single developable surface between two curves.

Arguments

- **crv0** (*rhino3dm.Curve*) – The first rail curve.
- **crv1** (*rhino3dm.Curve*) – The second rail curve.
- **reverse0** (*bool*) – Reverse the first rail curve.
- **reverse1** (*bool*) – Reverse the second rail curve
- **density** (*int*) – The number of rulings across the surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The output Breps if successful, otherwise an empty array.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createDevelopableLoft1**(*rail0*, *rail1*, *fixedRulings*, *multiple=false*)
Creates a single developable surface between two curves.

Arguments

- **rail0** (*rhino3dm.NurbsCurve*) – The first rail curve.
- **rail1** (*rhino3dm.NurbsCurve*) – The second rail curve.
- **fixedRulings** (*list[rhino3dm.Point2d]*) – Rulings define lines across the surface that define the straight sections on the developable surface, where rulings[i].X = parameter on first rail curve, and rulings[i].Y = parameter on second rail curve. Note, rulings will be automatically adjusted to minimum twist.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The output Breps if successful, otherwise an empty array.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarBreps**(*inputLoops*, *multiple=false*)
Constructs a set of planar breps as outlines by the loops.

Arguments

- **inputLoops** (*list[rhino3dm.Curve]*) – Curve loops that delineate the planar boundaries.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarBreps1**(*inputLoops*, *tolerance*, *multiple=false*)
Constructs a set of planar breps as outlines by the loops.

Arguments

- **inputLoops** (*list[rhino3dm.Curve]*) – Curve loops that delineate the planar boundaries.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarBreps2**(*inputLoop*, *multiple=false*)
Constructs a set of planar breps as outlines by the loops.

Arguments

- **inputLoop** (*rhino3dm.Curve*) – A curve that should form the boundaries of the surfaces or polysurfaces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarBreps3**(*inputLoop*, *tolerance*, *multiple=false*)

Constructs a set of planar breps as outlines by the loops.

Arguments

- **inputLoop** (*rhino3dm.Curve*) – A curve that should form the boundaries of the surfaces or polysurfaces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createTrimmedSurface**(*trimSource*, *surfaceSource*, *multiple=false*)

Constructs a Brep using the trimming information of a brep face and a surface. Surface must be roughly the same shape and in the same location as the trimming brep face.

Arguments

- **trimSource** (*rhino3dm.BrepFace*) – BrepFace which contains trimmingSource brep.
- **surfaceSource** (*rhino3dm.Surface*) – Surface that trims of BrepFace will be applied to.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep with the shape of surfaceSource and the trims of trimSource or None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createTrimmedSurface1**(*trimSource*, *surfaceSource*, *tolerance*, *multiple=false*)

Constructs a Brep using the trimming information of a brep face and a surface. Surface must be roughly the same shape and in the same location as the trimming brep face.

Arguments

- **trimSource** (*rhino3dm.BrepFace*) – BrepFace which contains trimmingSource brep.
- **surfaceSource** (*rhino3dm.Surface*) – Surface that trims of BrepFace will be applied to.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep with the shape of surfaceSource and the trims of trimSource or None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createFromCornerPoints**(*corner1*, *corner2*, *corner3*, *tolerance*, *multiple=false*)

Makes a Brep with one face from three corner points.

Arguments

- **corner1** (*rhino3dm.Point3d*) – A first corner.
- **corner2** (*rhino3dm.Point3d*) – A second corner.
- **corner3** (*rhino3dm.Point3d*) – A third corner.

- **tolerance** (*float*) – Minimum edge length allowed before collapsing the side into a singularity.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A boundary representation, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createFromCornerPoints1**(*corner1*, *corner2*, *corner3*, *corner4*, *tolerance*, *multiple=false*)

Makes a Brep with one face from four corner points.

Arguments

- **corner1** (*rhino3dm.Point3d*) – A first corner.
- **corner2** (*rhino3dm.Point3d*) – A second corner.
- **corner3** (*rhino3dm.Point3d*) – A third corner.
- **corner4** (*rhino3dm.Point3d*) – A fourth corner.
- **tolerance** (*float*) – Minimum edge length allowed before collapsing the side into a singularity.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A boundary representation, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createEdgeSurface**(*curves*, *multiple=false*)

Constructs a coons patch from 2, 3, or 4 curves.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – A list, an array or any enumerable set of curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns resulting brep or None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createPlanarBreps4**(*inputLoops*, *multiple=false*)

Constructs a set of planar Breps as outlines by the loops.

Arguments

- **inputLoops** (*Rhino.Collections.CurveList*) – Curve loops that delineate the planar boundaries.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps or None on error.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarBreps5**(*inputLoops*, *tolerance*, *multiple=false*)

Constructs a set of planar Breps as outlines by the loops.

Arguments

- **inputLoops** (*Rhino.Collections.CurveList*) – Curve loops that delineate the planar boundaries.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Planar Breps.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromOffsetFace**(*face*, *offsetDistance*, *offsetTolerance*, *bothSides*, *createSolid*, *multiple=false*)

Offsets a face including trim information to create a new brep.

Arguments

- **face** (*rhino3dm.BrepFace*) – the face to offset.
- **offsetDistance** (*float*) – An offset distance.
- **offsetTolerance** (*float*) – Use 0.0 to make a loose offset. Otherwise, the document's absolute tolerance is usually sufficient.
- **bothSides** (*bool*) – When true, offset to both sides of the input face.
- **createSolid** (*bool*) – When true, make a solid object.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new brep if successful. The brep can be disjoint if bothSides is True and createSolid is false, or if createSolid is True and connecting the offsets with side surfaces fails. None if unsuccessful.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createSolid**(*breps*, *tolerance*, *multiple=false*)

Constructs closed polysurfaces from surfaces and polysurfaces that bound a region in space.

Arguments

- **breps** (*list[rhino3dm.Brep]*) – The intersecting surfaces and polysurfaces to automatically trim and join into closed polysurfaces.
- **tolerance** (*float*) – The trim and join tolerance. If set to RhinoMath.UnsetValue, Rhino's global absolute tolerance is used.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting polysurfaces on success or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**mergeSurfaces**(*surface0*, *surface1*, *tolerance*, *angleToleranceRadians*, *multiple=false*)

Merges two surfaces into one surface at untrimmed edges.

Arguments

- **surface0** (*rhino3dm.Surface*) – The first surface to merge.
- **surface1** (*rhino3dm.Surface*) – The second surface to merge.
- **tolerance** (*float*) – Surface edges must be within this tolerance for the two surfaces to merge.

- **angleToleranceRadians** (*float*) – Edge must be within this angle tolerance in order for contiguous edges to be combined into a single edge.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The merged surfaces as a Brep if successful, None if not successful.

Return type rhino3dm.Brep

RhinoCompute.Brep.**mergeSurfaces1**(*brep0*, *brep1*, *tolerance*, *angleToleranceRadians*, *multiple=false*)

Merges two surfaces into one surface at untrimmed edges. Both surfaces must be untrimmed and share an edge.

Arguments

- **brep0** (*rhino3dm.Brep*) – The first single-face Brep to merge.
- **brep1** (*rhino3dm.Brep*) – The second single-face Brep to merge.
- **tolerance** (*float*) – Surface edges must be within this tolerance for the two surfaces to merge.
- **angleToleranceRadians** (*float*) – Edge must be within this angle tolerance in order for contiguous edges to be combined into a single edge.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The merged Brep if successful, None if not successful.

Return type rhino3dm.Brep

RhinoCompute.Brep.**mergeSurfaces2**(*brep0*, *brep1*, *tolerance*, *angleToleranceRadians*, *point0*, *point1*, *roundness*, *smooth*, *multiple=false*)

Merges two surfaces into one surface at untrimmed edges. Both surfaces must be untrimmed and share an edge.

Arguments

- **brep0** (*rhino3dm.Brep*) – The first single-face Brep to merge.
- **brep1** (*rhino3dm.Brep*) – The second single-face Brep to merge.
- **tolerance** (*float*) – Surface edges must be within this tolerance for the two surfaces to merge.
- **angleToleranceRadians** (*float*) – Edge must be within this angle tolerance in order for contiguous edges to be combined into a single edge.
- **point0** (*rhino3dm.Point2d*) – 2D pick point on the first single-face Brep. The value can be unset.
- **point1** (*rhino3dm.Point2d*) – 2D pick point on the second single-face Brep. The value can be unset.
- **roundness** (*float*) – Defines the roundness of the merge. Acceptable values are between 0.0 (sharp) and 1.0 (smooth).
- **smooth** (*bool*) – The surface will be smooth. This makes the surface behave better for control point editing, but may alter the shape of both surfaces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The merged Brep if successful, None if not successful.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createPatch**(*geometry*, *startingSurface*, *tolerance*, *multiple=false*)

Constructs a brep patch. This is the simple version of fit that uses a specified starting surface.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – Combination of Curves, BrepTrims, Points, PointClouds or Meshes. Curves and trims are sampled to get points. Trims are sampled for points and normals.
- **startingSurface** (*rhino3dm.Surface*) – A starting surface (can be null).
- **tolerance** (*float*) – Tolerance used by input analysis functions for loop finding, trimming, etc.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Brep fit through input on success, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createPatch1**(*geometry*, *uSpans*, *vSpans*, *tolerance*, *multiple=false*)

Constructs a brep patch. This is the simple version of fit that uses a plane with u x v spans. It makes a plane by fitting to the points from the input geometry to use as the starting surface. The surface has the specified u and v span count.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – A combination of curves, brep trims, points, point clouds or meshes. Curves and trims are sampled to get points. Trims are sampled for points and normals.
- **uSpans** (*int*) – The number of spans in the U direction.
- **vSpans** (*int*) – The number of spans in the V direction.
- **tolerance** (*float*) – Tolerance used by input analysis functions for loop finding, trimming, etc.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep fit through input on success, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createPatch2**(*geometry*, *startingSurface*, *uSpans*, *vSpans*, *trim*, *tangency*, *pointSpacing*, *flexibility*, *surfacePull*, *fixEdges*, *tolerance*, *multiple=false*)

Constructs a brep patch using all controls

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – A combination of curves, brep trims, points, point clouds or meshes. Curves and trims are sampled to get points. Trims are sampled for points and normals.
- **startingSurface** (*rhino3dm.Surface*) – A starting surface (can be null).
- **uSpans** (*int*) – Number of surface spans used when a plane is fit through points to start in the U direction.
- **vSpans** (*int*) – Number of surface spans used when a plane is fit through points to start in the U direction.

- **trim** (*bool*) – If true, try to find an outer loop from among the input curves and trim the result to that loop
- **tangency** (*bool*) – If true, try to find brep trims in the outer loop of curves and try to fit to the normal direction of the trim's surface at those locations.
- **pointSpacing** (*float*) – Basic distance between points sampled from input curves.
- **flexibility** (*float*) – Determines the behavior of the surface in areas where its not otherwise controlled by the input. Lower numbers make the surface behave more like a stiff material; higher, less like a stiff material. That is, each span is made to more closely match the spans adjacent to it if there is no input geometry mapping to that area of the surface when the flexibility value is low. The scale is logarithmic. Numbers around 0.001 or 0.1 make the patch pretty stiff and numbers around 10 or 100 make the surface flexible.
- **surfacePull** (*float*) – Tends to keep the result surface where it was before the fit in areas where there is on influence from the input geometry
- **fixEdges** (*bool[]*) – Array of four elements. Flags to keep the edges of a starting (untrimmed) surface in place while fitting the interior of the surface. Order of flags is left, bottom, right, top
- **tolerance** (*float*) – Tolerance used by input analysis functions for loop finding, trimming, etc.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep fit through input on success, or None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createPipe**(*rail*, *radius*, *localBlending*, *cap*, *fitRail*, *absoluteTolerance*, *angleToleranceRadians*, *multiple=false*)

Creates a single walled pipe.

Arguments

- **rail** (*rhino3dm.Curve*) – The rail, or path, curve.
- **radius** (*float*) – The radius of the pipe.
- **localBlending** (*bool*) – The shape blending. If True, Local (pipe radius stays constant at the ends and changes more rapidly in the middle) is applied. If False, Global (radius is linearly blended from one end to the other, creating pipes that taper from one radius to the other) is applied.
- **cap** (*PipeCapMode*) – The end cap mode.
- **fitRail** (*bool*) – If the curve is a polycurve of lines and arcs, the curve is fit and a single surface is created; otherwise the result is a Brep with joined surfaces created from the polycurve segments.
- **absoluteTolerance** (*float*) – The sweeping and fitting tolerance. When in doubt, use the document's absolute tolerance.
- **angleToleranceRadians** (*float*) – The angle tolerance. When in doubt, use the document's angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps success.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPipe1**(*rail*, *railRadiiParameters*, *radii*, *localBlending*, *cap*, *fitRail*, *absoluteTolerance*, *angleToleranceRadians*, *multiple=false*)

Creates a single walled pipe.

Arguments

- **rail** (*rhino3dm.Curve*) – The rail, or path, curve.
- **railRadiiParameters** (*list[float]*) – One or more normalized curve parameters where changes in radius occur. Important: curve parameters must be normalized - ranging between 0.0 and 1.0. Use Interval.NormalizedParameterAt to calculate these.
- **radii** (*list[float]*) – One or more radii - one at each normalized curve parameter in railRadiiParameters.
- **localBlending** (*bool*) – The shape blending. If True, Local (pipe radius stays constant at the ends and changes more rapidly in the middle) is applied. If False, Global (radius is linearly blended from one end to the other, creating pipes that taper from one radius to the other) is applied.
- **cap** (*PipeCapMode*) – The end cap mode.
- **fitRail** (*bool*) – If the curve is a polycurve of lines and arcs, the curve is fit and a single surface is created; otherwise the result is a Brep with joined surfaces created from the polycurve segments.
- **absoluteTolerance** (*float*) – The sweeping and fitting tolerance. When in doubt, use the document's absolute tolerance.
- **angleToleranceRadians** (*float*) – The angle tolerance. When in doubt, use the document's angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps success.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createThickPipe**(*rail*, *radius0*, *radius1*, *localBlending*, *cap*, *fitRail*, *absoluteTolerance*, *angleToleranceRadians*, *multiple=false*)

Creates a double-walled pipe.

Arguments

- **rail** (*rhino3dm.Curve*) – The rail, or path, curve.
- **radius0** (*float*) – The first radius of the pipe.
- **radius1** (*float*) – The second radius of the pipe.
- **localBlending** (*bool*) – The shape blending. If True, Local (pipe radius stays constant at the ends and changes more rapidly in the middle) is applied. If False, Global (radius is linearly blended from one end to the other, creating pipes that taper from one radius to the other) is applied.
- **cap** (*PipeCapMode*) – The end cap mode.
- **fitRail** (*bool*) – If the curve is a polycurve of lines and arcs, the curve is fit and a single surface is created; otherwise the result is a Brep with joined surfaces created from the polycurve segments.
- **absoluteTolerance** (*float*) – The sweeping and fitting tolerance. When in doubt, use the document's absolute tolerance.

- **angleToleranceRadians** (*float*) – The angle tolerance. When in doubt, use the document's angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps success.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createThickPipe1**(*rail*, *railRadiiParameters*, *radii0*, *radii1*, *localBlending*, *cap*, *fitRail*, *absoluteTolerance*, *angleToleranceRadians*, *multiple=false*)

Creates a double-walled pipe.

Arguments

- **rail** (*rhino3dm.Curve*) – The rail, or path, curve.
- **railRadiiParameters** (*list[float]*) – One or more normalized curve parameters where changes in radius occur. Important: curve parameters must be normalized - ranging between 0.0 and 1.0. Use Interval.NormalizedParameterAt to calculate these.
- **radii0** (*list[float]*) – One or more radii for the first wall - one at each normalized curve parameter in railRadiiParameters.
- **radii1** (*list[float]*) – One or more radii for the second wall - one at each normalized curve parameter in railRadiiParameters.
- **localBlending** (*bool*) – The shape blending. If True, Local (pipe radius stays constant at the ends and changes more rapidly in the middle) is applied. If False, Global (radius is linearly blended from one end to the other, creating pipes that taper from one radius to the other) is applied.
- **cap** (*PipeCapMode*) – The end cap mode.
- **fitRail** (*bool*) – If the curve is a polycurve of lines and arcs, the curve is fit and a single surface is created; otherwise the result is a Brep with joined surfaces created from the polycurve segments.
- **absoluteTolerance** (*float*) – The sweeping and fitting tolerance. When in doubt, use the document's absolute tolerance.
- **angleToleranceRadians** (*float*) – The angle tolerance. When in doubt, use the document's angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps success.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep**(*rail*, *shape*, *closed*, *tolerance*, *multiple=false*)

Sweep1 function that fits a surface through a profile curve that define the surface cross-sections and one curve that defines a surface edge.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shape** (*rhino3dm.Curve*) – Shape curve
- **closed** (*bool*) – Only matters if shape is closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep1**(*rail*, *shapes*, *closed*, *tolerance*, *multiple=false*)

Sweep1 function that fits a surface through profile curves that define the surface cross-sections and one curve that defines a surface edge.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves
- **closed** (*bool*) – Only matters if shapes are closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep2**(*rail*, *shapes*, *startPoint*, *endPoint*, *frameType*, *roadlikeNormal*, *closed*, *blendType*, *miterType*, *tolerance*, *rebuildType*, *rebuildPointCount*, *refitTolerance*, *multiple=false*)

Sweep1 function that fits a surface through a series of profile curves that define the surface cross-sections and one curve that defines a surface edge.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along.
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves.
- **startPoint** (*rhino3dm.Point3d*) – Optional starting point of sweep. Use Point3d.Unset if you do not want to include a start point.
- **endPoint** (*rhino3dm.Point3d*) – Optional ending point of sweep. Use Point3d.Unset if you do not want to include an end point.
- **frameType** (*SweepFrame*) – The frame type.
- **roadlikeNormal** (*rhino3dm.Vector3d*) – The roadlike normal directoion. Use Vector3d.Unset if the frame type is not set to roadlike.
- **closed** (*bool*) – Only matters if shapes are closed.
- **blendType** (*SweepBlend*) – The shape blending type.
- **miterType** (*SweepMiter*) – The mitering type.
- **rebuildType** (*SweepRebuild*) – The rebuild style.
- **rebuildPointCount** (*int*) – If rebuild == SweepRebuild.Rebuild, the number of points. Otherwise specify 0.
- **refitTolerance** (*float*) – If rebuild == SweepRebuild.Refit, the refit tolerance. Otherwise, specify 0.0
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweepSegmented**(*rail*, *shape*, *closed*, *tolerance*, *multiple=false*)

Sweep1 function that fits a surface through a profile curve that define the surface cross-sections and one curve that defines a surface edge. The Segmented version breaks the rail at curvature kinks and sweeps each piece separately, then put the results together into a Brep.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shape** (*rhino3dm.Curve*) – Shape curve
- **closed** (*bool*) – Only matters if shape is closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweepSegmented1**(*rail*, *shapes*, *closed*, *tolerance*, *multiple=false*)

Sweep1 function that fits a surface through a series of profile curves that define the surface cross-sections and one curve that defines a surface edge. The Segmented version breaks the rail at curvature kinks and sweeps each piece separately, then put the results together into a Brep.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along.
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves.
- **closed** (*bool*) – Only matters if shapes are closed.
- **tolerance** (*float*) – Tolerance for fitting surface and rails.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweepSegmented2**(*rail*, *shapes*, *startPoint*, *endPoint*, *frameType*, *roadlikeNormal*, *closed*, *blendType*, *miterType*, *tolerance*, *rebuildType*, *rebuildPointCount*, *refitTolerance*, *multiple=false*)

Sweep1 function that fits a surface through a series of profile curves that define the surface cross-sections and one curve that defines a surface edge. The Segmented version breaks the rail at curvature kinks and sweeps each piece separately, then put the results together into a Brep.

Arguments

- **rail** (*rhino3dm.Curve*) – Rail to sweep shapes along.
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves.
- **startPoint** (*rhino3dm.Point3d*) – Optional starting point of sweep. Use Point3d.Unset if you do not want to include a start point.

- **endPoint** (*rhino3dm.Point3d*) – Optional ending point of sweep. Use Point3d.Unset if you do not want to include an end point.
- **frameType** (*SweepFrame*) – The frame type.
- **roadlikeNormal** (*rhino3dm.Vector3d*) – The roadlike normal directoion. Use Vector3d.Unset if the frame type is not set to roadlike.
- **closed** (*bool*) – Only matters if shapes are closed.
- **blendType** (*SweepBlend*) – The shape blending type.
- **miterType** (*SweepMiter*) – The mitering type.
- **rebuildType** (*SweepRebuild*) – The rebuild style.
- **rebuildPointCount** (*int*) – If rebuild == SweepRebuild.Rebuild, the number of points. Otherwise specify 0.
- **refitTolerance** (*float*) – If rebuild == SweepRebuild.Refit, the refit tolerance. Otherwise, specify 0.0
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep3**(*rail1*, *rail2*, *shape*, *closed*, *tolerance*, *multiple=false*)

General 2 rail sweep. If you are not producing the sweep results that you are after, then use the SweepTwoRail class with options to generate the swept geometry.

Arguments

- **rail1** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **rail2** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shape** (*rhino3dm.Curve*) – Shape curve
- **closed** (*bool*) – Only matters if shape is closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep4**(*rail1*, *rail2*, *shapes*, *closed*, *tolerance*, *multiple=false*)

General 2 rail sweep. If you are not producing the sweep results that you are after, then use the SweepTwoRail class with options to generate the swept geometry.

Arguments

- **rail1** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **rail2** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves
- **closed** (*bool*) – Only matters if shapes are closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweep5**(*rail1*, *rail2*, *shapes*, *start*, *end*, *closed*, *tolerance*, *rebuild*, *rebuildPointCount*, *refitTolerance*, *preserveHeight*, *multiple=false*)

Sweep2 function that fits a surface through profile curves that define the surface cross-sections and two curves that defines the surface edges.

Arguments

- **rail1** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **rail2** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves
- **start** (*rhino3dm.Point3d*) – Optional starting point of sweep. Use Point3d.Unset if you do not want to include a start point.
- **end** (*rhino3dm.Point3d*) – Optional ending point of sweep. Use Point3d.Unset if you do not want to include an end point.
- **closed** (*bool*) – Only matters if shapes are closed.
- **tolerance** (*float*) – Tolerance for fitting surface and rails.
- **rebuild** (*SweepRebuild*) – The rebuild style.
- **rebuildPointCount** (*int*) – If rebuild == SweepRebuild.Rebuild, the number of points. Otherwise specify 0.
- **refitTolerance** (*float*) – If rebuild == SweepRebuild.Refit, the refit tolerance. Otherwise, specify 0.0
- **preserveHeight** (*bool*) – Removes the association between the height scaling from the width scaling
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromSweepInParts**(*rail1*, *rail2*, *shapes*, *rail_params*, *closed*, *tolerance*, *multiple=false*)

Makes a 2 rail sweep. Like CreateFromSweep but the result is split where parameterization along a rail changes abruptly.

Arguments

- **rail1** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **rail2** (*rhino3dm.Curve*) – Rail to sweep shapes along
- **shapes** (*list[rhino3dm.Curve]*) – Shape curves
- **rail_params** (*list[rhino3dm.Point2d]*) – Shape parameters
- **closed** (*bool*) – Only matters if shapes are closed
- **tolerance** (*float*) – Tolerance for fitting surface and rails

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Brep sweep results

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromTaperedExtrude**(*curveToExtrude*, *distance*, *direction*, *basePoint*, *draftAngleRadians*, *cornerType*, *tolerance*, *angleToleranceRadians*, *multiple=false*)

Extrude a curve to a taper making a brep (potentially more than 1)

Arguments

- **curveToExtrude** (*rhino3dm.Curve*) – the curve to extrude
- **distance** (*float*) – the distance to extrude
- **direction** (*rhino3dm.Vector3d*) – the direction of the extrusion
- **basePoint** (*rhino3dm.Point3d*) – the base point of the extrusion
- **draftAngleRadians** (*float*) – angle of the extrusion
- **tolerance** (*float*) – tolerance to use for the extrusion
- **angleToleranceRadians** (*float*) – angle tolerance to use for the extrusion
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns array of breps on success

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromTaperedExtrude1**(*curveToExtrude*, *distance*, *direction*, *basePoint*, *draftAngleRadians*, *cornerType*, *multiple=false*)

Extrude a curve to a taper making a brep (potentially more than 1)

Arguments

- **curveToExtrude** (*rhino3dm.Curve*) – the curve to extrude
- **distance** (*float*) – the distance to extrude
- **direction** (*rhino3dm.Vector3d*) – the direction of the extrusion
- **basePoint** (*rhino3dm.Point3d*) – the base point of the extrusion
- **draftAngleRadians** (*float*) – angle of the extrusion
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns array of breps on success

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromTaperedExtrudeWithRef**(*curve*, *direction*, *distance*, *draftAngle*, *plane*, *tolerance*, *multiple=false*)

Creates one or more Breps by extruding a curve a distance along an axis with draft angle.

Arguments

- **curve** (*rhino3dm.Curve*) – The curve to extrude.

- **direction** (*rhino3dm.Vector3d*) – The extrusion direction.
- **distance** (*float*) – The extrusion distance.
- **draftAngle** (*float*) – The extrusion draft angle in radians.
- **plane** (*rhino3dm.Plane*) – The end of the extrusion will be parallel to this plane, and "distance" from the plane's origin. The plane's origin is generally be a point on the curve. For planar curves, a natural choice for the plane's normal direction will be the normal direction of the curve's plane. In any case, plane.Normal = direction may make sense.
- **tolerance** (*float*) – The intersecting and trimming tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBlendSurface**(*face0*, *edge0*, *domain0*, *rev0*, *continuity0*, *face1*, *edge1*, *domain1*, *rev1*, *continuity1*, *multiple=false*)

Makes a surface blend between two surface edges.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to blend from.
- **edge0** (*rhino3dm.BrepEdge*) – First edge to blend from.
- **domain0** (*rhino3dm.Interval*) – The domain of edge0 to use.
- **rev0** (*bool*) – If false, edge0 will be used in its natural direction. If true, edge0 will be used in the reversed direction.
- **continuity0** (*BlendContinuity*) – Continuity for the blend at the start.
- **face1** (*rhino3dm.BrepFace*) – Second face to blend from.
- **edge1** (*rhino3dm.BrepEdge*) – Second edge to blend from.
- **domain1** (*rhino3dm.Interval*) – The domain of edge1 to use.
- **rev1** (*bool*) – If false, edge1 will be used in its natural direction. If true, edge1 will be used in the reversed direction.
- **continuity1** (*BlendContinuity*) – Continuity for the blend at the end.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBlendShape**(*face0*, *edge0*, *t0*, *rev0*, *continuity0*, *face1*, *edge1*, *t1*, *rev1*, *continuity1*, *multiple=false*)

Makes a curve blend between points on two surface edges. The blend will be tangent to the surfaces and perpendicular to the edges.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to blend from.
- **edge0** (*rhino3dm.BrepEdge*) – First edge to blend from.
- **t0** (*float*) – Location on first edge for first end of blend curve.

- **rev0** (*bool*) – If false, edge0 will be used in its natural direction. If true, edge0 will be used in the reversed direction.
- **continuity0** (*BlendContinuity*) – Continuity for the blend at the start.
- **face1** (*rhino3dm.BrepFace*) – Second face to blend from.
- **edge1** (*rhino3dm.BrepEdge*) – Second edge to blend from.
- **t1** (*float*) – Location on second edge for second end of blend curve.
- **rev1** (*bool*) – If false, edge1 will be used in its natural direction. If true, edge1 will be used in the reversed direction.
- **continuity1** (*BlendContinuity*) – >Continuity for the blend at the end.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The blend curve on success. None on failure

Return type rhino3dm.Curve

RhinoCompute.Brep.**createFilletSurface**(*face0*, *uv0*, *face1*, *uv1*, *radius*, *extend*, *tolerance*, *multiple=false*)

Creates a constant-radius round surface between two surfaces.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to fillet from.
- **uv0** (*rhino3dm.Point2d*) – A parameter face0 at the side you want to keep after filleting.
- **face1** (*rhino3dm.BrepFace*) – Second face to fillet from.
- **uv1** (*rhino3dm.Point2d*) – A parameter face1 at the side you want to keep after filleting.
- **radius** (*float*) – The fillet radius.
- **extend** (*bool*) – If true, then when one input surface is longer than the other, the fillet surface is extended to the input surface edges.
- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFilletSurface1**(*face0*, *uv0*, *face1*, *uv1*, *radius*, *trim*, *extend*, *tolerance*, *multiple=false*)

Creates a constant-radius round surface between two surfaces.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to fillet from.
- **uv0** (*rhino3dm.Point2d*) – A parameter face0 at the side you want to keep after filleting.
- **face1** (*rhino3dm.BrepFace*) – Second face to fillet from.

- **uv1** (*rhino3dm.Point2d*) – A parameter face1 at the side you want to keep after filleting.
- **radius** (*float*) – The fillet radius.
- **trim** (*bool*) – If true, the input faces will be trimmed, if false, the input faces will be split.
- **extend** (*bool*) – If true, then when one input surface is longer than the other, the fillet surface is extended to the input surface edges.
- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createChamferSurface**(*face0*, *uv0*, *radius0*, *face1*, *uv1*, *radius1*, *extend*, *tolerance*, *multiple=false*)

Creates a ruled surface as a bevel between two input surface edges.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to chamfer from.
- **uv0** (*rhino3dm.Point2d*) – A parameter face0 at the side you want to keep after chamfering.
- **radius0** (*float*) – The distance from the intersection of face0 to the edge of the chamfer.
- **face1** (*rhino3dm.BrepFace*) – Second face to chamfer from.
- **uv1** (*rhino3dm.Point2d*) – A parameter face1 at the side you want to keep after chamfering.
- **radius1** (*float*) – The distance from the intersection of face1 to the edge of the chamfer.
- **extend** (*bool*) – If true, then when one input surface is longer than the other, the chamfer surface is extended to the input surface edges.
- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createChamferSurface1**(*face0*, *uv0*, *radius0*, *face1*, *uv1*, *radius1*, *trim*, *extend*, *tolerance*, *multiple=false*)

Creates a ruled surface as a bevel between two input surface edges.

Arguments

- **face0** (*rhino3dm.BrepFace*) – First face to chamfer from.
- **uv0** (*rhino3dm.Point2d*) – A parameter face0 at the side you want to keep after chamfering.
- **radius0** (*float*) – The distance from the intersection of face0 to the edge of the chamfer.
- **face1** (*rhino3dm.BrepFace*) – Second face to chamfer from.

- **uv1** (*rhino3dm.Point2d*) – A parameter face1 at the side you want to keep after chamfering.
- **radius1** (*float*) – The distance from the intersection of face1 to the edge of the chamfer.
- **trim** (*bool*) – If true, the input faces will be trimmed, if false, the input faces will be split.
- **extend** (*bool*) – If true, then when one input surface is longer than the other, the chamfer surface is extended to the input surface edges.
- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFilletEdges**(*brep*, *edgeIndices*, *startRadii*, *endRadii*, *blendType*, *railType*, *tolerance*, *multiple=false*)

Fillets, chamfers, or blends the edges of a brep.

Arguments

- **brep** (*rhino3dm.Brep*) – The brep to fillet, chamfer, or blend edges.
- **edgeIndices** (*list[int]*) – An array of one or more edge indices where the fillet, chamfer, or blend will occur.
- **startRadii** (*list[float]*) – An array of starting fillet, chamfer, or blend radaii, one for each edge index.
- **endRadii** (*list[float]*) – An array of ending fillet, chamfer, or blend radaii, one for each edge index.
- **blendType** (*BlendType*) – The blend type.
- **railType** (*RailType*) – The rail type.
- **tolerance** (*float*) – The tolerance to be used to perform calculations.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createOffsetBrep**(*brep*, *distance*, *solid*, *extend*, *tolerance*, *multiple=false*)

Offsets a Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – The Brep to offset.
- **distance** (*float*) – The distance to offset. This is a signed distance value with respect to face normals and flipped faces.
- **solid** (*bool*) – If true, then the function makes a closed solid from the input and offset surfaces by lofting a ruled surface between all of the matching edges.
- **extend** (*bool*) – If true, then the function maintains the sharp corners when the original surfaces have sharps corner. If False, then the function creates fillets at sharp corners in the original surfaces.

- **tolerance** (*float*) – The offset tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful. If the function succeeds in offsetting, a single Brep will be returned. Otherwise, the array will contain the offset surfaces, outBlends will contain the set of blends used to fill in gaps (if extend is false), and outWalls will contain the set of wall surfaces that was supposed to join the offset to the original (if solid is true).

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createOffsetBrep1**(*brep*, *distance*, *solid*, *extend*, *shrink*, *tolerance*, *multiple=false*)

Offsets a Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – The Brep to offset.
- **distance** (*float*) – The distance to offset. This is a signed distance value with respect to face normals and flipped faces.
- **solid** (*bool*) – If true, then the function makes a closed solid from the input and offset surfaces by lofting a ruled surface between all of the matching edges.
- **extend** (*bool*) – If true, then the function maintains the sharp corners when the original surfaces have sharps corner. If False, then the function creates fillets at sharp corners in the original surfaces.
- **shrink** (*bool*) – If true, then the function shrinks the underlying surfaces to their face's outer boundary loop.
- **tolerance** (*float*) – The offset tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of Breps if successful. If the function succeeds in offsetting, a single Brep will be returned. Otherwise, the array will contain the offset surfaces, outBlends will contain the set of blends used to fill in gaps (if extend is false), and outWalls will contain the set of wall surfaces that was supposed to join the offset to the original (if solid is true).

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**removeFins**(*thisBrep*, *multiple=false*)

Recursively removes any Brep face with a naked edge. This function is only useful for non-manifold Breps.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False if everything is removed or if the result has any Brep edges with more than two Brep trims.

Return type bool

RhinoCompute.Brep.**createFromJoinedEdges**(*brep0*, *edgeIndex0*, *brep1*, *edgeIndex1*, *joinTolerance*, *multiple=false*)

Joins two naked edges, or edges that are coincident or close together, from two Breps.

Arguments

- **brep0** (*rhino3dm.Brep*) – The first Brep.

- **edgeIndex0** (*int*) – The edge index on the first Brep.
- **brep1** (*rhino3dm.Brep*) – The second Brep.
- **edgeIndex1** (*int*) – The edge index on the second Brep.
- **joinTolerance** (*float*) – The join tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting Brep if successful, None on failure.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createFromLoft**(*curves*, *start*, *end*, *loftType*, *closed*, *multiple=false*)

Constructs one or more Breps by lofting through a set of curves.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The curves to loft through. This function will not perform any curve sorting. You must pass in curves in the order you want them lofted. This function will not adjust the directions of open curves. Use Curve.DoDirectionsMatch and Curve.Reverse to adjust the directions of open curves. This function will not adjust the seams of closed curves. Use Curve.ChangeClosedCurveSeam to adjust the seam of closed curves.
- **start** (*rhino3dm.Point3d*) – Optional starting point of loft. Use Point3d.Unset if you do not want to include a start point.
- **end** (*rhino3dm.Point3d*) – Optional ending point of loft. Use Point3d.Unset if you do not want to include an end point.
- **loftType** (*LoftType*) – type of loft to perform.
- **closed** (*bool*) – True if the last curve in this loft should be connected back to the first one.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Constructs a closed surface, continuing the surface past the last curve around to the first curve. Available when you have selected three shape curves.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromLoftRebuild**(*curves*, *start*, *end*, *loftType*, *closed*, *rebuildPointCount*, *multiple=false*)

Constructs one or more Breps by lofting through a set of curves. Input for the loft is simplified by rebuilding to a specified number of control points.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The curves to loft through. This function will not perform any curve sorting. You must pass in curves in the order you want them lofted. This function will not adjust the directions of open curves. Use Curve.DoDirectionsMatch and Curve.Reverse to adjust the directions of open curves. This function will not adjust the seams of closed curves. Use Curve.ChangeClosedCurveSeam to adjust the seam of closed curves.
- **start** (*rhino3dm.Point3d*) – Optional starting point of loft. Use Point3d.Unset if you do not want to include a start point.
- **end** (*rhino3dm.Point3d*) – Optional ending point of lost. Use Point3d.Unset if you do not want to include an end point.

- **loftType** (*LoftType*) – type of loft to perform.
- **closed** (*bool*) – True if the last curve in this loft should be connected back to the first one.
- **rebuildPointCount** (*int*) – A number of points to use while rebuilding the curves. 0 leaves turns this parameter off.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Constructs a closed surface, continuing the surface past the last curve around to the first curve. Available when you have selected three shape curves.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromLoftRefit**(*curves*, *start*, *end*, *loftType*, *closed*, *refitTolerance*, *multiple=false*)

Constructs one or more Breps by lofting through a set of curves. Input for the loft is simplified by refitting to a specified tolerance.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The curves to loft through. This function will not perform any curve sorting. You must pass in curves in the order you want them lofted. This function will not adjust the directions of open curves. Use Curve.DoDirectionsMatch and Curve.Reverse to adjust the directions of open curves. This function will not adjust the seams of closed curves. Use Curve.ChangeClosedCurveSeam to adjust the seam of closed curves.
- **start** (*rhino3dm.Point3d*) – Optional starting point of loft. Use Point3d.Unset if you do not want to include a start point.
- **end** (*rhino3dm.Point3d*) – Optional ending point of lost. Use Point3d.Unset if you do not want to include an end point.
- **loftType** (*LoftType*) – type of loft to perform.
- **closed** (*bool*) – True if the last curve in this loft should be connected back to the first one.
- **refitTolerance** (*float*) – A distance to use in refitting, or 0 if you want to turn this parameter off.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Constructs a closed surface, continuing the surface past the last curve around to the first curve. Available when you have selected three shape curves.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createFromLoft1**(*curves*, *start*, *end*, *StartTangent*, *EndTangent*, *StartTrim*, *EndTrim*, *loftType*, *closed*, *multiple=false*)

Constructs one or more Breps by lofting through a set of curves, optionally matching start and end tangents of surfaces when first and/or last loft curves are surface edges

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The curves to loft through. This function will not perform any curve sorting. You must pass in curves in the order you want them lofted. This function will not adjust the directions of open curves. Use Curve.DoDirectionsMatch and Curve.Reverse to adjust the directions of open curves. This function will not adjust the

seams of closed curves. Use Curve.ChangeClosedCurveSeam to adjust the seam of closed curves.

- **start** (*rhino3dm.Point3d*) – Optional starting point of loft. Use Point3d.Unset if you do not want to include a start point. "start" and "StartTangent" cannot both be true.
- **end** (*rhino3dm.Point3d*) – Optional ending point of loft. Use Point3d.Unset if you do not want to include an end point. "end and "EndTangent" cannot both be true.
- **StartTangent** (*bool*) – If StartTangent is True and the first loft curve is a surface edge, the loft will match the tangent of the surface behind that edge.
- **EndTangent** (*bool*) – If EndTangent is True and the first loft curve is a surface edge, the loft will match the tangent of the surface behind that edge.
- **StartTrim** (*BrepTrim*) – BrepTrim from the surface edge where start tangent is to be matched
- **EndTrim** (*BrepTrim*) – BrepTrim from the surface edge where end tangent is to be matched
- **loftType** (*LoftType*) – type of loft to perform.
- **closed** (*bool*) – True if the last curve in this loft should be connected back to the first one.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Constructs a closed surface, continuing the surface past the last curve around to the first curve. Available when you have selected three shape curves.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarUnion**(*breps*, *plane*, *tolerance*, *multiple=false*)

CreatePlanarUnion

Arguments

- **breps** (*list[rhino3dm.Brep]*) – The planar regions on which to preform the union operation.
- **plane** (*rhino3dm.Plane*) – The plane in which all the input breps lie
- **tolerance** (*float*) – Tolerance to use for union operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarUnion1**(*b0*, *b1*, *plane*, *tolerance*, *multiple=false*)

CreatePlanarUnion

Arguments

- **b0** (*rhino3dm.Brep*) – The first brep to union.
- **b1** (*rhino3dm.Brep*) – The first brep to union.
- **plane** (*rhino3dm.Plane*) – The plane in which all the input breps lie
- **tolerance** (*float*) – Tolerance to use for union operation.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarDifference**(*b0*, *b1*, *plane*, *tolerance*, *multiple=false*)
CreatePlanarDifference

Arguments

- **b0** (*rhino3dm.Brep*) – The first brep to intersect.
- **b1** (*rhino3dm.Brep*) – The first brep to intersect.
- **plane** (*rhino3dm.Plane*) – The plane in which all the input breps lie
- **tolerance** (*float*) – Tolerance to use for Difference operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createPlanarIntersection**(*b0*, *b1*, *plane*, *tolerance*, *multiple=false*)
CreatePlanarIntersection

Arguments

- **b0** (*rhino3dm.Brep*) – The first brep to intersect.
- **b1** (*rhino3dm.Brep*) – The first brep to intersect.
- **plane** (*rhino3dm.Plane*) – The plane in which all the input breps lie
- **tolerance** (*float*) – Tolerance to use for intersection operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanUnion**(*breps*, *tolerance*, *multiple=false*)
Compute the Boolean Union of a set of Breps.

Arguments

- **breps** (*list[rhino3dm.Brep]*) – Breps to union.
- **tolerance** (*float*) – Tolerance to use for union operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanUnion1**(*breps*, *tolerance*, *manifoldOnly*, *multiple=false*)
Compute the Boolean Union of a set of Breps.

Arguments

- **breps** (*list[rhino3dm.Brep]*) – Breps to union.

- **tolerance** (*float*) – Tolerance to use for union operation.
- **manifoldOnly** (*bool*) – If true, non-manifold input breps are ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanIntersection**(*firstSet*, *secondSet*, *tolerance*, *multiple=false*)

Compute the Solid Intersection of two sets of Breps.

Arguments

- **firstSet** (*list[rhino3dm.Brep]*) – First set of Breps.
- **secondSet** (*list[rhino3dm.Brep]*) – Second set of Breps.
- **tolerance** (*float*) – Tolerance to use for intersection operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanIntersection1**(*firstSet*, *secondSet*, *tolerance*, *manifoldOnly*, *multiple=false*)

Compute the Solid Intersection of two sets of Breps.

Arguments

- **firstSet** (*list[rhino3dm.Brep]*) – First set of Breps.
- **secondSet** (*list[rhino3dm.Brep]*) – Second set of Breps.
- **tolerance** (*float*) – Tolerance to use for intersection operation.
- **manifoldOnly** (*bool*) – If true, non-manifold input breps are ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanIntersection2**(*firstBrep*, *secondBrep*, *tolerance*, *multiple=false*)

Compute the Solid Intersection of two Breps.

Arguments

- **firstBrep** (*rhino3dm.Brep*) – First Brep for boolean intersection.
- **secondBrep** (*rhino3dm.Brep*) – Second Brep for boolean intersection.
- **tolerance** (*float*) – Tolerance to use for intersection operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanIntersection3**(*firstBrep*, *secondBrep*, *tolerance*, *manifoldOnly*, *multiple=false*)

Compute the Solid Intersection of two Breps.

Arguments

- **firstBrep** (*rhino3dm.Brep*) – First Brep for boolean intersection.
- **secondBrep** (*rhino3dm.Brep*) – Second Brep for boolean intersection.
- **tolerance** (*float*) – Tolerance to use for intersection operation.
- **manifoldOnly** (*bool*) – If true, non-manifold input breps are ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanDifference**(*firstSet*, *secondSet*, *tolerance*, *multiple=false*)

Compute the Solid Difference of two sets of Breps.

Arguments

- **firstSet** (*list[rhino3dm.Brep]*) – First set of Breps (the set to subtract from).
- **secondSet** (*list[rhino3dm.Brep]*) – Second set of Breps (the set to subtract).
- **tolerance** (*float*) – Tolerance to use for difference operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanDifference1**(*firstSet*, *secondSet*, *tolerance*, *manifoldOnly*, *multiple=false*)

Compute the Solid Difference of two sets of Breps.

Arguments

- **firstSet** (*list[rhino3dm.Brep]*) – First set of Breps (the set to subtract from).
- **secondSet** (*list[rhino3dm.Brep]*) – Second set of Breps (the set to subtract).
- **tolerance** (*float*) – Tolerance to use for difference operation.
- **manifoldOnly** (*bool*) – If true, non-manifold input breps are ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanDifference2**(*firstBrep*, *secondBrep*, *tolerance*, *multiple=false*)

Compute the Solid Difference of two Breps.

Arguments

- **firstBrep** (*rhino3dm.Brep*) – First Brep for boolean difference.
- **secondBrep** (*rhino3dm.Brep*) – Second Brep for boolean difference.

- **tolerance** (*float*) – Tolerance to use for difference operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanDifference3**(*firstBrep*, *secondBrep*, *tolerance*, *manifoldOnly*, *multiple=false*)

Compute the Solid Difference of two Breps.

Arguments

- **firstBrep** (*rhino3dm.Brep*) – First Brep for boolean difference.
- **secondBrep** (*rhino3dm.Brep*) – Second Brep for boolean difference.
- **tolerance** (*float*) – Tolerance to use for difference operation.
- **manifoldOnly** (*bool*) – If true, non-manifold input breps are ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanSplit**(*firstBrep*, *secondBrep*, *tolerance*, *multiple=false*)

Splits shared areas of Breps and creates separate Breps from the shared and unshared parts.

Arguments

- **firstBrep** (*rhino3dm.Brep*) – The Brep to split.
- **secondBrep** (*rhino3dm.Brep*) – The cutting Brep.
- **tolerance** (*float*) – Tolerance to use for splitting operation. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep if successful, an empty array on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createBooleanSplit1**(*firstSet*, *secondSet*, *tolerance*, *multiple=false*)

Splits shared areas of Breps and creates separate Breps from the shared and unshared parts.

Arguments

- **firstSet** (*list[rhino3dm.Brep]*) – The Breps to split.
- **secondSet** (*list[rhino3dm.Brep]*) – The cutting Breps.
- **tolerance** (*float*) – Tolerance to use for splitting operation. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep if successful, an empty array on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**createShell**(*brep*, *facesToRemove*, *distance*, *tolerance*, *multiple=false*)

Creates a hollowed out shell from a solid Brep. Function only operates on simple, solid, manifold Breps.

Arguments

- **brep** (*rhino3dm.Brep*) – The solid Brep to shell.
- **facesToRemove** (*list[int]*) – The indices of the Brep faces to remove. These surfaces are removed and the remainder is offset inward, using the outer parts of the removed surfaces to join the inner and outer parts.
- **distance** (*float*) – The distance, or thickness, for the shell. This is a signed distance value with respect to face normals and flipped faces.
- **tolerance** (*float*) – The offset tolerance. When in doubt, use the document's absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Brep results or None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**joinBreps**(*brepsToJoin*, *tolerance*, *multiple=false*)

Joins the breps in the input array at any overlapping edges to form as few as possible resulting breps. There may be more than one brep in the result array.

Arguments

- **brepsToJoin** (*list[rhino3dm.Brep]*) – A list, an array or any enumerable set of breps to join.
- **tolerance** (*float*) – 3d distance tolerance for detecting overlapping edges.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns new joined breps on success, None on failure.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**mergeBreps**(*brepsToMerge*, *tolerance*, *multiple=false*)

Combines two or more breps into one. A merge is like a boolean union that keeps the inside pieces. This function creates non-manifold Breps which in general are unusual in Rhino. You may want to consider using JoinBreps or CreateBooleanUnion functions instead.

Arguments

- **brepsToMerge** (*list[rhino3dm.Brep]*) – must contain more than one Brep.
- **tolerance** (*float*) – the tolerance to use when merging.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Single merged Brep on success. Null on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**createContourCurves**(*brepToContour*, *contourStart*, *contourEnd*, *interval*, *multiple=false*)

Constructs the contour curves for a brep at a specified interval.

Arguments

- **brepToContour** (*rhino3dm.Brep*) – A brep or polysurface.

- **contourStart** (*rhino3dm.Point3d*) – A point to start.
- **contourEnd** (*rhino3dm.Point3d*) – A point to use as the end.
- **interval** (*float*) – The interaxial offset in world units.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array with intersected curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Brep.**createContourCurves1**(*brepToContour*, *sectionPlane*, *multiple=false*)
Constructs the contour curves for a brep, using a slicing plane.

Arguments

- **brepToContour** (*rhino3dm.Brep*) – A brep or polysurface.
- **sectionPlane** (*rhino3dm.Plane*) – A plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array with intersected curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Brep.**createCurvatureAnalysisMesh**(*brep*, *state*, *multiple=false*)
Create an array of analysis meshes for the brep using the specified settings. Meshes aren't set on the brep.

Arguments

- **state** (*Rhino.ApplicationSettings.CurvatureAnalysisSettingsState*) – CurvatureAnalysisSettingsState
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if meshes were created

Return type rhino3dm.Mesh[]

RhinoCompute.Brep.**destroyRegionTopology**(*thisBrep*, *multiple=false*)
Destroys a Brep's region topology information.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Brep.**getRegions**(*thisBrep*, *multiple=false*)
Gets an array containing all regions in this brep.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of regions in this brep. This array can be empty, but not null.

Return type BrepRegion[]

RhinoCompute.Brep.**getWireframe**(*thisBrep*, *density*, *multiple=false*)
Constructs all the Wireframe curves for this Brep.

Arguments

- **density** (*int*) – Wireframe density. Valid values range between -1 and 99.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Wireframe curves or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Brep.**closestPoint**(*thisBrep*, *testPoint*, *multiple=false*)

Finds a point on the brep that is closest to testPoint.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Base point to project to brep.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The point on the Brep closest to testPoint or Point3d.Unset if the operation failed.

Return type rhino3dm.Point3d

RhinoCompute.Brep.**isPointInside**(*thisBrep*, *point*, *tolerance*, *strictlyIn*, *multiple=false*)

Determines if point is inside a Brep. This question only makes sense when the brep is a closed and manifold. This function does not check for closed or manifold, so result is not valid in those cases. Intersects a line through point with brep, finds the intersection point Q closest to point, and looks at face normal at Q. If the point Q is on an edge or the intersection is not transverse at Q, then another line is used.

Arguments

- **point** (*rhino3dm.Point3d*) – 3d point to test.
- **tolerance** (*float*) – 3d distance tolerance used for intersection and determining strict inclusion. A good default is RhinoMath.SqrtEpsilon.
- **strictlyIn** (*bool*) – if true, point is in if inside brep by at least tolerance. if false, point is in if truly in or within tolerance of boundary.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if point is in, False if not.

Return type bool

RhinoCompute.Brep.**getPointInside**(*thisBrep*, *tolerance*, *multiple=false*)

Finds a point inside of a solid Brep.

Arguments

- **tolerance** (*float*) – Used for intersecting rays and is not necessarily related to the distance from the brep to the found point. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns False if the input is not solid and manifold, if the Brep's bounding box is less than 2.0 * tolerance wide, or if no point could be found due to ray shooting or other errors. Otherwise, True is returned.

Return type bool

RhinoCompute.Brep.**capPlanarHoles**(*thisBrep*, *tolerance*, *multiple=false*)
Returns a new Brep that is equivalent to this Brep with all planar holes capped.

Arguments

- **tolerance** (*float*) – Tolerance to use for capping.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New brep on success. None on error.

Return type rhino3dm.Brep

RhinoCompute.Brep.**join**(*thisBrep*, *otherBrep*, *tolerance*, *compact*, *multiple=false*)
If any edges of this brep overlap edges of otherBrep, merge a copy of otherBrep into this brep joining all edges that overlap within tolerance.

Arguments

- **otherBrep** (*rhino3dm.Brep*) – Brep to be added to this brep.
- **tolerance** (*float*) – 3d distance tolerance for detecting overlapping edges.
- **compact** (*bool*) – if true, set brep flags and tolerances, remove unused faces and edges.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if any edges were joined.

Return type bool

RhinoCompute.Brep.**joinNakedEdges**(*thisBrep*, *tolerance*, *multiple=false*)
Joins naked edge pairs within the same brep that overlap within tolerance.

Arguments

- **tolerance** (*float*) – The tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns number of joins made.

Return type int

RhinoCompute.Brep.**mergeCoplanarFaces**(*thisBrep*, *tolerance*, *multiple=false*)
Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.Brep.**mergeCoplanarFaces1**(*thisBrep*, *tolerance*, *angleTolerance*, *multiple=false*)
Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **angleTolerance** (*float*) – Angle tolerance, in radians, for determining when faces are parallel. When in doubt, use the document's ModelAngleToleranceRadians property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.Brep.**split**(*thisBrep*, *cutter*, *intersectionTolerance*, *multiple=false*)
Splits a Brep into pieces using a Brep as a cutter.

Arguments

- **cutter** (*rhino3dm.Brep*) – The Brep to use as a cutter.
- **intersectionTolerance** (*float*) – The tolerance with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Breps. This array can be empty.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**split1**(*thisBrep*, *cutter*, *intersectionTolerance*, *multiple=false*)
Splits a Brep into pieces using a Brep as a cutter.

Arguments

- **cutter** (*rhino3dm.Brep*) – The Brep to use as a cutter.
- **intersectionTolerance** (*float*) – The tolerance with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Breps. This array can be empty.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**split2**(*thisBrep*, *cutters*, *intersectionTolerance*, *multiple=false*)
Splits a Brep into pieces using Breps as cutters.

Arguments

- **cutters** (*list[rhino3dm.Brep]*) – One or more Breps to use as cutters.
- **intersectionTolerance** (*float*) – The tolerance with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Breps. This array can be empty.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**split3**(*thisBrep*, *cutters*, *intersectionTolerance*, *multiple=false*)
Splits a Brep into pieces using curves, at least partially on the Brep, as cutters.

Arguments

- **cutters** (*list[rhino3dm.Curve]*) – The splitting curves. Only the portion of the curve on the Brep surface will be used for cutting.
- **intersectionTolerance** (*float*) – The tolerance with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Breps. This array can be empty.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**split4**(*thisBrep*, *cutters*, *normal*, *planView*, *intersectionTolerance*, *multiple=false*)

Splits a Brep into pieces using a combination of curves, to be extruded, and Breps as cutters.

Arguments

- **cutters** (*list[rhino3dm.GeometryBase]*) – The curves, surfaces, faces and Breps to be used as cutters. Any other geometry is ignored.
- **normal** (*rhino3dm.Vector3d*) – A construction plane normal, used in deciding how to extrude a curve into a cutter.
- **planView** (*bool*) – Set True if the assume view is a plan, or parallel projection, view.
- **intersectionTolerance** (*float*) – The tolerance with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of Breps. This array can be empty.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**trim**(*thisBrep*, *cutter*, *intersectionTolerance*, *multiple=false*)

Trims a brep with an oriented cutter. The parts of the brep that lie inside (opposite the normal) of the cutter are retained while the parts to the outside (in the direction of the normal) are discarded. If the Cutter is closed, then a connected component of the Brep that does not intersect the cutter is kept if and only if it is contained in the inside of cutter. That is the region bounded by cutter opposite from the normal of cutter, If cutter is not closed all these components are kept.

Arguments

- **cutter** (*rhino3dm.Brep*) – A cutting brep.
- **intersectionTolerance** (*float*) – A tolerance value with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns This Brep is not modified, the trim results are returned in an array.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**trim1**(*thisBrep*, *cutter*, *intersectionTolerance*, *multiple=false*)

Trims a Brep with an oriented cutter. The parts of Brep that lie inside (opposite the normal) of the cutter are retained while the parts to the outside (in the direction of the normal) are discarded. A connected component of Brep that does not intersect the cutter is kept if and only if it is contained in the inside of Cutter. That is

the region bounded by cutter opposite from the normal of cutter, or in the case of a Plane cutter the half space opposite from the plane normal.

Arguments

- **cutter** (*rhino3dm.Plane*) – A cutting plane.
- **intersectionTolerance** (*float*) – A tolerance value with which to compute intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns This Brep is not modified, the trim results are returned in an array.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**unjoinEdges**(*thisBrep*, *edgesToUnjoin*, *multiple=false*)

Un-joins, or separates, edges within the Brep. Note, seams in closed surfaces will not separate.

Arguments

- **edgesToUnjoin** (*list[int]*) – The indices of the Brep edges to un-join.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns This Brep is not modified, the trim results are returned in an array.

Return type rhino3dm.Brep[]

RhinoCompute.Brep.**joinEdges**(*thisBrep*, *edgeIndex0*, *edgeIndex1*, *joinTolerance*, *compact*, *multiple=false*)

Joins two naked edges, or edges that are coincident or close together.

Arguments

- **edgeIndex0** (*int*) – The first edge index.
- **edgeIndex1** (*int*) – The second edge index.
- **joinTolerance** (*float*) – The join tolerance.
- **compact** (*bool*) – If joining more than one edge pair and want the edge indices of subsequent pairs to remain valid, set to false. But then call Brep.Compact() on the final result.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Brep.**transformComponent**(*thisBrep*, *components*, *xform*, *tolerance*, *timeLimit*, *useMultipleThreads*, *multiple=false*)

Transform an array of Brep components, bend neighbors to match, and leave the rest fixed.

Arguments

- **components** (*IEnumerable<ComponentIndex>*) – The Brep components to transform.
- **xform** (*Transform*) – The transformation to apply.
- **tolerance** (*float*) – The desired fitting tolerance to use when bending faces that share edges with both fixed and transformed components.

- **timeLimit** (*float*) – If the deformation is extreme, it can take a long time to calculate the result. If time_limit > 0, then the value specifies the maximum amount of time in seconds you want to spend before giving up.
- **useMultipleThreads** (*bool*) – True if multiple threads can be used.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Brep.**getArea**(*thisBrep*, *multiple=false*)

Compute the Area of the Brep. If you want proper Area data with moments and error information, use the AreaMassProperties class.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The area of the Brep.

Return type float

RhinoCompute.Brep.**getArea1**(*thisBrep*, *relativeTolerance*, *absoluteTolerance*, *multiple=false*)

Compute the Area of the Brep. If you want proper Area data with moments and error information, use the AreaMassProperties class.

Arguments

- **relativeTolerance** (*float*) – Relative tolerance to use for area calculation.
- **absoluteTolerance** (*float*) – Absolute tolerance to use for area calculation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The area of the Brep.

Return type float

RhinoCompute.Brep.**getVolume**(*thisBrep*, *multiple=false*)

Compute the Volume of the Brep. If you want proper Volume data with moments and error information, use the VolumeMassProperties class.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The volume of the Brep.

Return type float

RhinoCompute.Brep.**getVolume1**(*thisBrep*, *relativeTolerance*, *absoluteTolerance*, *multiple=false*)

Compute the Volume of the Brep. If you want proper Volume data with moments and error information, use the VolumeMassProperties class.

Arguments

- **relativeTolerance** (*float*) – Relative tolerance to use for area calculation.
- **absoluteTolerance** (*float*) – Absolute tolerance to use for area calculation.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The volume of the Brep.

Return type float

RhinoCompute.Brep.**rebuildTrimsForV2**(*thisBrep*, *face*, *nurbsSurface*, *multiple=false*)

No support is available for this function. Expert user function used by MakeValidForV2 to convert trim curves from one surface to its NURBS form. After calling this function, you need to change the surface of the face to a NurbsSurface.

Arguments

- **face** (*rhino3dm.BrepFace*) – Face whose underlying surface has a parameterization that is different from its NURBS form.
- **nurbsSurface** (*NurbsSurface*) – NURBS form of the face's underlying surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Brep.**makeValidForV2**(*thisBrep*, *multiple=false*)

No support is available for this function. Expert user function that converts all geometry in Brep to NURB form.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type bool

RhinoCompute.Brep.**repair**(*thisBrep*, *tolerance*, *multiple=false*)

Fills in missing or fixes incorrect component information from a Brep. Useful when reading Brep information from other file formats that do not provide as complete of a Brep definition as required by Rhino.

Arguments

- **tolerance** (*float*) – The repair tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success.

Return type bool

RhinoCompute.Brep.**removeHoles**(*thisBrep*, *tolerance*, *multiple=false*)

Remove all inner loops, or holes, in a Brep.

Arguments

- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The Brep without holes if successful, None otherwise.

Return type rhino3dm.Brep

RhinoCompute.Brep.**removeHoles1**(*thisBrep*, *loops*, *tolerance*, *multiple=false*)

Removes inner loops, or holes, in a Brep.

Arguments

- **loops** (*IEnumerable<ComponentIndex>*) – A list of BrepLoop component indexes, where BrepLoop.LoopType == Rhino.Geometry.BrepLoopType.Inner.
- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The Brep without holes if successful, None otherwise.

Return type rhino3dm.Brep

CHAPTER 4

RhinoCompute.BrepFace

RhinoCompute.BrepFace.**pullPointsToFace**(*thisBrepFace*, *points*, *tolerance*, *multiple=false*)

Pulls one or more points to a brep face.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – Points to pull.
- **tolerance** (*float*) – Tolerance for pulling operation. Only points that are closer than tolerance will be pulled to the face.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of pulled points.

Return type rhino3dm.Point3d[]

RhinoCompute.BrepFace.**draftAnglePoint**(*thisBrepFace*, *testPoint*, *testAngle*, *pullDirection*, *edge*, *multiple=false*)

Returns the surface draft angle and point at a parameter.

Arguments

- **testPoint** (*rhino3dm.Point2d*) – The u,v parameter on the face to evaluate.
- **testAngle** (*float*) – The angle in radians to test.
- **pullDirection** (*rhino3dm.Vector3d*) – The pull direction.
- **edge** (*bool*) – Restricts the point placement to an edge.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.BrepFace.**removeHoles**(*thisBrepFace*, *tolerance*, *multiple=false*)

Remove all inner loops, or holes, from a Brep face.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Brep

RhinoCompute.BrepFace.**shrinkSurfaceToEdge**(*thisBrepFace*, *multiple=false*)
Shrinks the underlying untrimmed surface of this Brep face right to the trimming boundaries. Note, shrinking the trimmed surface can sometimes cause problems later since having the edges so close to the trimming boundaries can cause commands that use the surface edges as input to fail.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.BrepFace.**split**(*thisBrepFace*, *curves*, *tolerance*, *multiple=false*)
Split this face using 3D trimming curves.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – Curves to split with.
- **tolerance** (*float*) – Tolerance for splitting, when in doubt use the Document Absolute Tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A brep consisting of all the split fragments, or None on failure.

Return type rhino3dm.Brep

RhinoCompute.BrepFace.**isPointOnFace**(*thisBrepFace*, *u*, *v*, *multiple=false*)
Tests if a parameter space point is in the active region of a face.

Arguments

- **u** (*float*) – Parameter space point U value.
- **v** (*float*) – Parameter space point V value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A value describing the relationship between the point and the face.

Return type PointFaceRelation

RhinoCompute.BrepFace.**isPointOnFace1**(*thisBrepFace*, *u*, *v*, *tolerance*, *multiple=false*)
Tests if a parameter space point is in the active region of a face.

Arguments

- **u** (*float*) – Parameter space point U value.
- **v** (*float*) – Parameter space point V value.
- **tolerance** (*float*) – 3D tolerance used when checking to see if the point is on a face or inside of a loop.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A value describing the relationship between the point and the face.

Return type PointFaceRelation

RhinoCompute.BrepFace.**trimAwareIsoIntervals**(*thisBrepFace*, *direction*, *constantParameter*, *multiple=false*)

Gets intervals where the iso curve exists on a BrepFace (trimmed surface)

Arguments

- **direction** (*int*) – Direction of isocurve. 0 = Isocurve connects all points with a constant U value.1 = Isocurve connects all points with a constant V value.
- **constantParameter** (*float*) – Surface parameter that remains identical along the isocurves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If direction = 0, the parameter space iso interval connects the 2d points (intervals[i][0],iso_constant) and (intervals[i][1],iso_constant). If direction = 1, the parameter space iso interval connects the 2d points (iso_constant,intervals[i][0]) and (iso_constant,intervals[i][1]).

Return type rhino3dm.Interval[]

RhinoCompute.BrepFace.**trimAwareIsoCurve**(*thisBrepFace*, *direction*, *constantParameter*, *multiple=false*)

Similar to IsoCurve function, except this function pays attention to trims on faces and may return multiple curves.

Arguments

- **direction** (*int*) – Direction of isocurve. 0 = Isocurve connects all points with a constant U value.1 = Isocurve connects all points with a constant V value.
- **constantParameter** (*float*) – Surface parameter that remains identical along the isocurves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Isoparametric curves connecting all points with the constantParameter value.

Return type rhino3dm.Curve[]

RhinoCompute.BrepFace.**changeSurface**(*thisBrepFace*, *surfaceIndex*, *multiple=false*)

Expert user tool that replaces the 3d surface geometry use by the face.

Arguments

- **surfaceIndex** (*int*) – brep surface index of new surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful.

Return type bool

RhinoCompute.BrepFace.**rebuildEdges**(*thisBrepFace*, *tolerance*, *rebuildSharedEdges*, *rebuildVertices*, *multiple=false*)

Rebuild the edges used by a face so they lie on the surface.

Arguments

- **tolerance** (*float*) – tolerance for fitting 3d edge curves.
- **rebuildSharedEdges** (*bool*) – if False and edge is used by this face and a neighbor, then the edge will be skipped.
- **rebuildVertices** (*bool*) – if true, vertex locations are updated to lie on the surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success.

Return type bool

CHAPTER 5

RhinoCompute.Curve

RhinoCompute.Curve.**getConicSectionType**(*thisCurve*, *multiple=false*)

Returns the type of conic section based on the curve's shape.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type ConicSectionType

RhinoCompute.Curve.**createInterpolatedCurve**(*points*, *degree*, *multiple=false*)

Interpolates a sequence of points. Used by InterpCurve Command This routine works best when degree=3.

Arguments

- **degree** (*int*) – The degree of the curve >=1. Degree must be odd.
- **points** (*list[rhino3dm.Point3d]*) – Points to interpolate (Count must be >= 2)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns interpolated curve on success. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createInterpolatedCurve1**(*points*, *degree*, *knots*, *multiple=false*)

Interpolates a sequence of points. Used by InterpCurve Command This routine works best when degree=3.

Arguments

- **degree** (*int*) – The degree of the curve >=1. Degree must be odd.
- **points** (*list[rhino3dm.Point3d]*) – Points to interpolate. For periodic curves if the final point is a duplicate of the initial point it is ignored. (Count must be >=2)
- **knots** (*CurveKnotStyle*) – Knot-style to use and specifies if the curve should be periodic.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns interpolated curve on success. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createInterpolatedCurve2**(*points*, *degree*, *knots*, *startTangent*, *endTangent*, *multiple=false*)

Interpolates a sequence of points. Used by InterpCurve Command This routine works best when degree=3.

Arguments

- **degree** (*int*) – The degree of the curve >=1. Degree must be odd.
- **points** (*list[rhino3dm.Point3d]*) – Points to interpolate. For periodic curves if the final point is a duplicate of the initial point it is ignored. (Count must be >=2)
- **knots** (*CurveKnotStyle*) – Knot-style to use and specifies if the curve should be periodic.
- **startTangent** (*rhino3dm.Vector3d*) – A starting tangent.
- **endTangent** (*rhino3dm.Vector3d*) – An ending tangent.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns interpolated curve on success. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createSoftEditCurve**(*curve*, *t*, *delta*, *length*, *fixEnds*, *multiple=false*)

Creates a soft edited curve from an existing curve using a smooth field of influence.

Arguments

- **curve** (*rhino3dm.Curve*) – The curve to soft edit.
- **t** (*float*) – A parameter on the curve to move from. This location on the curve is moved, and the move is smoothly tapered off with increasing distance along the curve from this parameter.
- **delta** (*rhino3dm.Vector3d*) – The direction and magnitude, or maximum distance, of the move.
- **length** (*float*) – The distance along the curve from the editing point over which the strength of the editing falls off smoothly.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The soft edited curve if successful. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createFilletCornersCurve**(*curve*, *radius*, *tolerance*, *angleTolerance*, *multiple=false*)

Rounds the corners of a kinked curve with arcs of a single, specified radius.

Arguments

- **curve** (*rhino3dm.Curve*) – The curve to fillet.
- **radius** (*float*) – The fillet radius.

- **tolerance** (*float*) – The tolerance. When in doubt, use the document's model space absolute tolerance.
- **angleTolerance** (*float*) – The angle tolerance in radians. When in doubt, use the document's model space angle tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The filleted curve if successful. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**joinCurves**(*inputCurves*, *joinTolerance*, *preserveDirection*, *multiple=false*)

Joins a collection of curve segments together.

Arguments

- **inputCurves** (*list[rhino3dm.Curve]*) – An array, a list or any enumerable set of curve segments to join.
- **joinTolerance** (*float*) – Joining tolerance, i.e. the distance between segment endpoints that is allowed.
- **preserveDirection** (*bool*) – If true, curve endpoints will be compared to curve start points.If false, all start and endpoints will be compared and copies of input curves may be reversed in output.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createArcLineArcBlend**(*startPt*, *startDir*, *endPt*, *endDir*, *radius*, *multiple=false*)

Creates an arc-line-arc blend curve between two curves. The output is generally a PolyCurve with three segments: arc, line, arc. In some cases, one or more of those segments will be absent because they would have 0 length. If there is only a single segment, the result will either be an ArcCurve or a LineCurve.

Arguments

- **startPt** (*rhino3dm.Point3d*) – Start of the blend curve.
- **startDir** (*rhino3dm.Vector3d*) – Start direction of the blend curve.
- **endPt** (*rhino3dm.Point3d*) – End of the blend curve.
- **endDir** (*rhino3dm.Vector3d*) – End direction of the arc blend curve.
- **radius** (*float*) – The radius of the arc segments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The blend curve if successful, False otherwise.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createArcBlend**(*startPt*, *startDir*, *endPt*, *endDir*, *controlPointLengthRatio*, *multiple=false*)

Creates a polycurve consisting of two tangent arc segments that connect two points and two directions.

Arguments

- **startPt** (*rhino3dm.Point3d*) – Start of the arc blend curve.

- **startDir** (*rhino3dm.Vector3d*) – Start direction of the arc blend curve.
- **endPt** (*rhino3dm.Point3d*) – End of the arc blend curve.
- **endDir** (*rhino3dm.Vector3d*) – End direction of the arc blend curve.
- **controlPointLengthRatio** (*float*) – The ratio of the control polygon lengths of the two arcs. Note, a value of 1.0 means the control polygon lengths for both arcs will be the same.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The arc blend curve, or None on error.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createMeanCurve**(*curveA*, *curveB*, *angleToleranceRadians*, *multiple=false*)

Constructs a mean, or average, curve from two curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – A first curve.
- **curveB** (*rhino3dm.Curve*) – A second curve.
- **angleToleranceRadians** (*float*) – The angle tolerance, in radians, used to match kinks between curves. If you are unsure how to set this parameter, then either use the document's angle tolerance RhinoDoc.AngleToleranceRadians, or the default value (RhinoMath.UnsetValue)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The average curve, or None on error.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createMeanCurve1**(*curveA*, *curveB*, *multiple=false*)

Constructs a mean, or average, curve from two curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – A first curve.
- **curveB** (*rhino3dm.Curve*) – A second curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The average curve, or None on error.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createBlendCurve**(*curveA*, *curveB*, *continuity*, *multiple=false*)

Create a Blend curve between two existing curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – Curve to blend from (blending will occur at curve end point).
- **curveB** (*rhino3dm.Curve*) – Curve to blend to (blending will occur at curve start point).
- **continuity** (*BlendContinuity*) – Continuity of blend.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve representing the blend between A and B or None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createBlendCurve1**(*curveA*, *curveB*, *continuity*, *bulgeA*, *bulgeB*, *multiple=false*)

Create a Blend curve between two existing curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – Curve to blend from (blending will occur at curve end point).
- **curveB** (*rhino3dm.Curve*) – Curve to blend to (blending will occur at curve start point).
- **continuity** (*BlendContinuity*) – Continuity of blend.
- **bulgeA** (*float*) – Bulge factor at curveA end of blend. Values near 1.0 work best.
- **bulgeB** (*float*) – Bulge factor at curveB end of blend. Values near 1.0 work best.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve representing the blend between A and B or None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createBlendCurve2**(*curve0*, *t0*, *reverse0*, *continuity0*, *curve1*, *t1*, *reverse1*, *continuity1*, *multiple=false*)

Makes a curve blend between 2 curves at the parameters specified with the directions and continuities specified

Arguments

- **curve0** (*rhino3dm.Curve*) – First curve to blend from
- **t0** (*float*) – Parameter on first curve for blend endpoint
- **reverse0** (*bool*) – If false, the blend will go in the natural direction of the curve. If true, the blend will go in the opposite direction to the curve
- **continuity0** (*BlendContinuity*) – Continuity for the blend at the start
- **curve1** (*rhino3dm.Curve*) – Second curve to blend from
- **t1** (*float*) – Parameter on second curve for blend endpoint
- **reverse1** (*bool*) – If false, the blend will go in the natural direction of the curve. If true, the blend will go in the opposite direction to the curve
- **continuity1** (*BlendContinuity*) – Continuity for the blend at the end
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The blend curve on success. None on failure

Return type rhino3dm.Curve

RhinoCompute.Curve.**createMatchCurve**(*curve0*, *reverse0*, *continuity*, *curve1*, *reverse1*, *preserve*, *average*, *multiple=false*)

Changes a curve end to meet a specified curve with a specified continuity.

Arguments

- **curve0** (*rhino3dm.Curve*) – The open curve to change.
- **reverse0** (*bool*) – Reverse the directon of the curve to change before matching.
- **continuity** (*BlendContinuity*) – The continuity at the curve end.
- **curve1** (*rhino3dm.Curve*) – The open curve to match.
- **reverse1** (*bool*) – Reverse the directon of the curve to match before matching.
- **preserve** (*PreserveEnd*) – Prevent modification of the curvature at the end opposite the match for curves with fewer than six control points.
- **average** (*bool*) – Adjust both curves to match each other.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The results of the curve matching, if successful, otherwise an empty array.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurves**(*curve0*, *curve1*, *numCurves*, *multiple=false*)

Creates curves between two open or closed input curves. Uses the control points of the curves for finding tween curves. That means the first control point of first curve is matched to first control point of the second curve and so on. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurves1**(*curve0*, *curve1*, *numCurves*, *tolerance*, *multiple=false*)

Creates curves between two open or closed input curves. Uses the control points of the curves for finding tween curves. That means the first control point of first curve is matched to first control point of the second curve and so on. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurvesWithMatching**(*curve0*, *curve1*, *numCurves*, *multiple=false*)

Creates curves between two open or closed input curves. Make the structure of input curves compatible if needed. Refits the input curves to have the same structure. The resulting curves are usually more complex than input unless input curves are compatible and no refit is needed. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurvesWithMatching1**(*curve0*, *curve1*, *numCurves*, *tolerance*, *multiple=false*)

Creates curves between two open or closed input curves. Make the structure of input curves compatible if needed. Refits the input curves to have the same structure. The resulting curves are usually more complex than input unless input curves are compatible and no refit is needed. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurvesWithSampling**(*curve0*, *curve1*, *numCurves*, *numSamples*, *multiple=false*)

Creates curves between two open or closed input curves. Use sample points method to make curves compatible. This is how the algorithm works: Divides the two curves into an equal number of points, finds the midpoint between the corresponding points on the curves and interpolates the tween curve through those points. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **numSamples** (*int*) – Number of sample points along input curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns >An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createTweenCurvesWithSampling1**(*curve0*, *curve1*, *numCurves*, *numSamples*, *tolerance*, *multiple=false*)

Creates curves between two open or closed input curves. Use sample points method to make curves compatible. This is how the algorithm works: Divides the two curves into an equal number of points, finds the midpoint between the corresponding points on the curves and interpolates the tween curve through those points. There is no matching of curves direction. Caller must match input curves direction before calling the function.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first, or starting, curve.
- **curve1** (*rhino3dm.Curve*) – The second, or ending, curve.
- **numCurves** (*int*) – Number of tween curves to create.
- **numSamples** (*int*) – Number of sample points along input curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns >An array of joint curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**makeEndsMeet**(*curveA*, *adjustStartCurveA*, *curveB*, *adjustStartCurveB*, *multiple=false*)

Makes adjustments to the ends of one or both input curves so that they meet at a point.

Arguments

- **curveA** (*rhino3dm.Curve*) – 1st curve to adjust.
- **adjustStartCurveA** (*bool*) – Which end of the 1st curve to adjust: True is start, False is end.
- **curveB** (*rhino3dm.Curve*) – 2nd curve to adjust.
- **adjustStartCurveB** (*bool*) – which end of the 2nd curve to adjust true==start, false==end.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success.

Return type bool

RhinoCompute.Curve.**createFillet**(*curve0*, *curve1*, *radius*, *t0Base*, *t1Base*, *multiple=false*)

Computes the fillet arc for a curve filleting operation.

Arguments

- **curve0** (*rhino3dm.Curve*) – First curve to fillet.
- **curve1** (*rhino3dm.Curve*) – Second curve to fillet.
- **radius** (*float*) – Fillet radius.
- **t0Base** (*float*) – Parameter on curve0 where the fillet ought to start (approximately).
- **t1Base** (*float*) – Parameter on curve1 where the fillet ought to end (approximately).
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The fillet arc on success, or Arc.Unset on failure.

Return type Arc

RhinoCompute.Curve.**createFilletCurves**(*curve0*, *point0*, *curve1*, *point1*, *radius*, *join*, *trim*, *arcExtension*, *tolerance*, *angleTolerance*, *multiple=false*)

Creates a tangent arc between two curves and trims or extends the curves to the arc.

Arguments

- **curve0** (*rhino3dm.Curve*) – The first curve to fillet.
- **point0** (*rhino3dm.Point3d*) – A point on the first curve that is near the end where the fillet will be created.
- **curve1** (*rhino3dm.Curve*) – The second curve to fillet.
- **point1** (*rhino3dm.Point3d*) – A point on the second curve that is near the end where the fillet will be created.
- **radius** (*float*) – The radius of the fillet.
- **join** (*bool*) – Join the output curves.
- **trim** (*bool*) – Trim copies of the input curves to the output fillet curve.
- **arcExtension** (*bool*) – Applies when arcs are filleted but need to be extended to meet the fillet curve or chamfer line. If true, then the arc is extended maintaining its validity. If false, then the arc is extended with a line segment, which is joined to the arc converting it to a polycurve.
- **tolerance** (*float*) – The tolerance, generally the document's absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The results of the fillet operation. The number of output curves depends on the input curves and the values of the parameters that were used during the fillet operation. In most cases, the output array will contain either one or three curves, although two curves can be returned if the radius is zero and join = false. For example, if both join and trim = true, then the output curve will be a polycurve containing the fillet curve joined with trimmed copies of the input curves. If join = False and trim = true, then three curves, the fillet curve and trimmed copies of the input curves, will be returned. If both join and trim = false, then just the fillet curve is returned.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanUnion**(*curves*, *multiple=false*)

Calculates the boolean union of two or more closed, planar curves. Note, curves must be co-planar.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The co-planar curves to union.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no union could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanUnion1**(*curves*, *tolerance*, *multiple=false*)

Calculates the boolean union of two or more closed, planar curves. Note, curves must be co-planar.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The co-planar curves to union.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no union could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanIntersection**(*curveA*, *curveB*, *multiple=false*)
Calculates the boolean intersection of two closed, planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **curveB** (*rhino3dm.Curve*) – The second closed, planar curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no intersection could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanIntersection1**(*curveA*, *curveB*, *tolerance*, *multiple=false*)
Calculates the boolean intersection of two closed, planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **curveB** (*rhino3dm.Curve*) – The second closed, planar curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no intersection could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanDifference**(*curveA*, *curveB*, *multiple=false*)
Calculates the boolean difference between two closed, planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **curveB** (*rhino3dm.Curve*) – The second closed, planar curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no difference could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanDifference1**(*curveA*, *curveB*, *tolerance*, *multiple=false*)
Calculates the boolean difference between two closed, planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **curveB** (*rhino3dm.Curve*) – The second closed, planar curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no difference could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanDifference2**(*curveA*, *subtractors*, *multiple=false*)

Calculates the boolean difference between a closed planar curve, and a list of closed planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **subtractors** (*list[rhino3dm.Curve]*) – curves to subtract from the first closed curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no difference could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanDifference3**(*curveA*, *subtractors*, *tolerance*, *multiple=false*)

Calculates the boolean difference between a closed planar curve, and a list of closed planar curves. Note, curves must be co-planar.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first closed, planar curve.
- **subtractors** (*list[rhino3dm.Curve]*) – curves to subtract from the first closed curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Result curves on success, empty array if no difference could be calculated.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createBooleanRegions**(*curves*, *plane*, *points*, *combineRegions*, *tolerance*, *multiple=false*)

Curve Boolean method, which trims and splits curves based on their overlapping regions.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The input curves.
- **plane** (*rhino3dm.Plane*) – Regions will be found in the projection of the curves to this plane.
- **points** (*list[rhino3dm.Point3d]*) – These points will be projected to plane. All regions that contain at least one of these points will be found.
- **combineRegions** (*bool*) – If true, then adjacent regions will be combined.
- **tolerance** (*float*) – Function tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The curve Boolean regions if successful, None of no successful.

Return type CurveBooleanRegions

RhinoCompute.Curve.**createBooleanRegions1**(*curves*, *plane*, *combineRegions*, *tolerance*, *multiple=false*)

Calculates curve Boolean regions, which trims and splits curves based on their overlapping regions.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The input curves.
- **plane** (*rhino3dm.Plane*) – Regions will be found in the projection of the curves to this plane.
- **combineRegions** (*bool*) – If true, then adjacent regions will be combined.
- **tolerance** (*float*) – Function tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The curve Boolean regions if successful, None of no successful.

Return type CurveBooleanRegions

RhinoCompute.Curve.**createTextOutlines**(*text*, *font*, *textHeight*, *textStyle*, *closeLoops*, *plane*, *smallCapsScale*, *tolerance*, *multiple=false*)

Creates outline curves created from a text string. The functionality is similar to what you find in Rhino's TextObject command or TextEntity.Explode() in RhinoCommon.

Arguments

- **text** (*str*) – The text from which to create outline curves.
- **font** (*str*) – The text font.
- **textHeight** (*float*) – The text height.
- **textStyle** (*int*) – The font style. The font style can be any number of the following: 0 - Normal, 1 - Bold, 2 - Italic
- **closeLoops** (*bool*) – Set this value to True when dealing with normal fonts and when you expect closed loops. You may want to set this to False when specifying a single-stroke font where you don't want closed loops.
- **plane** (*rhino3dm.Plane*) – The plane on which the outline curves will lie.
- **smallCapsScale** (*float*) – Displays lower-case letters as small caps. Set the relative text size to a percentage of the normal text.
- **tolerance** (*float*) – The tolerance for the operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array containing one or more curves if successful.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**createCurve2View**(*curveA*, *curveB*, *vectorA*, *vectorB*, *tolerance*, *angleTolerance*, *multiple=false*)

Creates a third curve from two curves that are planar in different construction planes. The new curve looks the same as each of the original curves when viewed in each plane.

Arguments

- **curveA** (*rhino3dm.Curve*) – The first curve.
- **curveB** (*rhino3dm.Curve*) – The second curve.

- **vectorA** (*rhino3dm.Vector3d*) – A vector defining the normal direction of the plane which the first curve is drawn upon.
- **vectorB** (*rhino3dm.Vector3d*) – A vector defining the normal direction of the plane which the second curve is drawn upon.
- **tolerance** (*float*) – The tolerance for the operation.
- **angleTolerance** (*float*) – The angle tolerance for the operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array containing one or more curves if successful.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**doDirectionsMatch**(*curveA*, *curveB*, *multiple=false*)

Determines whether two curves travel more or less in the same direction.

Arguments

- **curveA** (*rhino3dm.Curve*) – First curve to test.
- **curveB** (*rhino3dm.Curve*) – Second curve to test.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if both curves more or less point in the same direction, False if they point in the opposite directions.

Return type bool

RhinoCompute.Curve.**projectToMesh**(*curve*, *mesh*, *direction*, *tolerance*, *multiple=false*)

Projects a curve to a mesh using a direction and tolerance.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve.
- **mesh** (*rhino3dm.Mesh*) – A mesh.
- **direction** (*rhino3dm.Vector3d*) – A direction vector.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve array.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToMesh1**(*curve*, *meshes*, *direction*, *tolerance*, *multiple=false*)

Projects a curve to a set of meshes using a direction and tolerance.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve.
- **meshes** (*list[rhino3dm.Mesh]*) – A list, an array or any enumerable of meshes.
- **direction** (*rhino3dm.Vector3d*) – A direction vector.
- **tolerance** (*float*) – A tolerance value.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve array.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToMesh2**(*curves*, *meshes*, *direction*, *tolerance*, *multiple=false*)
Projects a curve to a set of meshes using a direction and tolerance.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – A list, an array or any enumerable of curves.
- **meshes** (*list[rhino3dm.Mesh]*) – A list, an array or any enumerable of meshes.
- **direction** (*rhino3dm.Vector3d*) – A direction vector.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve array.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToBrep**(*curve*, *brep*, *direction*, *tolerance*, *multiple=false*)
Projects a Curve onto a Brep along a given direction.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to project.
- **brep** (*rhino3dm.Brep*) – Brep to project onto.
- **direction** (*rhino3dm.Vector3d*) – Direction of projection.
- **tolerance** (*float*) – Tolerance to use for projection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of projected curves or empty array if the projection set is empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToBrep1**(*curve*, *breps*, *direction*, *tolerance*, *multiple=false*)
Projects a Curve onto a collection of Breps along a given direction.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to project.
- **breps** (*list[rhino3dm.Brep]*) – Breps to project onto.
- **direction** (*rhino3dm.Vector3d*) – Direction of projection.
- **tolerance** (*float*) – Tolerance to use for projection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of projected curves or empty array if the projection set is empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToBrep2**(*curve*, *breps*, *direction*, *tolerance*, *multiple=false*)
Projects a Curve onto a collection of Breps along a given direction.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to project.
- **breps** (*list[rhino3dm.Brep]*) – Breps to project onto.
- **direction** (*rhino3dm.Vector3d*) – Direction of projection.
- **tolerance** (*float*) – Tolerance to use for projection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of projected curves or None if the projection set is empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToBrep3**(*curves*, *breps*, *direction*, *tolerance*, *multiple=false*)

Projects a collection of Curves onto a collection of Breps along a given direction.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – Curves to project.
- **breps** (*list[rhino3dm.Brep]*) – Breps to project onto.
- **direction** (*rhino3dm.Vector3d*) – Direction of projection.
- **tolerance** (*float*) – Tolerance to use for projection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of projected curves or empty array if the projection set is empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToBrep4**(*curves*, *breps*, *direction*, *tolerance*, *multiple=false*)

Projects a collection of Curves onto a collection of Breps along a given direction.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – Curves to project.
- **breps** (*list[rhino3dm.Brep]*) – Breps to project onto.
- **direction** (*rhino3dm.Vector3d*) – Direction of projection.
- **tolerance** (*float*) – Tolerance to use for projection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of projected curves. Array is empty if the projection set is empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**projectToPlane**(*curve*, *plane*, *multiple=false*)

Constructs a curve by projecting an existing curve to a plane.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve.
- **plane** (*rhino3dm.Plane*) – A plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The projected curve on success; None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**pullToBrepFace**(*curve*, *face*, *tolerance*, *multiple=false*)

Pull a curve to a BrepFace using closest point projection.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to pull.
- **face** (*rhino3dm.BrepFace*) – Brep face that pulls.
- **tolerance** (*float*) – Tolerance to use for pulling.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of pulled curves, or an empty array on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**planarClosedCurveRelationship**(*curveA*, *curveB*, *testPlane*, *tolerance*, *multiple=false*)

Determines whether two coplanar simple closed curves are disjoint or intersect; otherwise, if the regions have a containment relationship, discovers which curve encloses the other.

Arguments

- **curveA** (*rhino3dm.Curve*) – A first curve.
- **curveB** (*rhino3dm.Curve*) – A second curve.
- **testPlane** (*rhino3dm.Plane*) – A plane.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A value indicating the relationship between the first and the second curve.

Return type RegionContainment

RhinoCompute.Curve.**planarCurveCollision**(*curveA*, *curveB*, *testPlane*, *tolerance*, *multiple=false*)

Determines if two coplanar curves collide (intersect).

Arguments

- **curveA** (*rhino3dm.Curve*) – A curve.
- **curveB** (*rhino3dm.Curve*) – Another curve.
- **testPlane** (*rhino3dm.Plane*) – A valid plane containing the curves.
- **tolerance** (*float*) – A tolerance value for intersection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the curves intersect, otherwise false

Return type bool

RhinoCompute.Curve.**duplicateSegments**(*thisCurve*, *multiple=false*)

Duplicates curve segments. Explodes polylines, polycurves and G1 discontinuous NURBS curves. Single segment curves, such as lines, arcs, unkinked NURBS curves, are duplicated.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of all the segments that make up this curve.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**smooth**(*thisCurve*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *multiple=false*)

Smooths a curve by averaging the positions of control points in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much control points move towards the average of the neighboring control points.
- **bXSmooth** (*bool*) – When True control points move in X axis direction.
- **bYSmooth** (*bool*) – When True control points move in Y axis direction.
- **bZSmooth** (*bool*) – When True control points move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True the curve ends don't move.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The smoothed curve if successful, None otherwise.

Return type rhino3dm.Curve

RhinoCompute.Curve.**smooth1**(*thisCurve*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *plane*, *multiple=false*)

Smooths a curve by averaging the positions of control points in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much control points move towards the average of the neighboring control points.
- **bXSmooth** (*bool*) – When True control points move in X axis direction.
- **bYSmooth** (*bool*) – When True control points move in Y axis direction.
- **bZSmooth** (*bool*) – When True control points move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True the curve ends don't move.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **plane** (*rhino3dm.Plane*) – If SmoothingCoordinateSystem.CPlane specified, then the construction plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The smoothed curve if successful, None otherwise.

Return type rhino3dm.Curve

RhinoCompute.Curve.**getLocalPerpPoint**(*thisCurve*, *testPoint*, *seedParmameter*, *multiple=false*)

Search for a location on the curve, near seedParmameter, that is perpendicular to a test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – The test point.
- **seedParmameter** (*float*) – A "seed" parameter on the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if a solution is found, False otherwise.

Return type bool

RhinoCompute.Curve.**getLocalPerpPoint1**(*thisCurve*, *testPoint*, *seedParmameter*, *subDomain*, *multiple=false*)

Search for a location on the curve, near seedParmameter, that is perpendicular to a test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – The test point.
- **seedParmameter** (*float*) – A "seed" parameter on the curve.
- **subDomain** (*rhino3dm.Interval*) – The sub-domain of the curve to search.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if a solution is found, False otherwise.

Return type bool

RhinoCompute.Curve.**getLocalTangentPoint**(*thisCurve*, *testPoint*, *seedParmameter*, *multiple=false*)

Search for a location on the curve, near seedParmameter, that is tangent to a test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – The test point.
- **seedParmameter** (*float*) – A "seed" parameter on the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if a solution is found, False otherwise.

Return type bool

RhinoCompute.Curve.**getLocalTangentPoint1**(*thisCurve*, *testPoint*, *seedParmameter*, *subDomain*, *multiple=false*)

Search for a location on the curve, near seedParmameter, that is tangent to a test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – The test point.
- **seedParmameter** (*float*) – A "seed" parameter on the curve.
- **subDomain** (*rhino3dm.Interval*) – The sub-domain of the curve to search.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if a solution is found, False otherwise.

Return type bool

RhinoCompute.Curve.**inflectionPoints**(*thisCurve*, *multiple=false*)

Returns a curve's inflection points. An inflection point is a location on a curve at which the sign of the curvature (i.e., the concavity) changes. The curvature at these locations is always 0.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points if successful, None if not successful or on error.

Return type rhino3dm.Point3d[]

RhinoCompute.Curve.**maxCurvaturePoints**(*thisCurve*, *multiple=false*)

Returns a curve's maximum curvature points. The maximum curvature points identify where the curvature starts to decrease in both directions from the points.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points if successful, None if not successful or on error.

Return type rhino3dm.Point3d[]

RhinoCompute.Curve.**makeClosed**(*thisCurve*, *tolerance*, *multiple=false*)

If IsClosed, just return true. Otherwise, decide if curve can be closed as follows: Linear curves polylinear curves with 2 segments, NURBS with 3 or less control points cannot be made closed. Also, if tolerance > 0 and the gap between start and end is larger than tolerance, curve cannot be made closed. Adjust the curve's endpoint to match its start point.

Arguments

- **tolerance** (*float*) – If nonzero, and the gap is more than tolerance, curve cannot be made closed.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**combineShortSegments**(*thisCurve*, *tolerance*, *multiple=false*)

Looks for segments that are shorter than tolerance that can be combined. For NURBS of degree greater than 1, spans are combined by removing knots. Similarly for NURBS segments of polycurves. Otherwise, RemoveShortSegments() is called. Does not change the domain, but it will change the relative parameterization.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if short segments were combined or removed. False otherwise.

Return type bool

RhinoCompute.Curve.**lcoalClosestPoint**(*thisCurve*, *testPoint*, *seed*, *multiple=false*)

Find parameter of the point on a curve that is locally closest to the testPoint. The search for a local close point starts at a seed parameter.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – A point to test against.

- **seed** (*float*) – The seed parameter.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the search is successful, False if the search fails.

Return type bool

RhinoCompute.Curve.**localClosestPoint**(*thisCurve*, *testPoint*, *seed*, *multiple=false*)

Find parameter of the point on a curve that is locally closest to the testPoint. The search for a local close point starts at a seed parameter.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – A point to test against.
- **seed** (*float*) – The seed parameter.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the search is successful, False if the search fails.

Return type bool

RhinoCompute.Curve.**closestPoint**(*thisCurve*, *testPoint*, *multiple=false*)

Finds parameter of the point on a curve that is closest to testPoint. If the maximumDistance parameter is > 0, then only points whose distance to the given point is <= maximumDistance will be returned. Using a positive value of maximumDistance can substantially speed up the search.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to search from.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**closestPoint1**(*thisCurve*, *testPoint*, *maximumDistance*, *multiple=false*)

Finds the parameter of the point on a curve that is closest to testPoint. If the maximumDistance parameter is > 0, then only points whose distance to the given point is <= maximumDistance will be returned. Using a positive value of maximumDistance can substantially speed up the search.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to project.
- **maximumDistance** (*float*) – The maximum allowed distance. Past this distance, the search is given up and False is returned.Use 0 to turn off this parameter.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**closestPoints**(*thisCurve*, *otherCurve*, *multiple=false*)

Gets closest points between this and another curves.

Arguments

- **otherCurve** (*rhino3dm.Curve*) – The other curve.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success; False on error.

Return type bool

RhinoCompute.Curve.**contains**(*thisCurve*, *testPoint*, *multiple=false*)

Computes the relationship between a point and a closed curve region. This curve must be closed or the return value will be Unset. Both curve and point are projected to the World XY plane.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to test.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Relationship between point and curve region.

Return type PointContainment

RhinoCompute.Curve.**contains1**(*thisCurve*, *testPoint*, *plane*, *multiple=false*)

Computes the relationship between a point and a closed curve region. This curve must be closed or the return value will be Unset.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to test.
- **plane** (*rhino3dm.Plane*) – Plane in which to compare point and region.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Relationship between point and curve region.

Return type PointContainment

RhinoCompute.Curve.**contains2**(*thisCurve*, *testPoint*, *plane*, *tolerance*, *multiple=false*)

Computes the relationship between a point and a closed curve region. This curve must be closed or the return value will be Unset.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to test.
- **plane** (*rhino3dm.Plane*) – Plane in which to compare point and region.
- **tolerance** (*float*) – Tolerance to use during comparison.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Relationship between point and curve region.

Return type PointContainment

RhinoCompute.Curve.**extremeParameters**(*thisCurve*, *direction*, *multiple=false*)

Returns the parameter values of all local extrema. Parameter values are in increasing order so consecutive extrema define an interval on which each component of the curve is monotone. Note, non-periodic curves always return the end points.

Arguments

- **direction** (*rhino3dm.Vector3d*) – The direction in which to perform the calculation.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The parameter values of all local extrema.

Return type float[]

RhinoCompute.Curve.**createPeriodicCurve**(*curve*, *multiple=false*)

Removes kinks from a curve. Periodic curves deform smoothly without kinks.

Arguments

- **curve** (*rhino3dm.Curve*) – The curve to make periodic. Curve must have degree >= 2.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting curve if successful, None otherwise.

Return type rhino3dm.Curve

RhinoCompute.Curve.**createPeriodicCurve1**(*curve*, *smooth*, *multiple=false*)

Removes kinks from a curve. Periodic curves deform smoothly without kinks.

Arguments

- **curve** (*rhino3dm.Curve*) – The curve to make periodic. Curve must have degree >= 2.
- **smooth** (*bool*) – If true, smooths any kinks in the curve and moves control points to make a smooth curve. If false, control point locations are not changed or changed minimally (only one point may move) and only the knot vector is altered.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting curve if successful, None otherwise.

Return type rhino3dm.Curve

RhinoCompute.Curve.**pointAtLength**(*thisCurve*, *length*, *multiple=false*)

Gets a point at a certain length along the curve. The length must be non-negative and less than or equal to the length of the curve. Lengths will not be wrapped when the curve is closed or periodic.

Arguments

- **length** (*float*) – Length along the curve between the start point and the returned point.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Point on the curve at the specified length from the start point or Poin3d.Unset on failure.

Return type rhino3dm.Point3d

RhinoCompute.Curve.**pointAtNormalizedLength**(*thisCurve*, *length*, *multiple=false*)

Gets a point at a certain normalized length along the curve. The length must be between or including 0.0 and 1.0, where 0.0 equals the start of the curve and 1.0 equals the end of the curve.

Arguments

- **length** (*float*) – Normalized length along the curve between the start point and the returned point.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Point on the curve at the specified normalized length from the start point or Poin3d.Unset on failure.

Return type rhino3dm.Point3d

RhinoCompute.Curve.**perpendicularFrameAt**(*thisCurve*, *t*, *multiple=false*)

Return a 3d frame at a parameter. This is slightly different than FrameAt in that the frame is computed in a way so there is minimal rotation from one frame to the next.

Arguments

- **t** (*float*) – Evaluation parameter.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**getPerpendicularFrames**(*thisCurve*, *parameters*, *multiple=false*)

Gets a collection of perpendicular frames along the curve. Perpendicular frames are also known as 'Zero-twisting frames' and they minimize rotation from one frame to the next.

Arguments

- **parameters** (*list[float]*) – A collection of strictly increasing curve parameters to place perpendicular frames on.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of perpendicular frames on success or None on failure.

Return type rhino3dm.Plane[]

RhinoCompute.Curve.**getLength**(*thisCurve*, *multiple=false*)

Gets the length of the curve with a fractional tolerance of 1.0e-8.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The length of the curve on success, or zero on failure.

Return type float

RhinoCompute.Curve.**getLength1**(*thisCurve*, *fractionalTolerance*, *multiple=false*)

Get the length of the curve.

Arguments

- **fractionalTolerance** (*float*) – Desired fractional precision. fabs(("exact" length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The length of the curve on success, or zero on failure.

Return type float

RhinoCompute.Curve.**getLength2**(*thisCurve*, *subdomain*, *multiple=false*)
Get the length of a sub-section of the curve with a fractional tolerance of 1e-8.

Arguments

- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve (must be non-decreasing).
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The length of the sub-curve on success, or zero on failure.

Return type float

RhinoCompute.Curve.**getLength3**(*thisCurve*, *fractionalTolerance*, *subdomain*, *multiple=false*)
Get the length of a sub-section of the curve.

Arguments

- **fractionalTolerance** (*float*) – Desired fractional precision. fabs(("exact" length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve (must be non-decreasing).
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The length of the sub-curve on success, or zero on failure.

Return type float

RhinoCompute.Curve.**isShort**(*thisCurve*, *tolerance*, *multiple=false*)
Used to quickly find short curves.

Arguments

- **tolerance** (*float*) – Length threshold value for "shortness".
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the length of the curve is <= tolerance.

Return type bool

RhinoCompute.Curve.**isShort1**(*thisCurve*, *tolerance*, *subdomain*, *multiple=false*)
Used to quickly find short curves.

Arguments

- **tolerance** (*float*) – Length threshold value for "shortness".
- **subdomain** (*rhino3dm.Interval*) – The test is performed on the interval that is the intersection of sub-domain with Domain()
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the length of the curve is <= tolerance.

Return type bool

RhinoCompute.Curve.**removeShortSegments**(*thisCurve*, *tolerance*, *multiple=false*)
Looks for segments that are shorter than tolerance that can be removed. Does not change the domain, but it will change the relative parameterization.

Arguments

- **tolerance** (*float*) – Tolerance which defines "short" segments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if removable short segments were found. False if no removable short segments were found.

Return type bool

RhinoCompute.Curve.**lengthParameter**(*thisCurve*, *segmentLength*, *multiple=false*)

Gets the parameter along the curve which coincides with a given length along the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **segmentLength** (*float*) – Length of segment to measure. Must be less than or equal to the length of the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**lengthParameter1**(*thisCurve*, *segmentLength*, *fractionalTolerance*, *multiple=false*)

Gets the parameter along the curve which coincides with a given length along the curve.

Arguments

- **segmentLength** (*float*) – Length of segment to measure. Must be less than or equal to the length of the curve.
- **fractionalTolerance** (*float*) – Desired fractional precision. fabs(("exact" length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**lengthParameter2**(*thisCurve*, *segmentLength*, *subdomain*, *multiple=false*)

Gets the parameter along the curve which coincides with a given length along the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **segmentLength** (*float*) – Length of segment to measure. Must be less than or equal to the length of the sub-domain.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve rather than the whole curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**lengthParameter3**(*thisCurve*, *segmentLength*, *fractionalTolerance*, *subdomain*, *multiple=false*)

Gets the parameter along the curve which coincides with a given length along the curve.

Arguments

- **segmentLength** (*float*) – Length of segment to measure. Must be less than or equal to the length of the sub-domain.
- **fractionalTolerance** (*float*) – Desired fractional precision. fabs((“exact” length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve rather than the whole curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**normalizedLengthParameter**(*thisCurve*, *s*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **s** (*float*) – Normalized arc length parameter. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**normalizedLengthParameter1**(*thisCurve*, *s*, *fractionalTolerance*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve.

Arguments

- **s** (*float*) – Normalized arc length parameter. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **fractionalTolerance** (*float*) – Desired fractional precision. fabs((“exact” length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**normalizedLengthParameter2**(*thisCurve*, *s*, *subdomain*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **s** (*float*) – Normalized arc length parameter. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.

- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**normalizedLengthParameter3**(*thisCurve*, *s*, *fractionalTolerance*, *subdomain*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve.

Arguments

- **s** (*float*) – Normalized arc length parameter. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **fractionalTolerance** (*float*) – Desired fractional precision. fabs((“exact” length from start to t) - arc_length)/arc_length <= fractionalTolerance.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Curve.**normalizedLengthParameters**(*thisCurve*, *s*, *absoluteTolerance*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **s** (*float[]*) – Array of normalized arc length parameters. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **absoluteTolerance** (*float*) – If absoluteTolerance > 0, then the difference between (s[i+1]-s[i])*curve_length and the length of the curve segment from t[i] to t[i+1] will be <= absoluteTolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If successful, array of curve parameters such that the length of the curve from its start to t[i] is s[i]*curve_length. Null on failure.

Return type float[]

RhinoCompute.Curve.**normalizedLengthParameters1**(*thisCurve*, *s*, *absoluteTolerance*, *fractionalTolerance*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve.

Arguments

- **s** (*float[]*) – Array of normalized arc length parameters. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.

- **absoluteTolerance** (*float*) – If absoluteTolerance > 0, then the difference between (s[i+1]-s[i])*curve_length and the length of the curve segment from t[i] to t[i+1] will be <= absoluteTolerance.
- **fractionalTolerance** (*float*) – Desired fractional precision for each segment. fabs("true" length - actual length)/(actual length) <= fractionalTolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If successful, array of curve parameters such that the length of the curve from its start to t[i] is s[i]*curve_length. Null on failure.

Return type float[]

RhinoCompute.Curve.**normalizedLengthParameters2**(*thisCurve*, *s*, *absoluteTolerance*, *subdomain*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve. A fractional tolerance of 1e-8 is used in this version of the function.

Arguments

- **s** (*float[]*) – Array of normalized arc length parameters. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **absoluteTolerance** (*float*) – If absoluteTolerance > 0, then the difference between (s[i+1]-s[i])*curve_length and the length of the curve segment from t[i] to t[i+1] will be <= absoluteTolerance.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve. A 0.0 s value corresponds to sub-domain->Min() and a 1.0 s value corresponds to sub-domain->Max().
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If successful, array of curve parameters such that the length of the curve from its start to t[i] is s[i]*curve_length. Null on failure.

Return type float[]

RhinoCompute.Curve.**normalizedLengthParameters3**(*thisCurve*, *s*, *absoluteTolerance*, *fractionalTolerance*, *subdomain*, *multiple=false*)

Input the parameter of the point on the curve that is a prescribed arc length from the start of the curve.

Arguments

- **s** (*float[]*) – Array of normalized arc length parameters. E.g., 0 = start of curve, 1/2 = midpoint of curve, 1 = end of curve.
- **absoluteTolerance** (*float*) – If absoluteTolerance > 0, then the difference between (s[i+1]-s[i])*curve_length and the length of the curve segment from t[i] to t[i+1] will be <= absoluteTolerance.
- **fractionalTolerance** (*float*) – Desired fractional precision for each segment. fabs("true" length - actual length)/(actual length) <= fractionalTolerance.
- **subdomain** (*rhino3dm.Interval*) – The calculation is performed on the specified sub-domain of the curve. A 0.0 s value corresponds to sub-domain->Min() and a 1.0 s value corresponds to sub-domain->Max().
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If successful, array of curve parameters such that the length of the curve from its start to t[i] is s[i]*curve_length. Null on failure.

Return type float[]

RhinoCompute.Curve.**divideByCount**(*thisCurve*, *segmentCount*, *includeEnds*, *multiple=false*)

Divide the curve into a number of equal-length segments.

Arguments

- **segmentCount** (*int*) – Segment count. Note that the number of division points may differ from the segment count.
- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns List of curve parameters at the division points on success, None on failure.

Return type float[]

RhinoCompute.Curve.**divideByCount1**(*thisCurve*, *segmentCount*, *includeEnds*, *multiple=false*)

Divide the curve into a number of equal-length segments.

Arguments

- **segmentCount** (*int*) – Segment count. Note that the number of division points may differ from the segment count.
- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array containing division curve parameters on success, None on failure.

Return type float[]

RhinoCompute.Curve.**divideByLength**(*thisCurve*, *segmentLength*, *includeEnds*, *multiple=false*)

Divide the curve into specific length segments.

Arguments

- **segmentLength** (*float*) – The length of each and every segment (except potentially the last one).
- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array containing division curve parameters if successful, None on failure.

Return type float[]

RhinoCompute.Curve.**divideByLength1**(*thisCurve*, *segmentLength*, *includeEnds*, *reverse*, *multiple=false*)

Divide the curve into specific length segments.

Arguments

- **segmentLength** (*float*) – The length of each and every segment (except potentially the last one).

- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **reverse** (*bool*) – If true, then the divisions start from the end of the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array containing division curve parameters if successful, None on failure.

Return type float[]

RhinoCompute.Curve.**divideByLength2**(*thisCurve*, *segmentLength*, *includeEnds*, *multiple=false*)

Divide the curve into specific length segments.

Arguments

- **segmentLength** (*float*) – The length of each and every segment (except potentially the last one).
- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array containing division curve parameters if successful, None on failure.

Return type float[]

RhinoCompute.Curve.**divideByLength3**(*thisCurve*, *segmentLength*, *includeEnds*, *reverse*, *multiple=false*)

Divide the curve into specific length segments.

Arguments

- **segmentLength** (*float*) – The length of each and every segment (except potentially the last one).
- **includeEnds** (*bool*) – If true, then the point at the start of the first division segment is returned.
- **reverse** (*bool*) – If true, then the divisions start from the end of the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array containing division curve parameters if successful, None on failure.

Return type float[]

RhinoCompute.Curve.**divideEquidistant**(*thisCurve*, *distance*, *multiple=false*)

Calculates 3d points on a curve where the linear distance between the points is equal.

Arguments

- **distance** (*float*) – The distance between division points.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of equidistant points, or None on error.

Return type rhino3dm.Point3d[]

RhinoCompute.Curve.**divideAsContour**(*thisCurve*, *contourStart*, *contourEnd*, *interval*, *multiple=false*)

Divides this curve at fixed steps along a defined contour line.

Arguments

- **contourStart** (*rhino3dm.Point3d*) – The start of the contouring line.
- **contourEnd** (*rhino3dm.Point3d*) – The end of the contouring line.
- **interval** (*float*) – A distance to measure on the contouring axis.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points; or None on error.

Return type rhino3dm.Point3d[]

RhinoCompute.Curve.**trim**(*thisCurve*, *side*, *length*, *multiple=false*)

Shortens a curve by a given length

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Trimmed curve if successful, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**split**(*thisCurve*, *cutter*, *tolerance*, *multiple=false*)

Splits a curve into pieces using a polysurface.

Arguments

- **cutter** (*rhino3dm.Brep*) – A cutting surface or polysurface.
- **tolerance** (*float*) – A tolerance for computing intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**split1**(*thisCurve*, *cutter*, *tolerance*, *angleToleranceRadians*, *multiple=false*)

Splits a curve into pieces using a polysurface.

Arguments

- **cutter** (*rhino3dm.Brep*) – A cutting surface or polysurface.
- **tolerance** (*float*) – A tolerance for computing intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**split2**(*thisCurve*, *cutter*, *tolerance*, *multiple=false*)

Splits a curve into pieces using a surface.

Arguments

- **cutter** (*rhino3dm.Surface*) – A cutting surface or polysurface.

- **tolerance** (*float*) – A tolerance for computing intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**split3**(*thisCurve*, *cutter*, *tolerance*, *angleToleranceRadians*, *multiple=false*)

Splits a curve into pieces using a surface.

Arguments

- **cutter** (*rhino3dm.Surface*) – A cutting surface or polysurface.
- **tolerance** (*float*) – A tolerance for computing intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**extend**(*thisCurve*, *t0*, *t1*, *multiple=false*)

Where possible, analytically extends curve to include the given domain. This will not work on closed curves. The original curve will be identical to the restriction of the resulting curve to the original curve domain.

Arguments

- **t0** (*float*) – Start of extension domain, if the start is not inside the Domain of this curve, an attempt will be made to extend the curve.
- **t1** (*float*) – End of extension domain, if the end is not inside the Domain of this curve, an attempt will be made to extend the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Extended curve on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extend1**(*thisCurve*, *domain*, *multiple=false*)

Where possible, analytically extends curve to include the given domain. This will not work on closed curves. The original curve will be identical to the restriction of the resulting curve to the original curve domain.

Arguments

- **domain** (*rhino3dm.Interval*) – Extension domain.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Extended curve on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extend2**(*thisCurve*, *side*, *length*, *style*, *multiple=false*)

Extends a curve by a specific length.

Arguments

- **side** (*CurveEnd*) – Curve end to extend.
- **length** (*float*) – Length to add to the curve end.

- **style** (*CurveExtensionStyle*) – Extension style.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve with extended ends or None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extend3**(*thisCurve*, *side*, *style*, *geometry*, *multiple=false*)
Extends a curve until it intersects a collection of objects.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.
- **style** (*CurveExtensionStyle*) – The style or type of extension to use.
- **geometry** (*System.Collections.Generic.IEnumerable<GeometryBase>*) – A collection of objects. Allowable object types are Curve, Surface, Brep.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extend4**(*thisCurve*, *side*, *style*, *endPoint*, *multiple=false*)
Extends a curve to a point.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.
- **style** (*CurveExtensionStyle*) – The style or type of extension to use.
- **endPoint** (*rhino3dm.Point3d*) – A new end point.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extendOnSurface**(*thisCurve*, *side*, *surface*, *multiple=false*)
Extends a curve on a surface.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.
- **surface** (*rhino3dm.Surface*) – Surface that contains the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extendOnSurface1**(*thisCurve*, *side*, *face*, *multiple=false*)
Extends a curve on a surface.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.

- **face** (*rhino3dm.BrepFace*) – BrepFace that contains the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extendByLine**(*thisCurve*, *side*, *geometry*, *multiple=false*)

Extends a curve by a line until it intersects a collection of objects.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.
- **geometry** (*System.Collections.Generic.IEnumerable<GeometryBase>*) – A collection of objects. Allowable object types are Curve, Surface, Brep.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**extendByArc**(*thisCurve*, *side*, *geometry*, *multiple=false*)

Extends a curve by an Arc until it intersects a collection of objects.

Arguments

- **side** (*CurveEnd*) – The end of the curve to extend.
- **geometry** (*System.Collections.Generic.IEnumerable<GeometryBase>*) – A collection of objects. Allowable object types are Curve, Surface, Brep.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended curve result on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**simplify**(*thisCurve*, *options*, *distanceTolerance*, *angleToleranceRadians*, *multiple=false*)

Returns a geometrically equivalent PolyCurve. The PolyCurve has the following properties 1. All the PolyCurve segments are LineCurve, PolylineCurve, ArcCurve, or NurbsCurve. 2. The NURBS Curves segments do not have fully multiple interior knots. 3. Rational NURBS curves do not have constant weights. 4. Any segment for which IsLinear() or IsArc() is True is a Line, Polyline segment, or an Arc. 5. Adjacent co-linear or co-circular segments are combined. 6. Segments that meet with G1-continuity have there ends tuned up so that they meet with G1-continuity to within machine precision.

Arguments

- **options** (*CurveSimplifyOptions*) – Simplification options.
- **distanceTolerance** (*float*) – A distance tolerance for the simplification.
- **angleToleranceRadians** (*float*) – An angle tolerance for the simplification.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New simplified curve on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**simplifyEnd**(*thisCurve*, *end*, *options*, *distanceTolerance*, *angleToleranceRadians*, *multiple=false*)

Same as SimplifyCurve, but simplifies only the last two segments at "side" end.

Arguments

- **end** (*CurveEnd*) – If CurveEnd.Start the function simplifies the last two start side segments, otherwise if CurveEnd.End the last two end side segments are simplified.
- **options** (*CurveSimplifyOptions*) – Simplification options.
- **distanceTolerance** (*float*) – A distance tolerance for the simplification.
- **angleToleranceRadians** (*float*) – An angle tolerance for the simplification.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New simplified curve on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**fair**(*thisCurve*, *distanceTolerance*, *angleTolerance*, *clampStart*, *clampEnd*, *iterations*, *multiple=false*)

Fairs a curve object. Fair works best on degree 3 (cubic) curves. Attempts to remove large curvature variations while limiting the geometry changes to be no more than the specified tolerance.

Arguments

- **distanceTolerance** (*float*) – Maximum allowed distance the faired curve is allowed to deviate from the input.
- **angleTolerance** (*float*) – (in radians) kinks with angles <= angleTolerance are smoothed out 0.05 is a good default.
- **clampStart** (*int*) – The number of (control vertices-1) to preserve at start. 0 = preserve start point1 = preserve start point and 1st derivative2 = preserve start point, 1st and 2nd derivative
- **clampEnd** (*int*) – Same as clampStart.
- **iterations** (*int*) – The number of iterations to use in adjusting the curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns new faired Curve on success, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**fit**(*thisCurve*, *degree*, *fitTolerance*, *angleTolerance*, *multiple=false*)

Fits a new curve through an existing curve.

Arguments

- **degree** (*int*) – The degree of the returned Curve. Must be bigger than 1.
- **fitTolerance** (*float*) – The fitting tolerance. If fitTolerance is RhinoMath.UnsetValue or <=0.0, the document absolute tolerance is used.
- **angleTolerance** (*float*) – The kink smoothing tolerance in radians. If angleTolerance is 0.0, all kinks are smoothedIf angleTolerance is >0.0, kinks smaller than angleTolerance are smoothedIf angleTolerance is RhinoMath.UnsetValue or <0.0, the document angle tolerance is used for the kink smoothing

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a new fitted Curve if successful, None on failure.

Return type rhino3dm.Curve

RhinoCompute.Curve.**rebuild**(*thisCurve*, *pointCount*, *degree*, *preserveTangents*, *multiple=false*)

Rebuild a curve with a specific point count.

Arguments

- **pointCount** (*int*) – Number of control points in the rebuild curve.
- **degree** (*int*) – Degree of curve. Valid values are between and including 1 and 11.
- **preserveTangents** (*bool*) – If true, the end tangents of the input curve will be preserved.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NURBS curve on success or None on failure.

Return type rhino3dm.NurbsCurve

RhinoCompute.Curve.**toPolyline**(*thisCurve*, *mainSegmentCount*, *subSegmentCount*, *maxAngleRadians*, *maxChordLengthRatio*, *maxAspectRatio*, *tolerance*, *minEdgeLength*, *maxEdgeLength*, *keepStartPoint*, *multiple=false*)

Gets a polyline approximation of a curve.

Arguments

- **mainSegmentCount** (*int*) – If mainSegmentCount <= 0, then both subSegmentCount and mainSegmentCount are ignored. If mainSegmentCount > 0, then subSegmentCount must be >= 1. In this case the NURBS will be broken into mainSegmentCount equally spaced chords. If needed, each of these chords can be split into as many subSegmentCount sub-parts if the subdivision is necessary for the mesh to meet the other meshing constraints. In particular, if subSegmentCount = 0, then the curve is broken into mainSegmentCount pieces and no further testing is performed.
- **subSegmentCount** (*int*) – An amount of subsegments.
- **maxAngleRadians** (*float*) – (0 to pi) Maximum angle (in radians) between unit tangents at adjacent vertices.
- **maxChordLengthRatio** (*float*) – Maximum permitted value of (distance chord midpoint to curve) / (length of chord).
- **maxAspectRatio** (*float*) – If maxAspectRatio < 1.0, the parameter is ignored. If 1 <= maxAspectRatio < sqrt(2), it is treated as if maxAspectRatio = sqrt(2). This parameter controls the maximum permitted value of (length of longest chord) / (length of shortest chord).
- **tolerance** (*float*) – If tolerance = 0, the parameter is ignored. This parameter controls the maximum permitted value of the distance from the curve to the polyline.
- **minEdgeLength** (*float*) – The minimum permitted edge length.
- **maxEdgeLength** (*float*) – If maxEdgeLength = 0, the parameter is ignored. This parameter controls the maximum permitted edge length.
- **keepStartPoint** (*bool*) – If True the starting point of the curve is added to the polyline. If False the starting point of the curve is not added to the polyline.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns PolylineCurve on success, None on error.

Return type PolylineCurve

RhinoCompute.Curve.**toPolyline1**(*thisCurve*, *mainSegmentCount*, *subSegmentCount*, *maxAngleRadians*, *maxChordLengthRatio*, *maxAspectRatio*, *tolerance*, *minEdgeLength*, *maxEdgeLength*, *keepStartPoint*, *curveDomain*, *multiple=false*)

Gets a polyline approximation of a curve.

Arguments

- **mainSegmentCount** (*int*) – If mainSegmentCount <= 0, then both subSegmentCount and mainSegmentCount are ignored. If mainSegmentCount > 0, then subSegmentCount must be >= 1. In this case the NURBS will be broken into mainSegmentCount equally spaced chords. If needed, each of these chords can be split into as many subSegmentCount sub-parts if the subdivision is necessary for the mesh to meet the other meshing constraints. In particular, if subSegmentCount = 0, then the curve is broken into mainSegmentCount pieces and no further testing is performed.
- **subSegmentCount** (*int*) – An amount of subsegments.
- **maxAngleRadians** (*float*) – (0 to pi) Maximum angle (in radians) between unit tangents at adjacent vertices.
- **maxChordLengthRatio** (*float*) – Maximum permitted value of (distance chord midpoint to curve) / (length of chord).
- **maxAspectRatio** (*float*) – If maxAspectRatio < 1.0, the parameter is ignored. If 1 <= maxAspectRatio < sqrt(2), it is treated as if maxAspectRatio = sqrt(2). This parameter controls the maximum permitted value of (length of longest chord) / (length of shortest chord).
- **tolerance** (*float*) – If tolerance = 0, the parameter is ignored. This parameter controls the maximum permitted value of the distance from the curve to the polyline.
- **minEdgeLength** (*float*) – The minimum permitted edge length.
- **maxEdgeLength** (*float*) – If maxEdgeLength = 0, the parameter is ignored. This parameter controls the maximum permitted edge length.
- **keepStartPoint** (*bool*) – If True the starting point of the curve is added to the polyline. If False the starting point of the curve is not added to the polyline.
- **curveDomain** (*rhino3dm.Interval*) – This sub-domain of the NURBS curve is approximated.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns PolylineCurve on success, None on error.

Return type PolylineCurve

RhinoCompute.Curve.**toPolyline2**(*thisCurve*, *tolerance*, *angleTolerance*, *minimumLength*, *maximumLength*, *multiple=false*)

Gets a polyline approximation of a curve.

Arguments

- **tolerance** (*float*) – The tolerance. This is the maximum deviation from line midpoints to the curve. When in doubt, use the document's model space absolute tolerance.
- **angleTolerance** (*float*) – The angle tolerance in radians. This is the maximum deviation of the line directions. When in doubt, use the document's model space angle tolerance.
- **minimumLength** (*float*) – The minimum segment length.
- **maximumLength** (*float*) – The maximum segment length.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns PolyCurve on success, None on error.

Return type PolylineCurve

RhinoCompute.Curve.**toArcsAndLines**(*thisCurve*, *tolerance*, *angleTolerance*, *minimumLength*, *maximumLength*, *multiple=false*)

Converts a curve into polycurve consisting of arc segments. Sections of the input curves that are nearly straight are converted to straight-line segments.

Arguments

- **tolerance** (*float*) – The tolerance. This is the maximum deviation from arc midpoints to the curve. When in doubt, use the document's model space absolute tolerance.
- **angleTolerance** (*float*) – The angle tolerance in radians. This is the maximum deviation of the arc end directions from the curve direction. When in doubt, use the document's model space angle tolerance.
- **minimumLength** (*float*) – The minimum segment length.
- **maximumLength** (*float*) – The maximum segment length.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns PolyCurve on success, None on error.

Return type PolyCurve

RhinoCompute.Curve.**pullToMesh**(*thisCurve*, *mesh*, *tolerance*, *multiple=false*)

Makes a polyline approximation of the curve and gets the closest point on the mesh for each point on the curve. Then it "connects the points" so that you have a polyline on the mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to project onto.
- **tolerance** (*float*) – Input tolerance (RhinoDoc.ModelAbsoluteTolerance is a good default)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A polyline curve on success, None on failure.

Return type PolylineCurve

RhinoCompute.Curve.**offset**(*thisCurve*, *plane*, *distance*, *tolerance*, *cornerStyle*, *multiple=false*)

Offsets this curve. If you have a nice offset, then there will be one entry in the array. If the original curve had kinks or the offset curve had self intersections, you will get multiple segments in the output array.

Arguments

- **plane** (*rhino3dm.Plane*) – Offset solution plane.

- **distance** (*float*) – The positive or negative distance to offset.
- **tolerance** (*float*) – The offset or fitting tolerance.
- **cornerStyle** (*CurveOffsetCornerStyle*) – Corner style for offset kinks.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offset1**(*thisCurve*, *directionPoint*, *normal*, *distance*, *tolerance*, *cornerStyle*, *multiple=false*)

Offsets this curve. If you have a nice offset, then there will be one entry in the array. If the original curve had kinks or the offset curve had self intersections, you will get multiple segments in the output array.

Arguments

- **directionPoint** (*rhino3dm.Point3d*) – A point that indicates the direction of the offset.
- **normal** (*rhino3dm.Vector3d*) – The normal to the offset plane.
- **distance** (*float*) – The positive or negative distance to offset.
- **tolerance** (*float*) – The offset or fitting tolerance.
- **cornerStyle** (*CurveOffsetCornerStyle*) – Corner style for offset kinks.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offset2**(*thisCurve*, *directionPoint*, *normal*, *distance*, *tolerance*, *angleTolerance*, *loose*, *cornerStyle*, *endStyle*, *multiple=false*)

Offsets this curve. If you have a nice offset, then there will be one entry in the array. If the original curve had kinks or the offset curve had self intersections, you will get multiple segments in the output array.

Arguments

- **directionPoint** (*rhino3dm.Point3d*) – A point that indicates the direction of the offset.
- **normal** (*rhino3dm.Vector3d*) – The normal to the offset plane.
- **distance** (*float*) – The positive or negative distance to offset.
- **tolerance** (*float*) – The offset or fitting tolerance.
- **angleTolerance** (*float*) – The angle tolerance, in radians, used to decide whether to split at kinks.
- **loose** (*bool*) – If false, offset within tolerance. If true, offset by moving edit points.
- **cornerStyle** (*CurveOffsetCornerStyle*) – Corner style for offset kinks.
- **endStyle** (*CurveOffsetEndStyle*) – End style for non-loose, non-closed curve offsets.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**ribbonOffset**(*thisCurve*, *distance*, *blendRadius*, *directionPoint*, *normal*, *tolerance*, *multiple=false*)

Offsets a closed curve in the following way: pProject the curve to a plane with given normal. Then, loose Offset the projection by distance + blend_radius and trim off self-intersection. THen, Offset the remaining curve back in the opposite direction by blend_radius, filling gaps with blends. Finally, use the elevations of the input curve to get the correct elevations of the result.

Arguments

- **distance** (*float*) – The positive distance to offset the curve.
- **blendRadius** (*float*) – Positive, typically the same as distance. When the offset results in a self-intersection that gets trimmed off at a kink, the kink will be blended out using this radius.
- **directionPoint** (*rhino3dm.Point3d*) – A point that indicates the direction of the offset. If the offset is inward, the point's projection to the plane should be well within the curve. It will be used to decide which part of the offset to keep if there are self-intersections.
- **normal** (*rhino3dm.Vector3d*) – A vector that indicates the normal of the plane in which the offset will occur.
- **tolerance** (*float*) – Used to determine self-intersections, not offset error.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The offset curve if successful.

Return type rhino3dm.Curve

RhinoCompute.Curve.**offsetOnSurface**(*thisCurve*, *face*, *distance*, *fittingTolerance*, *multiple=false*)

Offset this curve on a brep face surface. This curve must lie on the surface.

Arguments

- **face** (*rhino3dm.BrepFace*) – The brep face on which to offset.
- **distance** (*float*) – A distance to offset (+)left, (-)right.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetOnSurface1**(*thisCurve*, *face*, *throughPoint*, *fittingTolerance*, *multiple=false*)

Offset a curve on a brep face surface. This curve must lie on the surface. This overload allows to specify a surface point at which the offset will pass.

Arguments

- **face** (*rhino3dm.BrepFace*) – The brep face on which to offset.
- **throughPoint** (*rhino3dm.Point2d*) – 2d point on the brep face to offset through.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetOnSurface2**(*thisCurve*, *face*, *curveParameters*, *offsetDistances*, *fittingTolerance*, *multiple=false*)

Offset a curve on a brep face surface. This curve must lie on the surface. This overload allows to specify different offsets for different curve parameters.

Arguments

- **face** (*rhino3dm.BrepFace*) – The brep face on which to offset.
- **curveParameters** (*float[]*) – Curve parameters corresponding to the offset distances.
- **offsetDistances** (*float[]*) – distances to offset (+)left, (-)right.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetOnSurface3**(*thisCurve*, *surface*, *distance*, *fittingTolerance*, *multiple=false*)

Offset a curve on a surface. This curve must lie on the surface.

Arguments

- **surface** (*rhino3dm.Surface*) – A surface on which to offset.
- **distance** (*float*) – A distance to offset (+)left, (-)right.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetOnSurface4**(*thisCurve*, *surface*, *throughPoint*, *fittingTolerance*, *multiple=false*)

Offset a curve on a surface. This curve must lie on the surface. This overload allows to specify a surface point at which the offset will pass.

Arguments

- **surface** (*rhino3dm.Surface*) – A surface on which to offset.
- **throughPoint** (*rhino3dm.Point2d*) – 2d point on the brep face to offset through.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetOnSurface5**(*thisCurve*, *surface*, *curveParameters*, *offsetDistances*, *fittingTolerance*, *multiple=false*)

Offset this curve on a surface. This curve must lie on the surface. This overload allows to specify different offsets for different curve parameters.

Arguments

- **surface** (*rhino3dm.Surface*) – A surface on which to offset.
- **curveParameters** (*float[]*) – Curve parameters corresponding to the offset distances.
- **offsetDistances** (*float[]*) – Distances to offset (+)left, (-)right.
- **fittingTolerance** (*float*) – A fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curves on success, or None on failure.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**pullToBrepFace1**(*thisCurve*, *face*, *tolerance*, *multiple=false*)

Pulls this curve to a brep face and returns the result of that operation.

Arguments

- **face** (*rhino3dm.BrepFace*) – A brep face.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array containing the resulting curves after pulling. This array could be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Curve.**offsetNormalToSurface**(*thisCurve*, *surface*, *height*, *multiple=false*)

Finds a curve by offsetting an existing curve normal to a surface. The caller is responsible for ensuring that the curve lies on the input surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface from which normals are calculated.
- **height** (*float*) – offset distance (distance from surface to result curve)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Offset curve at distance height from the surface. The offset curve is interpolated through a small number of points so if the surface is irregular or complicated, the result will not be a very accurate offset.

Return type rhino3dm.Curve

CHAPTER 6

RhinoCompute.Extrusion

RhinoCompute.Extrusion.**getWireframe**(*thisExtrusion*, *multiple=false*)

Constructs all the Wireframe curves for this Extrusion.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Wireframe curves.

Return type rhino3dm.Curve[]

CHAPTER 7

RhinoCompute.GeometryBase

RhinoCompute.GeometryBase.**getBoundingBox**(*thisGeometryBase*, *accurate*, *multiple=false*)
Bounding box solver. Gets the world axis aligned bounding box for the geometry.

Arguments

- **accurate** (*bool*) – If true, a physically accurate bounding box will be computed. If not, a bounding box estimate will be computed. For some geometry types there is no difference between the estimate and the accurate bounding box. Estimated bounding boxes can be computed much (much) faster than accurate (or "tight") bounding boxes. Estimated bounding boxes are always similar to or larger than accurate bounding boxes.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The bounding box of the geometry in world coordinates or BoundingBox.Empty if not bounding box could be found.

Return type rhino3dm.BoundingBox

RhinoCompute.GeometryBase.**getBoundingBox1**(*thisGeometryBase*, *xform*, *multiple=false*)
Aligned Bounding box solver. Gets the world axis aligned bounding box for the transformed geometry.

Arguments

- **xform** (*Transform*) – Transformation to apply to object prior to the BoundingBox computation. The geometry itself is not modified.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The accurate bounding box of the transformed geometry in world coordinates or BoundingBox.Empty if not bounding box could be found.

Return type rhino3dm.BoundingBox

RhinoCompute.GeometryBase.**geometryEquals**(*first*, *second*, *multiple=false*)
Determines if two geometries equal one another, in pure geometrical shape. This version only compares the

geometry itself and does not include any user data comparisons. This is a comparison by value: for two identical items it will be true, no matter where in memory they may be stored.

Arguments

- **first** (*rhino3dm.GeometryBase*) – The first geometry
- **second** (*rhino3dm.GeometryBase*) – The second geometry
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The indication of equality

Return type bool

CHAPTER 8

RhinoCompute.Intersection

RhinoCompute.Intersection.**curvePlane**(*curve*, *plane*, *tolerance*, *multiple=false*)
Intersects a curve with an (infinite) plane.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to intersect.
- **plane** (*rhino3dm.Plane*) – Plane to intersect with.
- **tolerance** (*float*) – Tolerance to use during intersection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A list of intersection events or None if no intersections were recorded.

Return type CurveIntersections

RhinoCompute.Intersection.**meshPlane**(*mesh*, *plane*, *multiple=false*)
Intersects a mesh with an (infinite) plane.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to intersect.
- **plane** (*rhino3dm.Plane*) – Plane to intersect with.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines describing the intersection loops or None (Nothing in Visual Basic) if no intersections could be found.

Return type rhino3dm.Polyline[]

RhinoCompute.Intersection.**meshPlane1**(*mesh*, *planes*, *multiple=false*)
Intersects a mesh with a collection of (infinite) planes.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to intersect.

- **planes** (*list[rhino3dm.Plane]*) – Planes to intersect with.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines describing the intersection loops or None (Nothing in Visual Basic) if no intersections could be found.

Return type rhino3dm.Polyline[]

RhinoCompute.Intersection.**brepPlane**(*brep*, *plane*, *tolerance*, *multiple=false*)

Intersects a Brep with an (infinite) plane.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to intersect.
- **plane** (*rhino3dm.Plane*) – Plane to intersect with.
- **tolerance** (*float*) – Tolerance to use for intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Intersection.**curveSelf**(*curve*, *tolerance*, *multiple=false*)

Finds the places where a curve intersects itself.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for self-intersections.
- **tolerance** (*float*) – Intersection tolerance. If the curve approaches itself to within tolerance, an intersection is assumed.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveCurve**(*curveA*, *curveB*, *tolerance*, *overlapTolerance*, *multiple=false*)

Finds the intersections between two curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – First curve for intersection.
- **curveB** (*rhino3dm.Curve*) – Second curve for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curves approach each other to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveCurveValidate**(*curveA*, *curveB*, *tolerance*, *overlapTolerance*, *multiple=false*)

Finds the intersections between two curves.

Arguments

- **curveA** (*rhino3dm.Curve*) – First curve for intersection.
- **curveB** (*rhino3dm.Curve*) – Second curve for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curves approach each other to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveLine**(*curve*, *line*, *tolerance*, *overlapTolerance*, *multiple=false*)

Intersects a curve and an infinite line.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **line** (*Line*) – Infinite line to intersect.
- **tolerance** (*float*) – Intersection tolerance. If the curves approach each other to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveSurface**(*curve*, *surface*, *tolerance*, *overlapTolerance*, *multiple=false*)

Intersects a curve and a surface.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **surface** (*rhino3dm.Surface*) – Surface for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curve approaches the surface to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveSurfaceValidate**(*curve*, *surface*, *tolerance*, *overlapTolerance*, *multiple=false*)

Intersects a curve and a surface.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **surface** (*rhino3dm.Surface*) – Surface for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curve approaches the surface to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveSurface1**(*curve*, *curveDomain*, *surface*, *tolerance*, *overlapTolerance*, *multiple=false*)

Intersects a sub-curve and a surface.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **curveDomain** (*rhino3dm.Interval*) – Domain of sub-curve to take into consideration for Intersections.
- **surface** (*rhino3dm.Surface*) – Surface for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curve approaches the surface to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveSurfaceValidate1**(*curve*, *curveDomain*, *surface*, *tolerance*, *overlapTolerance*, *multiple=false*)

Intersects a sub-curve and a surface.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **curveDomain** (*rhino3dm.Interval*) – Domain of sub-curve to take into consideration for Intersections.
- **surface** (*rhino3dm.Surface*) – Surface for intersection.
- **tolerance** (*float*) – Intersection tolerance. If the curve approaches the surface to within tolerance, an intersection is assumed.
- **overlapTolerance** (*float*) – The tolerance with which the curves are tested.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A collection of intersection events.

Return type CurveIntersections

RhinoCompute.Intersection.**curveBrep**(*curve*, *brep*, *tolerance*, *multiple=false*)

Intersects a curve with a Brep. This function returns the 3D points of intersection and 3D overlap curves. If an error occurs while processing overlap curves, this function will return false, but it will still provide partial results.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve for intersection.
- **brep** (*rhino3dm.Brep*) – Brep for intersection.
- **tolerance** (*float*) – Fitting and near miss tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Intersection.**curveBrep1**(*curve*, *brep*, *tolerance*, *angleTolerance*, *multiple=false*)

Intersect a curve with a Brep. This function returns the intersection parameters on the curve.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve.
- **brep** (*rhino3dm.Brep*) – Brep.
- **tolerance** (*float*) – Absolute tolerance for intersections.
- **angleTolerance** (*float*) – Angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Intersection.**curveBrepFace**(*curve*, *face*, *tolerance*, *multiple=false*)

Intersects a curve with a Brep face.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve.
- **face** (*rhino3dm.BrepFace*) – A brep face.
- **tolerance** (*float*) – Fitting and near miss tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Intersection.**surfaceSurface**(*surfaceA*, *surfaceB*, *tolerance*, *multiple=false*)

Intersects two Surfaces.

Arguments

- **surfaceA** (*rhino3dm.Surface*) – First Surface for intersection.
- **surfaceB** (*rhino3dm.Surface*) – Second Surface for intersection.
- **tolerance** (*float*) – Intersection tolerance.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Intersection.**brepBrep**(*brepA*, *brepB*, *tolerance*, *multiple=false*)
Intersects two Breps.

Arguments

- **brepA** (*rhino3dm.Brep*) – First Brep for intersection.
- **brepB** (*rhino3dm.Brep*) – Second Brep for intersection.
- **tolerance** (*float*) – Intersection tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success; False on failure.

Return type bool

RhinoCompute.Intersection.**brepSurface**(*brep*, *surface*, *tolerance*, *multiple=false*)
Intersects a Brep and a Surface.

Arguments

- **brep** (*rhino3dm.Brep*) – A brep to be intersected.
- **surface** (*rhino3dm.Surface*) – A surface to be intersected.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success; False on failure.

Return type bool

RhinoCompute.Intersection.**meshMeshFast**(*meshA*, *meshB*, *multiple=false*)
This is an old overload kept for compatibility. Overlaps and near misses are ignored.

Arguments

- **meshA** (*rhino3dm.Mesh*) – First mesh for intersection.
- **meshB** (*rhino3dm.Mesh*) – Second mesh for intersection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of intersection line segments, or None if no intersections were found.

Return type Line[]

RhinoCompute.Intersection.**meshMeshAccurate**(*meshA*, *meshB*, *tolerance*, *multiple=false*)
Intersects two meshes. Overlaps and near misses are handled. This is an old method kept for compatibility.

Arguments

- **meshA** (*rhino3dm.Mesh*) – First mesh for intersection.
- **meshB** (*rhino3dm.Mesh*) – Second mesh for intersection.

- **tolerance** (*float*) – A tolerance value. If negative, the positive value will be used. WARNING! Good tolerance values are in the magnitude of 10^-7, or RhinoMath.SqrtEpsilon*10.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of intersection and overlaps polylines.

Return type rhino3dm.Polyline[]

RhinoCompute.Intersection.**meshRay**(*mesh*, *ray*, *multiple=false*)

Finds the first intersection of a ray with a mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect.
- **ray** (*Ray3d*) – A ray to be casted.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns >= 0.0 parameter along ray if successful. < 0.0 if no intersection found.

Return type float

RhinoCompute.Intersection.**meshRay1**(*mesh*, *ray*, *multiple=false*)

Finds the first intersection of a ray with a mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect.
- **ray** (*Ray3d*) – A ray to be casted.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns >= 0.0 parameter along ray if successful. < 0.0 if no intersection found.

Return type float

RhinoCompute.Intersection.**meshPolyline**(*mesh*, *curve*, *multiple=false*)

Finds the intersection of a mesh and a polyline. Starting from version 7, points are always sorted along the polyline.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect.
- **curve** (*PolylineCurve*) – A polyline curves to intersect.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each face that was passed by the faceIds out reference.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**meshPolylineSorted**(*mesh*, *curve*, *multiple=false*)

Finds the intersection of a mesh and a polyline. Points are guaranteed to be sorted along the polyline.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect.
- **curve** (*PolylineCurve*) – A polyline curves to intersect.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each face that was passed by the faceIds out reference.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**meshLine**(*mesh*, *line*, *multiple=false*)
Finds the intersections of a mesh and a line. The points are not necessarily sorted.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect
- **line** (*Line*) – The line to intersect with the mesh
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each face that was passed by the faceIds out reference. Empty if no items are found.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**meshLine1**(*mesh*, *line*, *multiple=false*)
Finds the intersections of a mesh and a line.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect
- **line** (*Line*) – The line to intersect with the mesh
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each face that was passed by the faceIds out reference. Empty if no items are found.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**meshLineSorted**(*mesh*, *line*, *multiple=false*)
Finds the intersections of a mesh and a line. Points are sorted along the line.

Arguments

- **mesh** (*rhino3dm.Mesh*) – A mesh to intersect
- **line** (*Line*) – The line to intersect with the mesh
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each face that was passed by the faceIds out reference. Empty if no items are found.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**rayShoot**(*ray*, *geometry*, *maxReflections*, *multiple=false*)
Computes point intersections that occur when shooting a ray to a collection of surfaces and Breps.

Arguments

- **ray** (*Ray3d*) – A ray used in intersection.
- **geometry** (*list[rhino3dm.GeometryBase]*) – Only Surface and Brep objects are currently supported. Trims are ignored on Breps.

- **maxReflections** (*int*) – The maximum number of reflections. This value should be any value between 1 and 1000, inclusive.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points: one for each surface or Brep face that was hit, or an empty array on failure.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**rayShoot1**(*geometry*, *ray*, *maxReflections*, *multiple=false*)

Computes point intersections that occur when shooting a ray to a collection of surfaces and Breps.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – The collection of surfaces and Breps to intersect. Trims are ignored on Breps.
- **ray** (*Ray3d*) – >A ray used in intersection.
- **maxReflections** (*int*) – The maximum number of reflections. This value should be any value between 1 and 1000, inclusive.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of RayShootEvent structs if successful, or an empty array on failure.

Return type RayShootEvent[]

RhinoCompute.Intersection.**projectPointsToMeshes**(*meshes*, *points*, *direction*, *tolerance*, *multiple=false*)

Projects points onto meshes.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – the meshes to project on to.
- **points** (*list[rhino3dm.Point3d]*) – the points to project.
- **direction** (*rhino3dm.Vector3d*) – the direction to project.
- **tolerance** (*float*) – Projection tolerances used for culling close points and for line-mesh intersection.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of projected points, or None in case of any error or invalid input.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**projectPointsToMeshesEx**(*meshes*, *points*, *direction*, *tolerance*, *multiple=false*)

Projects points onto meshes.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – the meshes to project on to.
- **points** (*list[rhino3dm.Point3d]*) – the points to project.
- **direction** (*rhino3dm.Vector3d*) – the direction to project.
- **tolerance** (*float*) – Projection tolerances used for culling close points and for line-mesh intersection.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of projected points, or None in case of any error or invalid input.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**projectPointsToBreps**(*breps*, *points*, *direction*, *tolerance*, *multiple=false*)

Projects points onto breps.

Arguments

- **breps** (*list[rhino3dm.Brep]*) – The breps projection targets.
- **points** (*list[rhino3dm.Point3d]*) – The points to project.
- **direction** (*rhino3dm.Vector3d*) – The direction to project.
- **tolerance** (*float*) – The tolerance used for intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of projected points, or None in case of any error or invalid input.

Return type rhino3dm.Point3d[]

RhinoCompute.Intersection.**projectPointsToBrepsEx**(*breps*, *points*, *direction*, *tolerance*, *multiple=false*)

Projects points onto breps.

Arguments

- **breps** (*list[rhino3dm.Brep]*) – The breps projection targets.
- **points** (*list[rhino3dm.Point3d]*) – The points to project.
- **direction** (*rhino3dm.Vector3d*) – The direction to project.
- **tolerance** (*float*) – The tolerance used for intersections.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of projected points, or None in case of any error or invalid input.

Return type rhino3dm.Point3d[]

CHAPTER 9

RhinoCompute.Mesh

RhinoCompute.Mesh.**createFromPlane**(*plane*, *xInterval*, *yInterval*, *xCount*, *yCount*, *multiple=false*)

Constructs a planar mesh grid.

Arguments

- **plane** (*rhino3dm.Plane*) – Plane of mesh.
- **xInterval** (*rhino3dm.Interval*) – Interval describing size and extends of mesh along plane x-direction.
- **yInterval** (*rhino3dm.Interval*) – Interval describing size and extends of mesh along plane y-direction.
- **xCount** (*int*) – Number of faces in x-direction.
- **yCount** (*int*) – Number of faces in y-direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromFilteredFaceList**(*original*, *inclusion*, *multiple=false*)

Constructs a sub-mesh, that contains a filtered list of faces.

Arguments

- **original** (*rhino3dm.Mesh*) – The mesh to copy. This item can be null, and in this case an empty mesh is returned.
- **inclusion** (*IEnumerable<bool>*) – A series of True and False values, that determine if each face is used in the new mesh. If the amount does not match the length of the face list, the pattern is repeated. If it exceeds the amount of faces in the mesh face list, the pattern is truncated. This items can be None or empty, and the mesh will simply be duplicated.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromBox**(*box*, *xCount*, *yCount*, *zCount*, *multiple=false*)
Constructs new mesh that matches a bounding box.

Arguments

- **box** (*rhino3dm.BoundingBox*) – A box to use for creation.
- **xCount** (*int*) – Number of faces in x-direction.
- **yCount** (*int*) – Number of faces in y-direction.
- **zCount** (*int*) – Number of faces in z-direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new brep, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromBox1**(*box*, *xCount*, *yCount*, *zCount*, *multiple=false*)
Constructs new mesh that matches an aligned box.

Arguments

- **box** (*rhino3dm.Box*) – Box to match.
- **xCount** (*int*) – Number of faces in x-direction.
- **yCount** (*int*) – Number of faces in y-direction.
- **zCount** (*int*) – Number of faces in z-direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromBox2**(*corners*, *xCount*, *yCount*, *zCount*, *multiple=false*)
Constructs new mesh from 8 corner points.

Arguments

- **corners** (*list[rhino3dm.Point3d]*) – 8 points defining the box corners arranged as the vN labels indicate. v7_____v6| |\| \ | \| \ _____\| v4 | v5| | | || | | |v3–|———-v2 | | | | | | v0_____v1
- **xCount** (*int*) – Number of faces in x-direction.
- **yCount** (*int*) – Number of faces in y-direction.
- **zCount** (*int*) – Number of faces in z-direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new brep, or None on failure. A new box mesh, on None on error.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromSphere**(*sphere*, *xCount*, *yCount*, *multiple=false*)
Constructs a mesh sphere.

Arguments

- **sphere** (*rhino3dm.Sphere*) – Base sphere for mesh.
- **xCount** (*int*) – Number of faces in the around direction.

- **yCount** (*int*) – Number of faces in the top-to-bottom direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createIcoSphere**(*sphere*, *subdivisions*, *multiple=false*)

Constructs a icospherical mesh. A mesh icosphere differs from a standard UV mesh sphere in that it's vertices are evenly distributed. A mesh icosphere starts from an icosahedron (a regular polyhedron with 20 equilateral triangles). It is then refined by splitting each triangle into 4 smaller triangles. This splitting can be done several times.

Arguments

- **sphere** (*rhino3dm.Sphere*) – The input sphere provides the orienting plane and radius.
- **subdivisions** (*int*) – The number of times you want the faces split, where 0 <= subdivisions <= 7. Note, the total number of mesh faces produces is: 20 * (4 ^ subdivisions)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A welded mesh icosphere if successful, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createQuadSphere**(*sphere*, *subdivisions*, *multiple=false*)

Constructs a quad mesh sphere. A quad mesh sphere differs from a standard UV mesh sphere in that it's vertices are evenly distributed. A quad mesh sphere starts from a cube (a regular polyhedron with 6 square sides). It is then refined by splitting each quad into 4 smaller quads. This splitting can be done several times.

Arguments

- **sphere** (*rhino3dm.Sphere*) – The input sphere provides the orienting plane and radius.
- **subdivisions** (*int*) – The number of times you want the faces split, where 0 <= subdivisions <= 8. Note, the total number of mesh faces produces is: 6 * (4 ^ subdivisions)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A welded quad mesh sphere if successful, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCylinder**(*cylinder*, *vertical*, *around*, *multiple=false*)

Constructs a capped mesh cylinder.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cylinder.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a mesh cylinder if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCylinder1**(*cylinder*, *vertical*, *around*, *capBottom*, *capTop*, *multiple=false*)

Constructs a mesh cylinder.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cylinder.
- **capBottom** (*bool*) – If True end at Cylinder.Height1 should be capped.
- **capTop** (*bool*) – If True end at Cylinder.Height2 should be capped.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a mesh cylinder if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCylinder2**(*cylinder*, *vertical*, *around*, *capBottom*, *capTop*, *quadCaps*, *multiple=false*)

Constructs a mesh cylinder.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cylinder.
- **capBottom** (*bool*) – If True end at Cylinder.Height1 should be capped.
- **capTop** (*bool*) – If True end at Cylinder.Height2 should be capped.
- **quadCaps** (*bool*) – If True and it's possible to make quad caps, i.e.. around is even, then caps will have quad faces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a mesh cylinder if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCylinder3**(*cylinder*, *vertical*, *around*, *capBottom*, *capTop*, *circumscribe*, *quadCaps*, *multiple=false*)

Constructs a mesh cylinder.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cylinder.
- **capBottom** (*bool*) – If True end at Cylinder.Height1 should be capped.
- **capTop** (*bool*) – If True end at Cylinder.Height2 should be capped.
- **circumscribe** (*bool*) – If True end polygons will circumscribe circle.
- **quadCaps** (*bool*) – If True and it's possible to make quad caps, i.e.. around is even, then caps will have quad faces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a mesh cylinder if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCone**(*cone*, *vertical*, *around*, *multiple=false*)
Constructs a solid mesh cone.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cone.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A valid mesh if successful.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCone1**(*cone*, *vertical*, *around*, *solid*, *multiple=false*)
Constructs a mesh cone.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cone.
- **solid** (*bool*) – If False the mesh will be open with no faces on the circular planar portion.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A valid mesh if successful.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCone2**(*cone*, *vertical*, *around*, *solid*, *quadCaps*, *multiple=false*)
Constructs a mesh cone.

Arguments

- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the cone.
- **solid** (*bool*) – If False the mesh will be open with no faces on the circular planar portion.
- **quadCaps** (*bool*) – If True and it's possible to make quad caps, i.e.. around is even, then caps will have quad faces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A valid mesh if successful.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromTorus**(*torus*, *vertical*, *around*, *multiple=false*)
Constructs a mesh torus.

Arguments

- **torus** (*Torus*) – The torus.
- **vertical** (*int*) – Number of faces in the top-to-bottom direction.
- **around** (*int*) – Number of faces around the torus.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Returns a mesh torus if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromPlanarBoundary**(*boundary*, *parameters*, *multiple=false*)

Do not use this overload. Use version that takes a tolerance parameter instead.

Arguments

- **boundary** (*rhino3dm.Curve*) – Do not use.
- **parameters** (*rhino3dm.MeshingParameters*) – Do not use.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Do not use.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromPlanarBoundary1**(*boundary*, *parameters*, *tolerance*, *multiple=false*)

Attempts to construct a mesh from a closed planar curve.RhinoMakePlanarMeshes

Arguments

- **boundary** (*rhino3dm.Curve*) – must be a closed planar curve.
- **parameters** (*rhino3dm.MeshingParameters*) – parameters used for creating the mesh.
- **tolerance** (*float*) – Tolerance to use during operation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New mesh on success or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromClosedPolyline**(*polyline*, *multiple=false*)

Attempts to create a Mesh that is a triangulation of a simple closed polyline that projects onto a plane.

Arguments

- **polyline** (*rhino3dm.Polyline*) – must be closed
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New mesh on success or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromTessellation**(*points*, *edges*, *plane*, *allowNewVertices*, *multiple=false*)

Attempts to create a mesh that is a triangulation of a list of points, projected on a plane, including its holes and fixed edges.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – A list, an array or any enumerable of points.
- **plane** (*rhino3dm.Plane*) – A plane.

- **allowNewVertices** (*bool*) – If true, the mesh might have more vertices than the list of input points, if doing so will improve long thin triangles.
- **edges** (*IEnumerable<IEnumerable<Point3d>>*) – A list of polylines, or other lists of points representing edges. This can be null. If nested enumerable items are null, they will be discarded.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh, or None if not successful.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromBrep**(*brep*, *multiple=false*)

Constructs a mesh from a brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to approximate.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of meshes.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createFromBrep1**(*brep*, *meshingParameters*, *multiple=false*)

Constructs a mesh from a brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to approximate.
- **meshingParameters** (*rhino3dm.MeshingParameters*) – Parameters to use during meshing.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of meshes.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createFromSurface**(*surface*, *multiple=false*)

Constructs a mesh from a surface

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to approximate
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New mesh representing the surface

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromSurface1**(*surface*, *meshingParameters*, *multiple=false*)

Constructs a mesh from a surface

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to approximate

- **meshingParameters** (*rhino3dm.MeshingParameters*) – settings used to create the mesh
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New mesh representing the surface

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromSubD**(*subd*, *displayDensity*, *multiple=false*)

Create a mesh from a SubD limit surface

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createPatch**(*outerBoundary*, *angleToleranceRadians*, *pullbackSurface*, *innerBoundaryCurves*, *innerBothSideCurves*, *innerPoints*, *trimback*, *divisions*, *multiple=false*)

Construct a mesh patch from a variety of input geometry.

Arguments

- **outerBoundary** (*rhino3dm.Polyline*) – (optional: can be null) Outer boundary polyline, if provided this will become the outer boundary of the resulting mesh. Any of the input that is completely outside the outer boundary will be ignored and have no impact on the result. If any of the input intersects the outer boundary the result will be unpredictable and is likely to not include the entire outer boundary.
- **angleToleranceRadians** (*float*) – Maximum angle between unit tangents and adjacent vertices. Used to divide curve inputs that cannot otherwise be represented as a polyline.
- **innerBoundaryCurves** (*list[rhino3dm.Curve]*) – (optional: can be null) Polylines to create holes in the output mesh. If innerBoundaryCurves are the only input then the result may be null if trimback is set to False (see comments for trimback) because the resulting mesh could be invalid (all faces created contained vertexes from the perimeter boundary).
- **pullbackSurface** (*rhino3dm.Surface*) – (optional: can be null) Initial surface where 3d input will be pulled to make a 2d representation used by the function that generates the mesh. Providing a pullbackSurface can be helpful when it is similar in shape to the pattern of the input, the pulled 2d points will be a better representation of the 3d points. If all of the input is more or less coplanar to start with, providing pullbackSurface has no real benefit.
- **innerBothSideCurves** (*list[rhino3dm.Curve]*) – (optional: can be null) These polylines will create faces on both sides of the edge. If there are only input points(innerPoints) there is no way to guarantee a triangulation that will create an edge between two particular points. Adding a line, or polyline, to innerBothsideCurves that includes points from innerPoints will help guide the triangulation.
- **innerPoints** (*list[rhino3dm.Point3d]*) – (optional: can be null) Points to be used to generate the mesh. If outerBoundary is not null, points outside of that boundary after it has been pulled to pullbackSurface (or the best plane through the input if pullbackSurface is null) will be ignored.
- **trimback** (*bool*) – Only used when a outerBoundary has not been provided. When that is the case, the function uses the perimeter of the surface as the outer boundary instead.

If true, any face of the resulting triangulated mesh that contains a vertex of the perimeter boundary will be removed.

- **divisions** (*int*) – Only used when a outerBoundary has not been provided. When that is the case, division becomes the number of divisions each side of the surface's perimeter will be divided into to create an outer boundary to work with.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns mesh on success; None on failure

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createBooleanUnion**(*meshes*, *multiple=false*)

Computes the solid union of a set of meshes.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to union.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Mesh results or None on failure.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createBooleanDifference**(*firstSet*, *secondSet*, *multiple=false*)

Computes the solid difference of two sets of Meshes.

Arguments

- **firstSet** (*list[rhino3dm.Mesh]*) – First set of Meshes (the set to subtract from).
- **secondSet** (*list[rhino3dm.Mesh]*) – Second set of Meshes (the set to subtract).
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Mesh results or None on failure.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createBooleanIntersection**(*firstSet*, *secondSet*, *multiple=false*)

Computes the solid intersection of two sets of meshes.

Arguments

- **firstSet** (*list[rhino3dm.Mesh]*) – First set of Meshes.
- **secondSet** (*list[rhino3dm.Mesh]*) – Second set of Meshes.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of Mesh results or None on failure.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createBooleanSplit**(*meshesToSplit*, *meshSplitters*, *multiple=false*)

Splits a set of meshes with another set.

Arguments

- **meshesToSplit** (*list[rhino3dm.Mesh]*) – A list, an array, or any enumerable set of meshes to be split. If this is null, None will be returned.

- **meshSplitters** (*list[rhino3dm.Mesh]*) – A list, an array, or any enumerable set of meshes that cut. If this is null, None will be returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh array, or None on error.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**createFromCurvePipe**(*curve*, *radius*, *segments*, *accuracy*, *capType*, *faceted*, *intervals*, *multiple=false*)

Constructs a new mesh pipe from a curve.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve to pipe.
- **radius** (*float*) – The radius of the pipe.
- **segments** (*int*) – The number of segments in the pipe.
- **accuracy** (*int*) – The accuracy of the pipe.
- **capType** (*MeshPipeCapStyle*) – The type of cap to be created at the end of the pipe.
- **faceted** (*bool*) – Specifies whether the pipe is faceted, or not.
- **intervals** (*list[rhino3dm.Interval]*) – A series of intervals to pipe. This value can be null.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromCurveExtrusion**(*curve*, *direction*, *parameters*, *boundingBox*, *multiple=false*)

Constructs a new extrusion from a curve.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve to extrude.
- **direction** (*rhino3dm.Vector3d*) – The direction of extrusion.
- **parameters** (*rhino3dm.MeshingParameters*) – The parameters of meshing.
- **boundingBox** (*rhino3dm.BoundingBox*) – The bounding box controls the length of the extrusion.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createFromIterativeCleanup**(*meshes*, *tolerance*, *multiple=false*)

Repairs meshes with vertices that are too near, using a tolerance value.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – The meshes to be repaired.
- **tolerance** (*float*) – A minimum distance for clean vertices.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A valid meshes array if successful. If no change was required, some meshes can be null. Otherwise, null, when no changes were done.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**requireIterativeCleanup**(*meshes*, *tolerance*, *multiple=false*)

Analyzes some meshes, and determines if a pass of CreateFromIterativeCleanup would change the array. All available cleanup steps are used. Currently available cleanup steps are:- mending of single precision coincidence even though double precision vertices differ.- union of nearly identical vertices, irrespectively of their origin.- removal of t-joints along edges.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – A list, and array or any enumerable of meshes.
- **tolerance** (*float*) – A 3d distance. This is usually a value of about 10e-7 magnitude.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if meshes would be changed, otherwise false.

Return type bool

RhinoCompute.Mesh.**volume**(*thisMesh*, *multiple=false*)

Compute volume of the mesh.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Volume of the mesh.

Return type float

RhinoCompute.Mesh.**isPointInside**(*thisMesh*, *point*, *tolerance*, *strictlyIn*, *multiple=false*)

Determines if a point is inside a solid mesh.

Arguments

- **point** (*rhino3dm.Point3d*) – 3d point to test.
- **tolerance** (*float*) – (>=0) 3d distance tolerance used for ray-mesh intersection and determining strict inclusion. This is expected to be a tiny value.
- **strictlyIn** (*bool*) – If strictlyIn is true, then point must be inside mesh by at least tolerance in order for this function to return true. If strictlyIn is false, then this function will return True if point is inside or the distance from point to a mesh face is <= tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if point is inside the solid mesh, False if not.

Return type bool

RhinoCompute.Mesh.**smooth**(*thisMesh*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *multiple=false*)

Smooths a mesh by averaging the positions of mesh vertices in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much vertices move towards the average of the neighboring vertices.
- **bXSmooth** (*bool*) – When True vertices move in X axis direction.
- **bYSmooth** (*bool*) – When True vertices move in Y axis direction.
- **bZSmooth** (*bool*) – When True vertices move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True vertices along naked edges will not be modified.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**smooth1**(*thisMesh*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *plane*, *multiple=false*)

Smooths a mesh by averaging the positions of mesh vertices in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much vertices move towards the average of the neighboring vertices.
- **bXSmooth** (*bool*) – When True vertices move in X axis direction.
- **bYSmooth** (*bool*) – When True vertices move in Y axis direction.
- **bZSmooth** (*bool*) – When True vertices move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True vertices along naked edges will not be modified.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **plane** (*rhino3dm.Plane*) – If SmoothingCoordinateSystem.CPlane specified, then the construction plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**smooth2**(*thisMesh*, *vertexIndices*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *plane*, *multiple=false*)

Smooths part of a mesh by averaging the positions of mesh vertices in a specified region.

Arguments

- **vertexIndices** (*list[int]*) – The mesh vertex indices that specify the part of the mesh to smooth.
- **smoothFactor** (*float*) – The smoothing factor, which controls how much vertices move towards the average of the neighboring vertices.
- **bXSmooth** (*bool*) – When True vertices move in X axis direction.
- **bYSmooth** (*bool*) – When True vertices move in Y axis direction.
- **bZSmooth** (*bool*) – When True vertices move in Z axis direction.

- **bFixBoundaries** (*bool*) – When True vertices along naked edges will not be modified.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **plane** (*rhino3dm.Plane*) – If SmoothingCoordinateSystem.CPlane specified, then the construction plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**unweld**(*thisMesh*, *angleToleranceRadians*, *modifyNormals*, *multiple=false*)

Makes sure that faces sharing an edge and having a difference of normal greater than or equal to angleToleranceRadians have unique vertexes along that edge, adding vertices if necessary.

Arguments

- **angleToleranceRadians** (*float*) – Angle at which to make unique vertices.
- **modifyNormals** (*bool*) – Determines whether new vertex normals will have the same vertex normal as the original (false) or vertex normals made from the corresponding face normals (true)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Mesh.**unweldEdge**(*thisMesh*, *edgeIndices*, *modifyNormals*, *multiple=false*)

Adds creases to a smooth mesh by creating coincident vertices along selected edges.

Arguments

- **edgeIndices** (*list[int]*) – An array of mesh topology edge indices.
- **modifyNormals** (*bool*) – If true, the vertex normals on each side of the edge take the same value as the face to which they belong, giving the mesh a hard edge look. If false, each of the vertex normals on either side of the edge is assigned the same value as the original normal that the pair is replacing, keeping a smooth look.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**unweldVertices**(*thisMesh*, *topologyVertexIndices*, *modifyNormals*, *multiple=false*)

Ensures that faces sharing a common topological vertex have unique indices into the collection.

Arguments

- **topologyVertexIndices** (*list[int]*) – Topological vertex indices, from the collection, to be unwelded. Use to convert from vertex indices to topological vertex indices.
- **modifyNormals** (*bool*) – If true, the new vertex normals will be calculated from the face normal.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**weld**(*thisMesh*, *angleToleranceRadians*, *multiple=false*)
Makes sure that faces sharing an edge and having a difference of normal greater than or equal to angleToleranceRadians share vertexes along that edge, vertex normals are averaged.

Arguments

- **angleToleranceRadians** (*float*) – Angle at which to weld vertices.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Mesh.**rebuildNormals**(*thisMesh*, *multiple=false*)
Removes mesh normals and reconstructs the face and vertex normals based on the orientation of the faces.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Mesh.**extractNonManifoldEdges**(*thisMesh*, *selective*, *multiple=false*)
Extracts, or removes, non-manifold mesh edges.

Arguments

- **selective** (*bool*) – If true, then extract hanging faces only.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A mesh containing the extracted non-manifold parts if successful, None otherwise.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**healNakedEdges**(*thisMesh*, *distance*, *multiple=false*)
Attempts to "heal" naked edges in a mesh based on a given distance. First attempts to move vertexes to neighboring vertexes that are within that distance away. Then it finds edges that have a closest point to the vertex within the distance and splits the edge. When it finds one it splits the edge and makes two new edges using that point.

Arguments

- **distance** (*float*) – Distance to not exceed when modifying the mesh.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**fillHoles**(*thisMesh*, *multiple=false*)
Attempts to determine "holes" in the mesh by chaining naked edges together. Then it triangulates the closed polygons adds the faces to the mesh.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**fileHole**(*thisMesh*, *topologyEdgeIndex*, *multiple=false*)

Given a starting "naked" edge index, this function attempts to determine a "hole" by chaining additional naked edges together until if returns to the start index. Then it triangulates the closed polygon and either adds the faces to the mesh.

Arguments

- **topologyEdgeIndex** (*int*) – Starting naked edge index.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.Mesh.**matchEdges**(*thisMesh*, *distance*, *rachet*, *multiple=false*)

Moves face edges of an open mesh to meet adjacent face edges. The method will first try to match vertices, and then then it will try to split edges to make the edges match.

Arguments

- **distance** (*float*) – The distance tolerance. Use larger tolerances only if you select specific edges to close.
- **rachet** (*bool*) – If true, matching the mesh takes place in four passes starting at a tolerance that is smaller than your specified tolerance and working up to the specified tolerance with successive passes. This matches small edges first and works up to larger edges. If false, then a single pass is made.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True of edges were matched, False otherwise.

Return type bool

RhinoCompute.Mesh.**unifyNormals**(*thisMesh*, *multiple=false*)

Attempts to fix inconsistencies in the directions of mesh faces in a mesh. This function does not modify mesh vertex normals, it rearranges the mesh face winding and face normals to make them all consistent. Note, you may want to call Mesh.Normals.ComputeNormals() to recompute vertex normals after calling this functions.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns number of faces that were modified.

Return type int

RhinoCompute.Mesh.**unifyNormals1**(*thisMesh*, *countOnly*, *multiple=false*)

Attempts to fix inconsistencies in the directions of mesh faces in a mesh. This function does not modify mesh vertex normals, it rearranges the mesh face winding and face normals to make them all consistent. Note, you may want to call Mesh.Normals.ComputeNormals() to recompute vertex normals after calling this functions.

Arguments

- **countOnly** (*bool*) – If true, then only the number of faces that would be modified is determined.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns If countOnly=false, the number of faces that were modified. If countOnly=true, the number of faces that would be modified.

Return type int

RhinoCompute.Mesh.**mergeAllCoplanarFaces**(*thisMesh*, *tolerance*, *multiple=false*)

Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.Mesh.**mergeAllCoplanarFaces1**(*thisMesh*, *tolerance*, *angleTolerance*, *multiple=false*)

Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **angleTolerance** (*float*) – Angle tolerance, in radians, for determining when faces are parallel. When in doubt, use the document's ModelAngleToleranceRadians property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.Mesh.**splitDisjointPieces**(*thisMesh*, *multiple=false*)

Splits up the mesh into its unconnected pieces.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array containing all the disjoint pieces that make up this Mesh.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**split**(*thisMesh*, *plane*, *multiple=false*)

Split a mesh by an infinite plane.

Arguments

- **plane** (*rhino3dm.Plane*) – The splitting plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh array with the split result. This can be None if no result was found.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**split1**(*thisMesh*, *mesh*, *multiple=false*)

Split a mesh with another mesh. Suggestion: upgrade to overload with tolerance.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to split with.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of mesh segments representing the split result.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**split2**(*thisMesh*, *meshes*, *multiple=false*)

Split a mesh with a collection of meshes. Suggestion: upgrade to overload with tolerance. Does not split at coplanar intersections.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to split with.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of mesh segments representing the split result.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**split3**(*thisMesh*, *meshes*, *tolerance*, *splitAtCoplanar*, *textLog*, *cancel*, *progress*, *multiple=false*)

Split a mesh with a collection of meshes.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to split with.
- **tolerance** (*float*) – A value for intersection tolerance. WARNING! Correct values are typically in the (10e-8 - 10e-4) range.An option is to use the document tolerance diminished by a few orders or magnitude.
- **splitAtCoplanar** (*bool*) – If false, coplanar areas will not be separated.
- **textLog** (*TextLog*) – A text log to write onto.
- **cancel** (*CancellationToken*) – A cancellation token.
- **progress** (*IProgress<double>*) – A progress reporter item. This can be null.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of mesh parts representing the split result, or null: when no mesh intersected, or if a cancel stopped the computation.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**split4**(*thisMesh*, *meshes*, *tolerance*, *splitAtCoplanar*, *createNgons*, *textLog*, *cancel*, *progress*, *multiple=false*)

Split a mesh with a collection of meshes.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to split with.

- **tolerance** (*float*) – A value for intersection tolerance. WARNING! Correct values are typically in the (10e-8 - 10e-4) range.An option is to use the document tolerance diminished by a few orders or magnitude.
- **splitAtCoplanar** (*bool*) – If false, coplanar areas will not be separated.
- **createNgons** (*bool*) – If true, creates ngons along the split ridge.
- **textLog** (*TextLog*) – A text log to write onto.
- **cancel** (*CancellationToken*) – A cancellation token.
- **progress** (*IProgress<double>*) – A progress reporter item. This can be null.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of mesh parts representing the split result, or null: when no mesh intersected, or if a cancel stopped the computation.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**getOutlines**(*thisMesh*, *plane*, *multiple=false*)

Constructs the outlines of a mesh projected against a plane.

Arguments

- **plane** (*rhino3dm.Plane*) – A plane to project against.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines, or None on error.

Return type rhino3dm.Polyline[]

RhinoCompute.Mesh.**getOutlines1**(*thisMesh*, *viewport*, *multiple=false*)

Constructs the outlines of a mesh. The projection information in the viewport is used to determine how the outlines are projected.

Arguments

- **viewport** (*Display.RhinoViewport*) – A viewport to determine projection direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines, or None on error.

Return type rhino3dm.Polyline[]

RhinoCompute.Mesh.**getOutlines2**(*thisMesh*, *viewportInfo*, *plane*, *multiple=false*)

Constructs the outlines of a mesh.

Arguments

- **viewportInfo** (*ViewportInfo*) – The viewport info that provides the outline direction.
- **plane** (*rhino3dm.Plane*) – Usually the view's construction plane. If a parallel projection and view plane is parallel to this, then project the results to the plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines, or None on error.

Return type rhino3dm.Polyline[]

RhinoCompute.Mesh.**getNakedEdges**(*thisMesh*, *multiple=false*)

Returns all edges of a mesh that are considered "naked" in the sense that the edge only has one face.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of polylines, or None on error.

Return type rhino3dm.Polyline[]

RhinoCompute.Mesh.**explodeAtUnweldedEdges**(*thisMesh*, *multiple=false*)

Explode the mesh into sub-meshes where a sub-mesh is a collection of faces that are contained within a closed loop of "unwelded" edges. Unwelded edges are edges where the faces that share the edge have unique mesh vertexes (not mesh topology vertexes) at both ends of the edge.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of sub-meshes on success; None on error. If the count in the returned array is 1, then nothing happened and the output is essentially a copy of the input.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**closestPoint**(*thisMesh*, *testPoint*, *multiple=false*)

Gets the point on the mesh that is closest to a given test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to search for.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The point on the mesh closest to testPoint, or Point3d.Unset on failure.

Return type rhino3dm.Point3d

RhinoCompute.Mesh.**closestMeshPoint**(*thisMesh*, *testPoint*, *maximumDistance*, *multiple=false*)

Gets the point on the mesh that is closest to a given test point. Similar to the ClosestPoint function except this returns a MeshPoint class which includes extra information beyond just the location of the closest point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – The source of the search.
- **maximumDistance** (*float*) – Optional upper bound on the distance from test point to the mesh. If you are only interested in finding a point Q on the mesh when testPoint.DistanceTo(Q) < maximumDistance, then set maximumDistance to that value. This parameter is ignored if you pass 0.0 for a maximumDistance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns closest point information on success. None on failure.

Return type MeshPoint

RhinoCompute.Mesh.**closestPoint1**(*thisMesh*, *testPoint*, *maximumDistance*, *multiple=false*)

Gets the point on the mesh that is closest to a given test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to search for.
- **maximumDistance** (*float*) – Optional upper bound on the distance from test point to the mesh. If you are only interested in finding a point Q on the mesh when test-Point.DistanceTo(Q) < maximumDistance, then set maximumDistance to that value. This parameter is ignored if you pass 0.0 for a maximumDistance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Index of face that the closest point lies on if successful. -1 if not successful; the value of pointOnMesh is undefined.

Return type int

RhinoCompute.Mesh.**closestPoint2**(*thisMesh*, *testPoint*, *maximumDistance*, *multiple=false*)

Gets the point on the mesh that is closest to a given test point.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – Point to search for.
- **maximumDistance** (*float*) – Optional upper bound on the distance from test point to the mesh. If you are only interested in finding a point Q on the mesh when test-Point.DistanceTo(Q) < maximumDistance, then set maximumDistance to that value. This parameter is ignored if you pass 0.0 for a maximumDistance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Index of face that the closest point lies on if successful. -1 if not successful; the value of pointOnMesh is undefined.

Return type int

RhinoCompute.Mesh.**pointAt**(*thisMesh*, *meshPoint*, *multiple=false*)

Evaluate a mesh at a set of barycentric coordinates.

Arguments

- **meshPoint** (*MeshPoint*) – MeshPoint instance containing a valid Face Index and Barycentric coordinates.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Point on the mesh or Point3d.Unset if the faceIndex is not valid or if the barycentric coordinates could not be evaluated.

Return type rhino3dm.Point3d

RhinoCompute.Mesh.**pointAt1**(*thisMesh*, *faceIndex*, *t0*, *t1*, *t2*, *t3*, *multiple=false*)

Evaluates a mesh at a set of barycentric coordinates. Barycentric coordinates must be assigned in accordance with the rules as defined by MeshPoint.T.

Arguments

- **faceIndex** (*int*) – Index of triangle or quad to evaluate.
- **t0** (*float*) – First barycentric coordinate.
- **t1** (*float*) – Second barycentric coordinate.
- **t2** (*float*) – Third barycentric coordinate.

- **t3** (*float*) – Fourth barycentric coordinate. If the face is a triangle, this coordinate will be ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Point on the mesh or Point3d.Unset if the faceIndex is not valid or if the barycentric coordinates could not be evaluated.

Return type rhino3dm.Point3d

RhinoCompute.Mesh.**normalAt**(*thisMesh*, *meshPoint*, *multiple=false*)

Evaluate a mesh normal at a set of barycentric coordinates.

Arguments

- **meshPoint** (*MeshPoint*) – MeshPoint instance containing a valid Face Index and Barycentric coordinates.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Normal vector to the mesh or Vector3d.Unset if the faceIndex is not valid or if the barycentric coordinates could not be evaluated.

Return type rhino3dm.Vector3d

RhinoCompute.Mesh.**normalAt1**(*thisMesh*, *faceIndex*, *t0*, *t1*, *t2*, *t3*, *multiple=false*)

Evaluate a mesh normal at a set of barycentric coordinates. Barycentric coordinates must be assigned in accordance with the rules as defined by MeshPoint.T.

Arguments

- **faceIndex** (*int*) – Index of triangle or quad to evaluate.
- **t0** (*float*) – First barycentric coordinate.
- **t1** (*float*) – Second barycentric coordinate.
- **t2** (*float*) – Third barycentric coordinate.
- **t3** (*float*) – Fourth barycentric coordinate. If the face is a triangle, this coordinate will be ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Normal vector to the mesh or Vector3d.Unset if the faceIndex is not valid or if the barycentric coordinates could not be evaluated.

Return type rhino3dm.Vector3d

RhinoCompute.Mesh.**colorAt**(*thisMesh*, *meshPoint*, *multiple=false*)

Evaluate a mesh color at a set of barycentric coordinates.

Arguments

- **meshPoint** (*MeshPoint*) – MeshPoint instance containing a valid Face Index and Barycentric coordinates.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The interpolated vertex color on the mesh or Color.Transparent if the faceIndex is not valid, if the barycentric coordinates could not be evaluated, or if there are no colors defined on the mesh.

Return type Color

RhinoCompute.Mesh.**colorAt1**(*thisMesh*, *faceIndex*, *t0*, *t1*, *t2*, *t3*, *multiple=false*)

Evaluate a mesh normal at a set of barycentric coordinates. Barycentric coordinates must be assigned in accordance with the rules as defined by MeshPoint.T.

Arguments

- **faceIndex** (*int*) – Index of triangle or quad to evaluate.
- **t0** (*float*) – First barycentric coordinate.
- **t1** (*float*) – Second barycentric coordinate.
- **t2** (*float*) – Third barycentric coordinate.
- **t3** (*float*) – Fourth barycentric coordinate. If the face is a triangle, this coordinate will be ignored.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The interpolated vertex color on the mesh or Color.Transparent if the faceIndex is not valid, if the barycentric coordinates could not be evaluated, or if there are no colors defined on the mesh.

Return type Color

RhinoCompute.Mesh.**pullPointsToMesh**(*thisMesh*, *points*, *multiple=false*)

Pulls a collection of points to a mesh.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – An array, a list or any enumerable set of points.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of points. This can be empty.

Return type rhino3dm.Point3d[]

RhinoCompute.Mesh.**pullCurve**(*thisMesh*, *curve*, *tolerance*, *multiple=false*)

Gets a polyline approximation of the input curve and then moves its control points to the closest point on the mesh. Then it "connects the points" over edges so that a polyline on the mesh is formed.

Arguments

- **curve** (*rhino3dm.Curve*) – A curve to pull.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A polyline curve, or None if none could be constructed.

Return type PolylineCurve

RhinoCompute.Mesh.**splitWithProjectedPolylines**(*thisMesh*, *curves*, *tolerance*, *multiple=false*)

Splits a mesh by adding edges in correspondence with input polylines, and divides the mesh at partitioned areas. Polyline segments that are measured not to be on the mesh will be ignored.

Arguments

- **curves** (*IEnumerable<PolylineCurve>*) – An array, a list or any enumerable of polyline curves.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of meshes, or None if no change would happen.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**splitWithProjectedPolylines1**(*thisMesh*, *curves*, *tolerance*, *textLog*, *cancel*, *progress*, *multiple=false*)

Splits a mesh by adding edges in correspondence with input polylines, and divides the mesh at partitioned areas. Polyline segments that are measured not to be on the mesh will be ignored.

Arguments

- **curves** (*IEnumerable<PolylineCurve>*) – An array, a list or any enumerable of polyline curves.
- **tolerance** (*float*) – A tolerance value.
- **textLog** (*TextLog*) – A text log, or null.
- **cancel** (*CancellationToken*) – A cancellation token to stop the computation at a given point.
- **progress** (*IProgress<double>*) – A progress reporter to inform the user about progress. The reported value is indicative.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of meshes, or None if no change would happen.

Return type rhino3dm.Mesh[]

RhinoCompute.Mesh.**offset**(*thisMesh*, *distance*, *multiple=false*)

Makes a new mesh with vertices offset a distance in the opposite direction of the existing vertex normals. Same as Mesh.Offset(distance, false)

Arguments

- **distance** (*float*) – A distance value to use for offsetting.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh on success, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**offset1**(*thisMesh*, *distance*, *solidify*, *multiple=false*)

Makes a new mesh with vertices offset a distance in the opposite direction of the existing vertex normals. Optionally, based on the value of solidify, adds the input mesh and a ribbon of faces along any naked edges. If solidify is False it acts exactly as the Offset(distance) function.

Arguments

- **distance** (*float*) – A distance value.
- **solidify** (*bool*) – True if the mesh should be solidified.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh on success, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**offset2**(*thisMesh*, *distance*, *solidify*, *direction*, *multiple=false*)

Makes a new mesh with vertices offset a distance along the direction parameter. Optionally, based on the value of solidify, adds the input mesh and a ribbon of faces along any naked edges. If solidify is False it acts exactly as the Offset(distance) function.

Arguments

- **distance** (*float*) – A distance value.
- **solidify** (*bool*) – True if the mesh should be solidified.
- **direction** (*rhino3dm.Vector3d*) – Direction of offset for all vertices.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh on success, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**offset3**(*thisMesh*, *distance*, *solidify*, *direction*, *multiple=false*)

Makes a new mesh with vertices offset a distance along the direction parameter. Optionally, based on the value of solidify, adds the input mesh and a ribbon of faces along any naked edges. If solidify is False it acts exactly as the Offset(distance) function. Returns list of wall faces, i.e. the faces that connect original and offset mesh when solidified.

Arguments

- **distance** (*float*) – A distance value.
- **solidify** (*bool*) – True if the mesh should be solidified.
- **direction** (*rhino3dm.Vector3d*) – Direction of offset for all vertices.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh on success, or None on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**collapseFacesByEdgeLength**(*thisMesh*, *bGreaterThan*, *edgeLength*, *multiple=false*)

Collapses multiple mesh faces, with greater/less than edge length, based on the principles found in Stan Melax's mesh reduction PDF, see http://pomax.nihongoresources.com/downloads/PolygonReduction.pdf

Arguments

- **bGreaterThan** (*bool*) – Determines whether edge with lengths greater than or less than edgeLength are collapsed.
- **edgeLength** (*float*) – Length with which to compare to edge lengths.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Number of edges (faces) that were collapsed, -1 for general failure (like bad topology or index out of range) or -2 if all of the edges would be collapsed and the resulting mesh would be invalid.

Return type int

RhinoCompute.Mesh.**collapseFacesByArea**(*thisMesh*, *lessThanArea*, *greaterThanArea*, *multiple=false*)

Collapses multiple mesh faces, with areas less than LessThanArea and greater than GreaterThanArea, based on the principles found in Stan Melax's mesh reduction PDF, see http://pomax.nihongoresources.com/downloads/PolygonReduction.pdf

Arguments

- **lessThanArea** (*float*) – Area in which faces are selected if their area is less than or equal to.
- **greaterThanArea** (*float*) – Area in which faces are selected if their area is greater than or equal to.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Number of faces that were collapsed in the process.

Return type int

RhinoCompute.Mesh.**collapseFacesByByAspectRatio**(*thisMesh*, *aspectRatio*, *multiple=false*)

Collapses a multiple mesh faces, determined by face aspect ratio, based on criteria found in Stan Melax's polygon reduction, see http://pomax.nihongoresources.com/downloads/PolygonReduction.pdf

Arguments

- **aspectRatio** (*float*) – Faces with an aspect ratio less than aspectRatio are considered as candidates.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Number of faces that were collapsed in the process.

Return type int

RhinoCompute.Mesh.**getUnsafeLock**(*thisMesh*, *writable*, *multiple=false*)

Allows to obtain unsafe pointers to the underlying unmanaged data structures of the mesh.

Arguments

- **writable** (*bool*) – True if user will need to write onto the structure. False otherwise.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A lock that needs to be released.

Return type MeshUnsafeLock

RhinoCompute.Mesh.**releaseUnsafeLock**(*thisMesh*, *meshData*, *multiple=false*)

Updates the Mesh data with the information that was stored via the .

Arguments

- **meshData** (*MeshUnsafeLock*) – The data that will be unlocked.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type void

RhinoCompute.Mesh.**withShutLining**(*thisMesh*, *faceted*, *tolerance*, *curves*, *multiple=false*)

Constructs new mesh from the current one, with shut lining applied to it.

Arguments

- **faceted** (*bool*) – Specifies whether the shutline is faceted.
- **tolerance** (*float*) – The tolerance of the shutline.
- **curves** (*IEnumerable<ShutLiningCurveInfo>*) – A collection of curve arguments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh with shutlining. Null on failure.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**withDisplacement**(*thisMesh*, *displacement*, *multiple=false*)

Constructs new mesh from the current one, with displacement applied to it.

Arguments

- **displacement** (*MeshDisplacementInfo*) – Information on mesh displacement.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh with shutlining.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**withEdgeSoftening**(*thisMesh*, *softeningRadius*, *chamfer*, *faceted*, *force*, *angleThreshold*, *multiple=false*)

Constructs new mesh from the current one, with edge softening applied to it.

Arguments

- **softeningRadius** (*float*) – The softening radius.
- **chamfer** (*bool*) – Specifies whether to chamfer the edges.
- **faceted** (*bool*) – Specifies whether the edges are faceted.
- **force** (*bool*) – Specifies whether to soften edges despite too large a radius.
- **angleThreshold** (*float*) – Threshold angle (in degrees) which controls whether an edge is softened or not. The angle refers to the angles between the adjacent faces of an edge.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new mesh with soft edges.

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**createVertexColorsFromBitmap**(*thisMesh*, *doc*, *mapping*, *xform*, *bitmap*, *multiple=false*)

Populate the vertex colors from a bitmap image.

Arguments

- **doc** (*RhinoDoc*) – The document associated with this operation for searching purposes.
- **mapping** (*TextureMapping*) – The texture mapping to be used on the mesh. Surface parameter mapping is assumed if None - but surface parameters must be available on the mesh.
- **xform** (*Transform*) – Local mapping transform for the mesh mapping. Use identity for surface parameter mapping.

- **bitmap** (*System.Drawing.Bitmap*) – The bitmap to use for the colors.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type bool

RhinoCompute.Mesh.**quadRemeshBrep**(*brep*, *parameters*, *multiple=false*)

Create QuadRemesh from a Brep Set Brep Face Mode by setting QuadRemeshParameters.PreserveMeshArrayEdgesMode

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**quadRemeshBrep1**(*brep*, *parameters*, *guideCurves*, *multiple=false*)

Create Quad Remesh from a Brep

Arguments

- **brep** (*rhino3dm.Brep*) – Set Brep Face Mode by setting QuadRemeshParameters.PreserveMeshArrayEdgesMode
- **guideCurves** (*list[rhino3dm.Curve]*) – A curve array used to influence mesh face layout The curves should touch the input mesh Set Guide Curve Influence by using QuadRemeshParameters.GuideCurveInfluence
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**quadRemeshBrepAsync**(*brep*, *parameters*, *progress*, *cancelToken*, *multiple=false*)

Quad remesh this Brep asynchronously.

Arguments

- **brep** (*rhino3dm.Brep*) – Set Brep Face Mode by setting QuadRemeshParameters.PreserveMeshArrayEdgesMode
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type Task<Mesh>

RhinoCompute.Mesh.**quadRemeshBrepAsync1**(*brep*, *parameters*, *guideCurves*, *progress*, *cancelToken*, *multiple=false*)

Quad remesh this Brep asynchronously.

Arguments

- **brep** (*rhino3dm.Brep*) – Set Brep Face Mode by setting QuadRemeshParameters.PreserveMeshArrayEdgesMode
- **guideCurves** (*list[rhino3dm.Curve]*) – A curve array used to influence mesh face layout The curves should touch the input mesh Set Guide Curve Influence by using QuadRemeshParameters.GuideCurveInfluence
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type Task<Mesh>

RhinoCompute.Mesh.**quadRemesh**(*thisMesh*, *parameters*, *multiple=false*)

Quad remesh this mesh.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**quadRemesh1**(*thisMesh*, *parameters*, *guideCurves*, *multiple=false*)

Quad remesh this mesh.

Arguments

- **guideCurves** (*list[rhino3dm.Curve]*) – A curve array used to influence mesh face layout The curves should touch the input mesh Set Guide Curve Influence by using QuadRemeshParameters.GuideCurveInfluence
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.Mesh

RhinoCompute.Mesh.**quadRemeshAsync**(*thisMesh*, *parameters*, *progress*, *cancelToken*, *multiple=false*)

Quad remesh this mesh asynchronously.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type Task<Mesh>

RhinoCompute.Mesh.**quadRemeshAsync1**(*thisMesh*, *parameters*, *guideCurves*, *progress*, *cancelToken*, *multiple=false*)

Quad remesh this mesh asynchronously.

Arguments

- **guideCurves** (*list[rhino3dm.Curve]*) – A curve array used to influence mesh face layout The curves should touch the input mesh Set Guide Curve Influence by using QuadRemeshParameters.GuideCurveInfluence
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type Task<Mesh>

RhinoCompute.Mesh.**quadRemeshAsync2**(*thisMesh*, *faceBlocks*, *parameters*, *guideCurves*, *progress*, *cancelToken*, *multiple=false*)

Quad remesh this mesh asynchronously.

Arguments

- **guideCurves** (*list[rhino3dm.Curve]*) – A curve array used to influence mesh face layout The curves should touch the input mesh Set Guide Curve Influence by using QuadRemeshParameters.GuideCurveInfluence
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type Task<Mesh>

RhinoCompute.Mesh.**reduce**(*thisMesh*, *desiredPolygonCount*, *allowDistortion*, *accuracy*, *normalizeSize*, *multiple=false*)

Reduce polygon count

Arguments

- **desiredPolygonCount** (*int*) – desired or target number of faces
- **allowDistortion** (*bool*) – If True mesh appearance is not changed even if the target polygon count is not reached
- **accuracy** (*int*) – Integer from 1 to 10 telling how accurate reduction algorithm to use. Greater number gives more accurate results
- **normalizeSize** (*bool*) – If True mesh is fitted to an axis aligned unit cube until reduction is complete
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**reduce1**(*thisMesh*, *desiredPolygonCount*, *allowDistortion*, *accuracy*, *normalizeSize*, *threaded*, *multiple=false*)

Reduce polygon count

Arguments

- **desiredPolygonCount** (*int*) – desired or target number of faces
- **allowDistortion** (*bool*) – If True mesh appearance is not changed even if the target polygon count is not reached
- **accuracy** (*int*) – Integer from 1 to 10 telling how accurate reduction algorithm to use. Greater number gives more accurate results
- **normalizeSize** (*bool*) – If True mesh is fitted to an axis aligned unit cube until reduction is complete
- **threaded** (*bool*) – If True then will run computation inside a worker thread and ignore any provided CancellationTokens and ProgressReporters. If False then will run on main thread.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**reduce2**(*thisMesh*, *desiredPolygonCount*, *allowDistortion*, *accuracy*, *normalizeSize*, *cancelToken*, *progress*, *multiple=false*)

Reduce polygon count

Arguments

- **desiredPolygonCount** (*int*) – desired or target number of faces
- **allowDistortion** (*bool*) – If True mesh appearance is not changed even if the target polygon count is not reached

- **accuracy** (*int*) – Integer from 1 to 10 telling how accurate reduction algorithm to use. Greater number gives more accurate results
- **normalizeSize** (*bool*) – If True mesh is fitted to an axis aligned unit cube until reduction is complete
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**reduce3**(*thisMesh*, *desiredPolygonCount*, *allowDistortion*, *accuracy*, *normalizeSize*, *cancelToken*, *progress*, *threaded*, *multiple=false*)

Reduce polygon count

Arguments

- **desiredPolygonCount** (*int*) – desired or target number of faces
- **allowDistortion** (*bool*) – If True mesh appearance is not changed even if the target polygon count is not reached
- **accuracy** (*int*) – Integer from 1 to 10 telling how accurate reduction algorithm to use. Greater number gives more accurate results
- **normalizeSize** (*bool*) – If True mesh is fitted to an axis aligned unit cube until reduction is complete
- **threaded** (*bool*) – If True then will run computation inside a worker thread and ignore any provided CancellationTokens and ProgressReporters. If False then will run on main thread.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**reduce4**(*thisMesh*, *parameters*, *multiple=false*)

Reduce polygon count

Arguments

- **parameters** (*ReduceMeshParameters*) – Parameters
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**reduce5**(*thisMesh*, *parameters*, *threaded*, *multiple=false*)

Reduce polygon count

Arguments

- **parameters** (*ReduceMeshParameters*) – Parameters

- **threaded** (*bool*) – If True then will run computation inside a worker thread and ignore any provided CancellationTokens and ProgressReporters. If False then will run on main thread.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if mesh is successfully reduced and False if mesh could not be reduced for some reason.

Return type bool

RhinoCompute.Mesh.**computeThickness**(*meshes*, *maximumThickness*, *multiple=false*)

Compute thickness metrics for this mesh.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to include in thickness analysis.
- **maximumThickness** (*float*) – Maximum thickness to consider. Use as small a thickness as possible to speed up the solver.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of thickness measurements.

Return type MeshThicknessMeasurement[]

RhinoCompute.Mesh.**computeThickness1**(*meshes*, *maximumThickness*, *cancelToken*, *multiple=false*)

Compute thickness metrics for this mesh.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to include in thickness analysis.
- **maximumThickness** (*float*) – Maximum thickness to consider. Use as small a thickness as possible to speed up the solver.
- **cancelToken** (*System.Threading.CancellationToken*) – Computation cancellation token.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of thickness measurements.

Return type MeshThicknessMeasurement[]

RhinoCompute.Mesh.**computeThickness2**(*meshes*, *maximumThickness*, *sharpAngle*, *cancelToken*, *multiple=false*)

Compute thickness metrics for this mesh.

Arguments

- **meshes** (*list[rhino3dm.Mesh]*) – Meshes to include in thickness analysis.
- **maximumThickness** (*float*) – Maximum thickness to consider. Use as small a thickness as possible to speed up the solver.
- **sharpAngle** (*float*) – Sharpness angle in radians.
- **cancelToken** (*System.Threading.CancellationToken*) – Computation cancellation token.

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Array of thickness measurements.

Return type MeshThicknessMeasurement[]

RhinoCompute.Mesh.**createContourCurves**(*meshToContour*, *contourStart*, *contourEnd*, *interval*, *multiple=false*)

(Old call maintained for compatibility.)

Arguments

- **meshToContour** (*rhino3dm.Mesh*) – Avoid.
- **contourStart** (*rhino3dm.Point3d*) – Avoid.
- **contourEnd** (*rhino3dm.Point3d*) – Avoid.
- **interval** (*float*) – Avoid.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Avoid.

Return type rhino3dm.Curve[]

RhinoCompute.Mesh.**createContourCurves1**(*meshToContour*, *contourStart*, *contourEnd*, *interval*, *tolerance*, *multiple=false*)

Constructs contour curves for a mesh, sectioned along a linear axis.

Arguments

- **meshToContour** (*rhino3dm.Mesh*) – A mesh to contour.
- **contourStart** (*rhino3dm.Point3d*) – A start point of the contouring axis.
- **contourEnd** (*rhino3dm.Point3d*) – An end point of the contouring axis.
- **interval** (*float*) – An interval distance.
- **tolerance** (*float*) – A tolerance value. If negative, the positive value will be used. WARNING! Good tolerance values are in the magnitude of 10^-7, or RhinoMath.SqrtEpsilon*10. See comments at
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

RhinoCompute.Mesh.**createContourCurves2**(*meshToContour*, *sectionPlane*, *multiple=false*)

(Old call maintained for compatibility.)

Arguments

- **meshToContour** (*rhino3dm.Mesh*) – Avoid.
- **sectionPlane** (*rhino3dm.Plane*) – Avoid.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns Avoid.

Return type rhino3dm.Curve[]

RhinoCompute.Mesh.**createContourCurves3**(*meshToContour*, *sectionPlane*, *tolerance*, *multiple=false*)

Constructs contour curves for a mesh, sectioned at a plane.

Arguments

- **meshToContour** (*rhino3dm.Mesh*) – A mesh to contour.
- **sectionPlane** (*rhino3dm.Plane*) – A cutting plane.
- **tolerance** (*float*) – A tolerance value. If negative, the positive value will be used. WARNING! Good tolerance values are in the magnitude of 10^-7, or RhinoMath.SqrtEpsilon*10. See comments at
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of curves. This array can be empty.

Return type rhino3dm.Curve[]

CHAPTER 10

RhinoCompute.NurbsCurve

RhinoCompute.NurbsCurve.**makeCompatible**(*curves*, *startPt*, *endPt*, *simplifyMethod*, *numPoints*, *refitTolerance*, *angleTolerance*, *multiple=false*)

For expert use only. From the input curves, make an array of compatible NURBS curves.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – The input curves.
- **startPt** (*rhino3dm.Point3d*) – The start point. To omit, specify Point3d.Unset.
- **endPt** (*rhino3dm.Point3d*) – The end point. To omit, specify Point3d.Unset.
- **simplifyMethod** (*int*) – The simplify method.
- **numPoints** (*int*) – The number of rebuild points.
- **refitTolerance** (*float*) – The refit tolerance.
- **angleTolerance** (*float*) – The angle tolerance in radians.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The output NURBS surfaces if successful.

Return type rhino3dm.NurbsCurve[]

RhinoCompute.NurbsCurve.**createParabolaFromVertex**(*vertex*, *startPoint*, *endPoint*, *multiple=false*)

Creates a parabola from vertex and end points.

Arguments

- **vertex** (*rhino3dm.Point3d*) – The vertex point.
- **startPoint** (*rhino3dm.Point3d*) – The start point.
- **endPoint** (*rhino3dm.Point3d*) – The end point
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A 2 degree NURBS curve if successful, False otherwise.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createParabolaFromFocus**(*focus*, *startPoint*, *endPoint*, *multiple=false*)

Creates a parabola from focus and end points.

Arguments

- **focus** (*rhino3dm.Point3d*) – The focal point.
- **startPoint** (*rhino3dm.Point3d*) – The start point.
- **endPoint** (*rhino3dm.Point3d*) – The end point
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A 2 degree NURBS curve if successful, False otherwise.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createFromArc**(*arc*, *degree*, *cvCount*, *multiple=false*)

Create a uniform non-rational cubic NURBS approximation of an arc.

Arguments

- **degree** (*int*) – >=1
- **cvCount** (*int*) – CV count >=5
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns NURBS curve approximation of an arc on success

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createHSpline**(*points*, *multiple=false*)

Construct an H-spline from a sequence of interpolation points

Arguments

- **points** (*list[rhino3dm.Point3d]*) – Points to interpolate
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createHSpline1**(*points*, *startTangent*, *endTangent*, *multiple=false*)

Construct an H-spline from a sequence of interpolation points and optional start and end derivative information

Arguments

- **points** (*list[rhino3dm.Point3d]*) – Points to interpolate
- **startTangent** (*rhino3dm.Vector3d*) – Unit tangent vector or Unset
- **endTangent** (*rhino3dm.Vector3d*) – Unit tangent vector or Unset
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns NURBS curve approximation of an arc on success

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createSubDFriendly**(*points*, *interpolatePoints*, *periodicClosedCurve*, *multiple=false*)

Create a NURBS curve, that is suitable for calculations like lofting SubD objects, through a sequence of curves.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – An enumeration of points. Adjacent points must not be equal. If periodicClosedCurve is false, there must be at least two points. If periodicClosedCurve is true, there must be at least three points and it is not necessary to duplicate the first and last points. When periodicClosedCurve is True and the first and last points are equal, the duplicate last point is automatically ignored.
- **interpolatePoints** (*bool*) – True if the curve should interpolate the points. False if points specify control point locations. In either case, the curve will begin at the first point and end at the last point.
- **periodicClosedCurve** (*bool*) – True to create a periodic closed curve. Do not duplicate the start/end point in the point input.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A SubD friendly NURBS curve is successful, None otherwise.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createSubDFriendly1**(*curve*, *multiple=false*)

Create a NURBS curve, that is suitable for calculations like lofting SubD objects, from an existing curve.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to rebuild as a SubD friendly curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A SubD friendly NURBS curve is successful, None otherwise.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createSubDFriendly2**(*curve*, *pointCount*, *periodicClosedCurve*, *multiple=false*)

Create a NURBS curve, that is suitable for calculations like lofting SubD objects, from an existing curve.

Arguments

- **curve** (*rhino3dm.Curve*) – Curve to rebuild as a SubD friendly curve.
- **pointCount** (*int*) – Desired number of control points. If periodicClosedCurve is true, the number must be >= 6, otherwise the number must be >= 4.
- **periodicClosedCurve** (*bool*) – True if the SubD friendly curve should be closed and periodic. False in all other cases.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A SubD friendly NURBS curve is successful, None otherwise.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createPlanarRailFrames**(*thisNurbsCurve*, *parameters*, *normal*, *multiple=false*)

Computes planar rail sweep frames at specified parameters.

Arguments

- **parameters** (*list[float]*) – A collection of curve parameters.
- **normal** (*rhino3dm.Vector3d*) – Unit normal to the plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of planes if successful, or an empty array on failure.

Return type rhino3dm.Plane[]

RhinoCompute.NurbsCurve.**createRailFrames**(*thisNurbsCurve*, *parameters*, *multiple=false*)
Computes relatively parallel rail sweep frames at specified parameters.

Arguments

- **parameters** (*list[float]*) – A collection of curve parameters.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns An array of planes if successful, or an empty array on failure.

Return type rhino3dm.Plane[]

RhinoCompute.NurbsCurve.**createFromCircle**(*circle*, *degree*, *cvCount*, *multiple=false*)
Create a uniform non-rational cubic NURBS approximation of a circle.

Arguments

- **degree** (*int*) – >=1
- **cvCount** (*int*) – CV count >=5
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns NURBS curve approximation of a circle on success

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**setEndCondition**(*thisNurbsCurve*, *bSetEnd*, *continuity*, *point*, *tangent*, *multiple=false*)
Set end condition of a NURBS curve to point, tangent and curvature.

Arguments

- **bSetEnd** (*bool*) – true: set end of curve, false: set start of curve
- **continuity** (*NurbsCurveEndConditionType*) – Position: set start or end point, Tangency: set point and tangent, Curvature: set point, tangent and curvature
- **point** (*rhino3dm.Point3d*) – point to set
- **tangent** (*rhino3dm.Vector3d*) – tangent to set
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.NurbsCurve.**setEndCondition1**(*thisNurbsCurve*, *bSetEnd*, *continuity*, *point*, *tangent*, *curvature*, *multiple=false*)
Set end condition of a NURBS curve to point, tangent and curvature.

Arguments

- **bSetEnd** (*bool*) – true: set end of curve, false: set start of curve
- **continuity** (*NurbsCurveEndConditionType*) – Position: set start or end point, Tangency: set point and tangent, Curvature: set point, tangent and curvature
- **point** (*rhino3dm.Point3d*) – point to set
- **tangent** (*rhino3dm.Vector3d*) – tangent to set
- **curvature** (*rhino3dm.Vector3d*) – curvature to set
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.NurbsCurve.**grevillePoints**(*thisNurbsCurve*, *all*, *multiple=false*)

Gets Greville points for this curve.

Arguments

- **all** (*bool*) – If true, then all Greville points are returns. If false, only edit points are returned.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A list of points if successful, None otherwise.

Return type Point3dList

RhinoCompute.NurbsCurve.**setGrevillePoints**(*thisNurbsCurve*, *points*, *multiple=false*)

Sets all Greville edit points for this curve.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – The new point locations. The number of points should match the number of point returned by NurbsCurve.GrevillePoints(false).
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False otherwise.

Return type bool

RhinoCompute.NurbsCurve.**createSpiral**(*axisStart*, *axisDir*, *radiusPoint*, *pitch*, *turnCount*, *radius0*, *radius1*, *multiple=false*)

Creates a C1 cubic NURBS approximation of a helix or spiral. For a helix, you may have radius0 == radius1. For a spiral radius0 == radius1 produces a circle. Zero and negative radii are permissible.

Arguments

- **axisStart** (*rhino3dm.Point3d*) – Helix's axis starting point or center of spiral.
- **axisDir** (*rhino3dm.Vector3d*) – Helix's axis vector or normal to spiral's plane.
- **radiusPoint** (*rhino3dm.Point3d*) – Point used only to get a vector that is perpendicular to the axis. In particular, this vector must not be (anti)parallel to the axis vector.
- **pitch** (*float*) – The pitch, where a spiral has a pitch = 0, and pitch > 0 is the distance between the helix's "threads".

- **turnCount** (*float*) – The number of turns in spiral or helix. Positive values produce counter-clockwise orientation, negative values produce clockwise orientation. Note, for a helix, turnCount * pitch = length of the helix's axis.
- **radius0** (*float*) – The starting radius.
- **radius1** (*float*) – The ending radius.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns NurbsCurve on success, None on failure.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsCurve.**createSpiral1**(*railCurve*, *t0*, *t1*, *radiusPoint*, *pitch*, *turnCount*, *radius0*, *radius1*, *pointsPerTurn*, *multiple=false*)

Create a C2 non-rational uniform cubic NURBS approximation of a swept helix or spiral.

Arguments

- **railCurve** (*rhino3dm.Curve*) – The rail curve.
- **t0** (*float*) – Starting portion of rail curve's domain to sweep along.
- **t1** (*float*) – Ending portion of rail curve's domain to sweep along.
- **radiusPoint** (*rhino3dm.Point3d*) – Point used only to get a vector that is perpendicular to the axis. In particular, this vector must not be (anti)parallel to the axis vector.
- **pitch** (*float*) – The pitch. Positive values produce counter-clockwise orientation, negative values produce clockwise orientation.
- **turnCount** (*float*) – The turn count. If != 0, then the resulting helix will have this many turns. If = 0, then pitch must be != 0 and the approximate distance between turns will be set to pitch. Positive values produce counter-clockwise orientation, negative values produce clockwise orientation.
- **radius0** (*float*) – The starting radius. At least one radii must be nonzero. Negative values are allowed.
- **radius1** (*float*) – The ending radius. At least one radii must be nonzero. Negative values are allowed.
- **pointsPerTurn** (*int*) – Number of points to interpolate per turn. Must be greater than 4. When in doubt, use 12.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns NurbsCurve on success, None on failure.

Return type rhino3dm.NurbsCurve

CHAPTER 11

RhinoCompute.NurbsSurface

RhinoCompute.NurbsSurface.**createSubDFriendly**(*surface*, *multiple=false*)

Create a bi-cubic SubD friendly surface from a surface.

Arguments

- **surface** (*rhino3dm.Surface*) – >Surface to rebuild as a SubD friendly surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A SubD friendly NURBS surface is successful, None otherwise.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createFromPlane**(*plane*, *uInterval*, *vInterval*, *uDegree*, *vDegree*, *uPointCount*, *vPointCount*, *multiple=false*)

Creates a NURBS surface from a plane and additonal parameters.

Arguments

- **plane** (*rhino3dm.Plane*) – The plane.
- **uInterval** (*rhino3dm.Interval*) – The interval describing the extends of the output surface in the U direction.
- **vInterval** (*rhino3dm.Interval*) – The interval describing the extends of the output surface in the V direction.
- **uDegree** (*int*) – The degree of the output surface in the U direction.
- **vDegree** (*int*) – The degree of the output surface in the V direction.
- **uPointCount** (*int*) – The number of control points of the output surface in the U direction.
- **vPointCount** (*int*) – The number of control points of the output surface in the V direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NURBS surface if successful, or None on failure.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createCurveOnSurfacePoints**(*surface*, *fixedPoints*, *tolerance*, *periodic*, *initCount*, *levels*, *multiple=false*)

Computes a discrete spline curve on the surface. In other words, computes a sequence of points on the surface, each with a corresponding parameter value.

Arguments

- **surface** (*rhino3dm.Surface*) – The surface on which the curve is constructed. The surface should be G1 continuous. If the surface is closed in the u or v direction and is G1 at the seam, the function will construct point sequences that cross over the seam.
- **fixedPoints** (*list[rhino3dm.Point2d]*) – Surface points to interpolate given by parameters. These must be distinct.
- **tolerance** (*float*) – Relative tolerance used by the solver. When in doubt, use a tolerance of 0.0.
- **periodic** (*bool*) – When True constructs a smoothly closed curve.
- **initCount** (*int*) – Maximum number of points to insert between fixed points on the first level.
- **levels** (*int*) – The number of levels (between 1 and 3) to be used in multi-level solver. Use 1 for single level solve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A sequence of surface points, given by surface parameters, if successful. The number of output points is approximately: 2 ^ (level-1) * initCount * fixedPoints.Count.

Return type rhino3dm.Point2d[]

RhinoCompute.NurbsSurface.**createCurveOnSurface**(*surface*, *points*, *tolerance*, *periodic*, *multiple=false*)

Fit a sequence of 2d points on a surface to make a curve on the surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface on which to construct curve.
- **points** (*list[rhino3dm.Point2d]*) – Parameter space coordinates of the points to interpolate.
- **tolerance** (*float*) – Curve should be within tolerance of surface and points.
- **periodic** (*bool*) – When True make a periodic curve.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A curve interpolating the points if successful, None on error.

Return type rhino3dm.NurbsCurve

RhinoCompute.NurbsSurface.**makeCompatible**(*surface0*, *surface1*, *multiple=false*)

For expert use only. Makes a pair of compatible NURBS surfaces based on two input surfaces.

Arguments

- **surface0** (*rhino3dm.Surface*) – The first surface.

- **surface1** (*rhino3dm.Surface*) – The second surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful, False on failure.

Return type bool

RhinoCompute.NurbsSurface.**createFromPoints**(*points*, *uCount*, *vCount*, *uDegree*, *vDegree*, *multiple=false*)

Constructs a NURBS surface from a 2D grid of control points.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – Control point locations.
- **uCount** (*int*) – Number of points in U direction.
- **vCount** (*int*) – Number of points in V direction.
- **uDegree** (*int*) – Degree of surface in U direction.
- **vDegree** (*int*) – Degree of surface in V direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NurbsSurface on success or None on failure.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createThroughPoints**(*points*, *uCount*, *vCount*, *uDegree*, *vDegree*, *uClosed*, *vClosed*, *multiple=false*)

Constructs a NURBS surface from a 2D grid of points.

Arguments

- **points** (*list[rhino3dm.Point3d]*) – Control point locations.
- **uCount** (*int*) – Number of points in U direction.
- **vCount** (*int*) – Number of points in V direction.
- **uDegree** (*int*) – Degree of surface in U direction.
- **vDegree** (*int*) – Degree of surface in V direction.
- **uClosed** (*bool*) – True if the surface should be closed in the U direction.
- **vClosed** (*bool*) – True if the surface should be closed in the V direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NurbsSurface on success or None on failure.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createFromCorners**(*corner1*, *corner2*, *corner3*, *corner4*, *multiple=false*)

Makes a surface from 4 corner points. This is the same as calling with tolerance 0.

Arguments

- **corner1** (*rhino3dm.Point3d*) – The first corner.
- **corner2** (*rhino3dm.Point3d*) – The second corner.

- **corner3** (*rhino3dm.Point3d*) – The third corner.
- **corner4** (*rhino3dm.Point3d*) – The fourth corner.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns the resulting surface or None on error.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createFromCorners1**(*corner1*, *corner2*, *corner3*, *corner4*, *tolerance*, *multiple=false*)

Makes a surface from 4 corner points.

Arguments

- **corner1** (*rhino3dm.Point3d*) – The first corner.
- **corner2** (*rhino3dm.Point3d*) – The second corner.
- **corner3** (*rhino3dm.Point3d*) – The third corner.
- **corner4** (*rhino3dm.Point3d*) – The fourth corner.
- **tolerance** (*float*) – Minimum edge length without collapsing to a singularity.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting surface or None on error.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createFromCorners2**(*corner1*, *corner2*, *corner3*, *multiple=false*)

Makes a surface from 3 corner points.

Arguments

- **corner1** (*rhino3dm.Point3d*) – The first corner.
- **corner2** (*rhino3dm.Point3d*) – The second corner.
- **corner3** (*rhino3dm.Point3d*) – The third corner.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The resulting surface or None on error.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createRailRevolvedSurface**(*profile*, *rail*, *axis*, *scaleHeight*, *multiple=false*)

Constructs a railed Surface-of-Revolution.

Arguments

- **profile** (*rhino3dm.Curve*) – Profile curve for revolution.
- **rail** (*rhino3dm.Curve*) – Rail curve for revolution.
- **axis** (*Line*) – Axis of revolution.
- **scaleHeight** (*bool*) – If true, surface will be locally scaled.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NurbsSurface or None on failure.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createNetworkSurface**(*uCurves*, *uContinuityStart*, *uContinuityEnd*, *vCurves*, *vContinuityStart*, *vContinuityEnd*, *edgeTolerance*, *interiorTolerance*, *angleTolerance*, *multiple=false*)

Builds a surface from an ordered network of curves/edges.

Arguments

- **uCurves** (*list[rhino3dm.Curve]*) – An array, a list or any enumerable set of U curves.
- **uContinuityStart** (*int*) – continuity at first U segment, 0 = loose, 1 = position, 2 = tan, 3 = curvature.
- **uContinuityEnd** (*int*) – continuity at last U segment, 0 = loose, 1 = position, 2 = tan, 3 = curvature.
- **vCurves** (*list[rhino3dm.Curve]*) – An array, a list or any enumerable set of V curves.
- **vContinuityStart** (*int*) – continuity at first V segment, 0 = loose, 1 = position, 2 = tan, 3 = curvature.
- **vContinuityEnd** (*int*) – continuity at last V segment, 0 = loose, 1 = position, 2 = tan, 3 = curvature.
- **edgeTolerance** (*float*) – tolerance to use along network surface edge.
- **interiorTolerance** (*float*) – tolerance to use for the interior curves.
- **angleTolerance** (*float*) – angle tolerance to use.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NurbsSurface or None on failure.

Return type NurbsSurface

RhinoCompute.NurbsSurface.**createNetworkSurface1**(*curves*, *continuity*, *edgeTolerance*, *interiorTolerance*, *angleTolerance*, *multiple=false*)

Builds a surface from an auto-sorted network of curves/edges.

Arguments

- **curves** (*list[rhino3dm.Curve]*) – An array, a list or any enumerable set of curves/edges, sorted automatically into U and V curves.
- **continuity** (*int*) – continuity along edges, 0 = loose, 1 = position, 2 = tan, 3 = curvature.
- **edgeTolerance** (*float*) – tolerance to use along network surface edge.
- **interiorTolerance** (*float*) – tolerance to use for the interior curves.
- **angleTolerance** (*float*) – angle tolerance to use.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A NurbsSurface or None on failure.

Return type NurbsSurface

CHAPTER 12

RhinoCompute.SubD

RhinoCompute.SubD.**joinSubDs**(*subdsToJoin*, *tolerance*, *joinedEdgesAreCreases*, *multiple=false*)

Joins an enumeration of SubDs to form as few as possible resulting SubDs. There may be more than one SubD in the result array.

Arguments

- **subdsToJoin** (*IEnumerable<SubD>*) – An enumeration of SubDs to join.
- **tolerance** (*float*) – The join tolerance.
- **joinedEdgesAreCreases** (*bool*) – If true, merged boundary edges will be creases. If false, merged boundary edges will be smooth.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type SubD[]

RhinoCompute.SubD.**toBrep**(*thisSubD*, *options*, *multiple=false*)

Create a Brep based on this SubD geometry.

Arguments

- **options** (*SubDToBrepOptions*) – The SubD to Brep conversion options. Use SubDToBrepOptions.Default for sensible defaults. Currently, these return unpacked faces and locally-G1 vertices in the output Brep.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new Brep if successful, or None on failure.

Return type rhino3dm.Brep

RhinoCompute.SubD.**toBrep1**(*thisSubD*, *multiple=false*)

Create a Brep based on this SubD geometry, based on SubDToBrepOptions.Default options.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new Brep if successful, or None on failure.

Return type rhino3dm.Brep

RhinoCompute.SubD.**createFromMesh**(*mesh*, *multiple=false*)
Create a new SubD from a mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – The input mesh.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**createFromMesh1**(*mesh*, *options*, *multiple=false*)
Create a new SubD from a mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – The input mesh.
- **options** (*SubDCreationOptions*) – The SubD creation options.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**createFromSurface**(*surface*, *method*, *corners*, *multiple=false*)
Create a SubD that approximates the surface. If the surface is a SubD friendly NURBS surface and withCorners is true, then the SubD and input surface will have the same geometry.

Arguments

- **method** (*SubDFromSurfaceMethods*) – Selects the method used to calculate the SubD.
- **corners** (*bool*) – If the surface is open, then the corner vertices with be tagged as VertexTagCorner. This makes the resulting SubD have sharp corners to match the appearance of the input surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type SubD

RhinoCompute.SubD.**offset**(*thisSubD*, *distance*, *solidify*, *multiple=false*)
Makes a new SubD with vertices offset at distance in the direction of the control net vertex normals. Optionally, based on the value of solidify, adds the input SubD and a ribbon of faces along any naked edges.

Arguments

- **distance** (*float*) – The distance to offset.
- **solidify** (*bool*) – True if the output SubD should be turned into a closed SubD.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**createFromLoft**(*curves*, *closed*, *addCorners*, *addCreases*, *divisions*, *multiple=false*)

Creates a SubD lofted through shape curves.

Arguments

- **curves** (*list[rhino3dm.NurbsCurve]*) – An enumeration of SubD-friendly NURBS curves to loft through.
- **closed** (*bool*) – Creates a SubD that is closed in the lofting direction. Must have three or more shape curves.
- **addCorners** (*bool*) – With open curves, adds creased vertices to the SubD at both ends of the first and last curves.
- **addCreases** (*bool*) – With kinked curves, adds creased edges to the SubD along the kinks.
- **divisions** (*int*) – The segment number between adjacent input curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**createFromSweep**(*rail1*, *shapes*, *closed*, *addCorners*, *roadlikeFrame*, *roadlikeNormal*, *multiple=false*)

Fits a SubD through a series of profile curves that define the SubD cross-sections and one curve that defines a SubD edge.

Arguments

- **rail1** (*rhino3dm.NurbsCurve*) – A SubD-friendly NURBS curve to sweep along.
- **shapes** (*list[rhino3dm.NurbsCurve]*) – An enumeration of SubD-friendly NURBS curves to sweep through.
- **closed** (*bool*) – Creates a SubD that is closed in the rail curve direction.
- **addCorners** (*bool*) – With open curves, adds creased vertices to the SubD at both ends of the first and last curves.
- **roadlikeFrame** (*bool*) – Determines how sweep frame rotations are calculated. If False (Freeform), frame are propogated based on a refrence direction taken from the rail curve curvature direction. If True (Roadlike), frame rotations are calculated based on a vector supplied in "roadlikeNormal" and the world coordinate system.
- **roadlikeNormal** (*rhino3dm.Vector3d*) – If roadlikeFrame = true, provide 3D vector used to calculate the frame rotations for sweep shapes. If roadlikeFrame = false, then pass .
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**createFromSweep1**(*rail1*, *rail2*, *shapes*, *closed*, *addCorners*, *multiple=false*)

Fits a SubD through a series of profile curves that define the SubD cross-sections and two curves that defines SubD edges.

Arguments

- **rail1** (*rhino3dm.NurbsCurve*) – The first SubD-friendly NURBS curve to sweep along.
- **rail2** (*rhino3dm.NurbsCurve*) – The second SubD-friendly NURBS curve to sweep along.
- **shapes** (*list[rhino3dm.NurbsCurve]*) – An enumeration of SubD-friendly NURBS curves to sweep through.
- **closed** (*bool*) – Creates a SubD that is closed in the rail curve direction.
- **addCorners** (*bool*) – With open curves, adds creased vertices to the SubD at both ends of the first and last curves.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new SubD if successful, or None on failure.

Return type SubD

RhinoCompute.SubD.**mergeAllCoplanarFaces**(*thisSubD*, *tolerance*, *multiple=false*)

Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.SubD.**mergeAllCoplanarFaces1**(*thisSubD*, *tolerance*, *angleTolerance*, *multiple=false*)

Merges adjacent coplanar faces into single faces.

Arguments

- **tolerance** (*float*) – Tolerance for determining when edges are adjacent. When in doubt, use the document's ModelAbsoluteTolerance property.
- **angleTolerance** (*float*) – Angle tolerance, in radians, for determining when faces are parallel. When in doubt, use the document's ModelAngleToleranceRadians property.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if faces were merged, False if no faces were merged.

Return type bool

RhinoCompute.SubD.**interpolateSurfacePoints**(*thisSubD*, *surfacePoints*, *multiple=false*)

Modifies the SubD so that the SubD vertex limit surface points are equal to surface_points[]

Arguments

- **surfacePoints** (*rhino3dm.Point3d[]*) – point for limit surface to interpolate. surface_points[i] is the location for the i-th vertex returned by SubVertexIterator vit(this)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Return type bool

CHAPTER 13

RhinoCompute.Surface

RhinoCompute.Surface.**createRollingBallFillet**(*surfaceA*, *surfaceB*, *radius*, *tolerance*, *multiple=false*)

Constructs a rolling ball fillet between two surfaces.

Arguments

- **surfaceA** (*rhino3dm.Surface*) – A first surface.
- **surfaceB** (*rhino3dm.Surface*) – A second surface.
- **radius** (*float*) – A radius value.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of rolling ball fillet surfaces; this array can be empty on failure.

Return type rhino3dm.Surface[]

RhinoCompute.Surface.**createRollingBallFillet1**(*surfaceA*, *flipA*, *surfaceB*, *flipB*, *radius*, *tolerance*, *multiple=false*)

Constructs a rolling ball fillet between two surfaces.

Arguments

- **surfaceA** (*rhino3dm.Surface*) – A first surface.
- **flipA** (*bool*) – A value that indicates whether A should be used in flipped mode.
- **surfaceB** (*rhino3dm.Surface*) – A second surface.
- **flipB** (*bool*) – A value that indicates whether B should be used in flipped mode.
- **radius** (*float*) – A radius value.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of rolling ball fillet surfaces; this array can be empty on failure.

Return type rhino3dm.Surface[]

RhinoCompute.Surface.**createRollingBallFillet2**(*surfaceA*, *uvA*, *surfaceB*, *uvB*, *radius*, *tolerance*, *multiple=false*)

Constructs a rolling ball fillet between two surfaces.

Arguments

- **surfaceA** (*rhino3dm.Surface*) – A first surface.
- **uvA** (*rhino3dm.Point2d*) – A point in the parameter space of FaceA near where the fillet is expected to hit the surface.
- **surfaceB** (*rhino3dm.Surface*) – A second surface.
- **uvB** (*rhino3dm.Point2d*) – A point in the parameter space of FaceB near where the fillet is expected to hit the surface.
- **radius** (*float*) – A radius value.
- **tolerance** (*float*) – A tolerance value used for approximating and intersecting offset surfaces.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new array of rolling ball fillet surfaces; this array can be empty on failure.

Return type rhino3dm.Surface[]

RhinoCompute.Surface.**createExtrusion**(*profile*, *direction*, *multiple=false*)

Constructs a surface by extruding a curve along a vector.

Arguments

- **profile** (*rhino3dm.Curve*) – Profile curve to extrude.
- **direction** (*rhino3dm.Vector3d*) – Direction and length of extrusion.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A surface on success or None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**createExtrusionToPoint**(*profile*, *apexPoint*, *multiple=false*)

Constructs a surface by extruding a curve to a point.

Arguments

- **profile** (*rhino3dm.Curve*) – Profile curve to extrude.
- **apexPoint** (*rhino3dm.Point3d*) – Apex point of extrusion.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Surface on success or None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**createPeriodicSurface**(*surface*, *direction*, *multiple=false*)

Constructs a periodic surface from a base surface and a direction.

Arguments

- **surface** (*rhino3dm.Surface*) – The surface to make periodic.
- **direction** (*int*) – The direction to make periodic, either 0 = U, or 1 = V.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A Surface on success or None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**createPeriodicSurface1**(*surface*, *direction*, *bSmooth*, *multiple=false*)

Constructs a periodic surface from a base surface and a direction.

Arguments

- **surface** (*rhino3dm.Surface*) – The surface to make periodic.
- **direction** (*int*) – The direction to make periodic, either 0 = U, or 1 = V.
- **bSmooth** (*bool*) – Controls kink removal. If true, smooths any kinks in the surface and moves control points to make a smooth surface. If false, control point locations are not changed or changed minimally (only one point may move) and only the knot vector is altered.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A periodic surface if successful, None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**createSoftEditSurface**(*surface*, *uv*, *delta*, *uLength*, *vLength*, *tolerance*, *fixEnds*, *multiple=false*)

Creates a soft edited surface from an existing surface using a smooth field of influence.

Arguments

- **surface** (*rhino3dm.Surface*) – The surface to soft edit.
- **uv** (*rhino3dm.Point2d*) – A point in the parameter space to move from. This location on the surface is moved, and the move is smoothly tapered off with increasing distance along the surface from this parameter.
- **delta** (*rhino3dm.Vector3d*) – The direction and magnitude, or maximum distance, of the move.
- **uLength** (*float*) – The distance along the surface's u-direction from the editing point over which the strength of the editing falls off smoothly.
- **vLength** (*float*) – The distance along the surface's v-direction from the editing point over which the strength of the editing falls off smoothly.
- **tolerance** (*float*) – The active document's model absolute tolerance.
- **fixEnds** (*bool*) – Keeps edge locations fixed.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The soft edited surface if successful. None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**smooth**(*thisSurface*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *multiple=false*)

Smooths a surface by averaging the positions of control points in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much control points move towards the average of the neighboring control points.
- **bXSmooth** (*bool*) – When True control points move in X axis direction.
- **bYSmooth** (*bool*) – When True control points move in Y axis direction.
- **bZSmooth** (*bool*) – When True control points move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True the surface edges don't move.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The smoothed surface if successful, None otherwise.

Return type rhino3dm.Surface

RhinoCompute.Surface.**smooth1**(*thisSurface*, *smoothFactor*, *bXSmooth*, *bYSmooth*, *bZSmooth*, *bFixBoundaries*, *coordinateSystem*, *plane*, *multiple=false*)

Smooths a surface by averaging the positions of control points in a specified region.

Arguments

- **smoothFactor** (*float*) – The smoothing factor, which controls how much control points move towards the average of the neighboring control points.
- **bXSmooth** (*bool*) – When True control points move in X axis direction.
- **bYSmooth** (*bool*) – When True control points move in Y axis direction.
- **bZSmooth** (*bool*) – When True control points move in Z axis direction.
- **bFixBoundaries** (*bool*) – When True the surface edges don't move.
- **coordinateSystem** (*SmoothingCoordinateSystem*) – The coordinates to determine the direction of the smoothing.
- **plane** (*rhino3dm.Plane*) – If SmoothingCoordinateSystem.CPlane specified, then the construction plane.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The smoothed surface if successful, None otherwise.

Return type rhino3dm.Surface

RhinoCompute.Surface.**variableOffset**(*thisSurface*, *uMinvMin*, *uMinvMax*, *uMaxvMin*, *uMaxvMax*, *tolerance*, *multiple=false*)

Copies a surface so that all locations at the corners of the copied surface are specified distances from the original surface.

Arguments

- **uMinvMin** (*float*) – Offset distance at Domain(0).Min, Domain(1).Min.
- **uMinvMax** (*float*) – Offset distance at Domain(0).Min, Domain(1).Max.
- **uMaxvMin** (*float*) – Offset distance at Domain(0).Max, Domain(1).Min.
- **uMaxvMax** (*float*) – Offset distance at Domain(0).Max, Domain(1).Max.

- **tolerance** (*float*) – The offset tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The offset surface if successful, None otherwise.

Return type rhino3dm.Surface

RhinoCompute.Surface.**variableOffset1**(*thisSurface*, *uMinvMin*, *uMinvMax*, *uMaxvMin*, *uMaxvMax*, *interiorParameters*, *interiorDistances*, *tolerance*, *multiple=false*)

Copies a surface so that all locations at the corners, and from specified interior locations, of the copied surface are specified distances from the original surface.

Arguments

- **uMinvMin** (*float*) – Offset distance at Domain(0).Min, Domain(1).Min.
- **uMinvMax** (*float*) – Offset distance at Domain(0).Min, Domain(1).Max.
- **uMaxvMin** (*float*) – Offset distance at Domain(0).Max, Domain(1).Min.
- **uMaxvMax** (*float*) – Offset distance at Domain(0).Max, Domain(1).Max.
- **interiorParameters** (*list[rhino3dm.Point2d]*) – An array of interior UV parameters to offset from.
- **interiorDistances** (*list[float]*) – >An array of offset distances at the interior UV parameters.
- **tolerance** (*float*) – The offset tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The offset surface if successful, None otherwise.

Return type rhino3dm.Surface

RhinoCompute.Surface.**getSurfaceSize**(*thisSurface*, *multiple=false*)

Gets an estimate of the size of the rectangle that would be created if the 3d surface where flattened into a rectangle.

Arguments

- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful.

Return type bool

RhinoCompute.Surface.**closestSide**(*thisSurface*, *u*, *v*, *multiple=false*)

Gets the side that is closest, in terms of 3D-distance, to a U and V parameter.

Arguments

- **u** (*float*) – A u parameter.
- **v** (*float*) – A v parameter.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A side.

Return type IsoStatus

RhinoCompute.Surface.**extend**(*thisSurface*, *edge*, *extensionLength*, *smooth*, *multiple=false*)
Extends an untrimmed surface along one edge.

Arguments

- **edge** (*IsoStatus*) – Edge to extend. Must be North, South, East, or West.
- **extensionLength** (*float*) – distance to extend.
- **smooth** (*bool*) – True for smooth (C-infinity) extension. False for a C1- ruled extension.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns New extended surface on success.

Return type rhino3dm.Surface

RhinoCompute.Surface.**rebuild**(*thisSurface*, *uDegree*, *vDegree*, *uPointCount*, *vPointCount*, *multiple=false*)
Rebuilds an existing surface to a given degree and point count.

Arguments

- **uDegree** (*int*) – the output surface u degree.
- **vDegree** (*int*) – the output surface u degree.
- **uPointCount** (*int*) – The number of points in the output surface u direction. Must be bigger than uDegree (maximum value is 1000)
- **vPointCount** (*int*) – The number of points in the output surface v direction. Must be bigger than vDegree (maximum value is 1000)
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns new rebuilt surface on success. None on failure.

Return type NurbsSurface

RhinoCompute.Surface.**rebuildOneDirection**(*thisSurface*, *direction*, *pointCount*, *loftType*, *refitTolerance*, *multiple=false*)
Rebuilds an existing surface with a new surface to a given point count in either the u or v directions independently.

Arguments

- **direction** (*int*) – The direction (0 = U, 1 = V).
- **pointCount** (*int*) – The number of points in the output surface in the "direction" direction.
- **loftType** (*LoftType*) – The loft type
- **refitTolerance** (*float*) – The refit tolerance. When in doubt, use the document's model absolute tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns new rebuilt surface on success. None on failure.

Return type NurbsSurface

RhinoCompute.Surface.**closestPoint**(*thisSurface*, *testPoint*, *multiple=false*)
Input the parameters of the point on the surface that is closest to testPoint.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – A point to test against.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True on success, False on failure.

Return type bool

RhinoCompute.Surface.**localClosestPoint**(*thisSurface*, *testPoint*, *seedU*, *seedV*, *multiple=false*)

Find parameters of the point on a surface that is locally closest to the testPoint. The search for a local close point starts at seed parameters.

Arguments

- **testPoint** (*rhino3dm.Point3d*) – A point to test against.
- **seedU** (*float*) – The seed parameter in the U direction.
- **seedV** (*float*) – The seed parameter in the V direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if the search is successful, False if the search fails.

Return type bool

RhinoCompute.Surface.**offset**(*thisSurface*, *distance*, *tolerance*, *multiple=false*)

Constructs a new surface which is offset from the current surface.

Arguments

- **distance** (*float*) – Distance (along surface normal) to offset.
- **tolerance** (*float*) – Offset accuracy.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The offset surface or None on failure.

Return type rhino3dm.Surface

RhinoCompute.Surface.**fit**(*thisSurface*, *uDegree*, *vDegree*, *fitTolerance*, *multiple=false*)

Fits a new surface through an existing surface.

Arguments

- **uDegree** (*int*) – the output surface U degree. Must be bigger than 1.
- **vDegree** (*int*) – the output surface V degree. Must be bigger than 1.
- **fitTolerance** (*float*) – The fitting tolerance.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A surface, or None on error.

Return type rhino3dm.Surface

RhinoCompute.Surface.**interpolatedCurveOnSurfaceUV**(*thisSurface*, *points*, *tolerance*, *multiple=false*)

Returns a curve that interpolates points on a surface. The interpolant lies on the surface.

Arguments

- **points** (*System.Collections.Generic.IEnumerable<Point2d>*) – List of at least two UV parameter locations on the surface.
- **tolerance** (*float*) – Tolerance used for the fit of the push-up curve. Generally, the resulting interpolating curve will be within tolerance of the surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new NURBS curve if successful, or None on error.

Return type rhino3dm.NurbsCurve

RhinoCompute.Surface.**interpolatedCurveOnSurfaceUV1**(*thisSurface*, *points*, *tolerance*, *closed*, *closedSurfaceHandling*, *multiple=false*)

Returns a curve that interpolates points on a surface. The interpolant lies on the surface.

Arguments

- **points** (*System.Collections.Generic.IEnumerable<Point2d>*) – List of at least two UV parameter locations on the surface.
- **tolerance** (*float*) – Tolerance used for the fit of the push-up curve. Generally, the resulting interpolating curve will be within tolerance of the surface.
- **closed** (*bool*) – If false, the interpolating curve is not closed. If true, the interpolating curve is closed, and the last point and first point should generally not be equal.
- **closedSurfaceHandling** (*int*) – If 0, all points must be in the rectangular domain of the surface. If the surface is closed in some direction, then this routine will interpret each point and place it at an appropriate location in the covering space. This is the simplest option and should give good results. If 1, then more options for more control of handling curves going across seams are available. If the surface is closed in some direction, then the points are taken as points in the covering space. Example, if srf.IsClosed(0)=True and srf.IsClosed(1)=False and srf.Domain(0)=srf.Domain(1)=Interval(0,1) then if closedSurfaceHandling=1 a point(u, v) in points can have any value for the u coordinate, but must have 0<=v<=1. In particular, if points = { (0.0,0.5), (2.0,0.5) } then the interpolating curve will wrap around the surface two times in the closed direction before ending at start of the curve. If closed=True the last point should equal the first point plus an integer multiple of the period on a closed direction.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new NURBS curve if successful, or None on error.

Return type rhino3dm.NurbsCurve

RhinoCompute.Surface.**interpolatedCurveOnSurface**(*thisSurface*, *points*, *tolerance*, *multiple=false*)

Constructs an interpolated curve on a surface, using 3D points.

Arguments

- **points** (*System.Collections.Generic.IEnumerable<Point3d>*) – A list, an array or any enumerable set of points.
- **tolerance** (*float*) – A tolerance value.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns A new NURBS curve, or None on error.

Return type rhino3dm.NurbsCurve

RhinoCompute.Surface.**shortPath**(*thisSurface*, *start*, *end*, *tolerance*, *multiple=false*)
Constructs a geodesic between 2 points, used by ShortPath command in Rhino.

Arguments

- **start** (*rhino3dm.Point2d*) – start point of curve in parameter space. Points must be distinct in the domain of the surface.
- **end** (*rhino3dm.Point2d*) – end point of curve in parameter space. Points must be distinct in the domain of the surface.
- **tolerance** (*float*) – tolerance used in fitting discrete solution.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns a geodesic curve on the surface on success. None on failure.

Return type rhino3dm.Curve

RhinoCompute.Surface.**pushup**(*thisSurface*, *curve2d*, *tolerance*, *curve2dSubdomain*, *multiple=false*)
Computes a 3d curve that is the composite of a 2d curve and the surface map.

Arguments

- **curve2d** (*rhino3dm.Curve*) – a 2d curve whose image is in the surface's domain.
- **tolerance** (*float*) – the maximum acceptable distance from the returned 3d curve to the image of curve_2d on the surface.
- **curve2dSubdomain** (*rhino3dm.Interval*) – The curve interval (a sub-domain of the original curve) to use.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns 3d curve.

Return type rhino3dm.Curve

RhinoCompute.Surface.**pushup1**(*thisSurface*, *curve2d*, *tolerance*, *multiple=false*)
Computes a 3d curve that is the composite of a 2d curve and the surface map.

Arguments

- **curve2d** (*rhino3dm.Curve*) – a 2d curve whose image is in the surface's domain.
- **tolerance** (*float*) – the maximum acceptable distance from the returned 3d curve to the image of curve_2d on the surface.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns 3d curve.

Return type rhino3dm.Curve

RhinoCompute.Surface.**pullback**(*thisSurface*, *curve3d*, *tolerance*, *multiple=false*)
Pulls a 3d curve back to the surface's parameter space.

Arguments

- **curve3d** (*rhino3dm.Curve*) – The curve to pull.

- **tolerance** (*float*) – the maximum acceptable 3d distance between from surface(curve_2d(t)) to the locus of points on the surface that are closest to curve_3d.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns 2d curve.

Return type rhino3dm.Curve

RhinoCompute.Surface.**pullback1**(*thisSurface*, *curve3d*, *tolerance*, *curve3dSubdomain*, *multiple=false*)

Pulls a 3d curve back to the surface's parameter space.

Arguments

- **curve3d** (*rhino3dm.Curve*) – A curve.
- **tolerance** (*float*) – the maximum acceptable 3d distance between from surface(curve_2d(t)) to the locus of points on the surface that are closest to curve_3d.
- **curve3dSubdomain** (*rhino3dm.Interval*) – A sub-domain of the curve to sample.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns 2d curve.

Return type rhino3dm.Curve

CHAPTER 14

RhinoCompute.VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute**(*mesh*, *multiple=false*)

Compute the VolumeMassProperties for a single Mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Mesh or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute1**(*mesh*, *volume*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the VolumeMassProperties for a single Mesh.

Arguments

- **mesh** (*rhino3dm.Mesh*) – Mesh to measure.
- **volume** (*bool*) – True to calculate volume.
- **firstMoments** (*bool*) – True to calculate volume first moments, volume, and volume centroid.
- **secondMoments** (*bool*) – True to calculate volume second moments.
- **productMoments** (*bool*) – True to calculate volume product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Mesh or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute2**(*brep*, *multiple=false*)

Compute the VolumeMassProperties for a single Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Brep or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute3**(*brep*, *volume*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the VolumeMassProperties for a single Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **volume** (*bool*) – True to calculate volume.
- **firstMoments** (*bool*) – True to calculate volume first moments, volume, and volume centroid.
- **secondMoments** (*bool*) – True to calculate volume second moments.
- **productMoments** (*bool*) – True to calculate volume product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Brep or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute4**(*brep*, *volume*, *firstMoments*, *secondMoments*, *productMoments*, *relativeTolerance*, *absoluteTolerance*, *multiple=false*)

Compute the VolumeMassProperties for a single Brep.

Arguments

- **brep** (*rhino3dm.Brep*) – Brep to measure.
- **volume** (*bool*) – True to calculate volume.
- **firstMoments** (*bool*) – True to calculate volume first moments, volume, and volume centroid.
- **secondMoments** (*bool*) – True to calculate volume second moments.
- **productMoments** (*bool*) – True to calculate volume product moments.
- **relativeTolerance** (*float*) – The relative tolerance used for the calculation. In overloads of this function where tolerances are not specified, 1.0e-6 is used.
- **absoluteTolerance** (*float*) – The absolute tolerancwe used for the calculation. In overloads of this function where tolerances are not specified, 1.0e-6 is used.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Brep or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute5**(*surface*, *multiple=false*)

Compute the VolumeMassProperties for a single Surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to measure.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Surface or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute6**(*surface*, *volume*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Compute the VolumeMassProperties for a single Surface.

Arguments

- **surface** (*rhino3dm.Surface*) – Surface to measure.
- **volume** (*bool*) – True to calculate volume.
- **firstMoments** (*bool*) – True to calculate volume first moments, volume, and volume centroid.
- **secondMoments** (*bool*) – True to calculate volume second moments.
- **productMoments** (*bool*) – True to calculate volume product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the given Surface or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute7**(*geometry*, *multiple=false*)

Computes the VolumeMassProperties for a collection of geometric objects. At present only Breps, Surfaces, and Meshes are supported.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – Objects to include in the volume computation.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the entire collection or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**compute8**(*geometry*, *volume*, *firstMoments*, *secondMoments*, *productMoments*, *multiple=false*)

Computes the VolumeMassProperties for a collection of geometric objects. At present only Breps, Surfaces, Meshes and Planar Closed Curves are supported.

Arguments

- **geometry** (*list[rhino3dm.GeometryBase]*) – Objects to include in the volume computation.
- **volume** (*bool*) – True to calculate volume.
- **firstMoments** (*bool*) – True to calculate volume first moments, volume, and volume centroid.
- **secondMoments** (*bool*) – True to calculate volume second moments.

- **productMoments** (*bool*) – True to calculate volume product moments.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns The VolumeMassProperties for the entire collection or None on failure.

Return type VolumeMassProperties

RhinoCompute.VolumeMassProperties.**sum**(*thisVolumeMassProperties*, *summand*, *multiple=false*)

Sum mass properties together to get an aggregate mass.

Arguments

- **summand** (*VolumeMassProperties*) – mass properties to add.
- **multiple** (*bool*) – (default False) If True, all parameters are expected as lists of equal length and input will be batch processed

Returns True if successful.

Return type bool

CHAPTER 15

Indices and tables

- genindex
- modindex
- search

Index

R