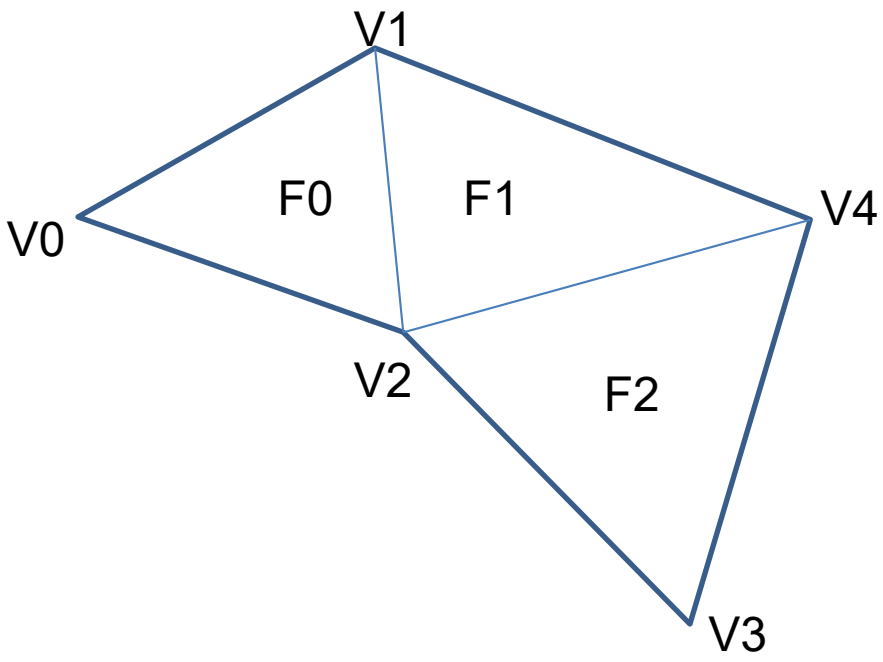
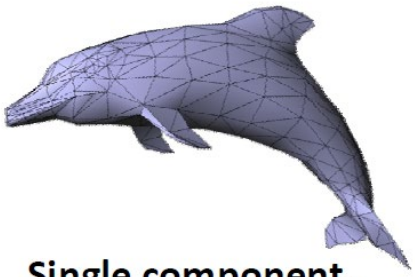
3_3_3: Meshes

A general mesh is conceptually a collection of triangles – or possibly more general polygons (ie quads). Precisely we can define:

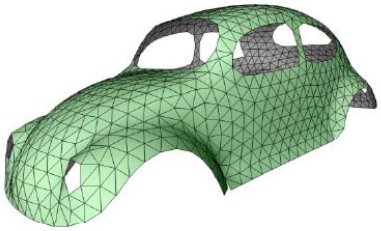
- A collection of vertices V0 - … Vn
- A collection of faces that reference the indices of these vertices:
  F0 -> (V0, V1, V2)
- Possible additional information on each of the faces, or the vertices of the faces.

Any topological surface, collection of surfaces, or closed “BREP” can be meshed with triangles or more general polygons.

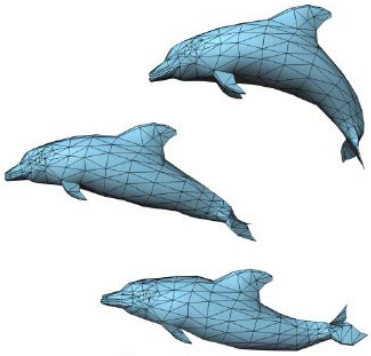




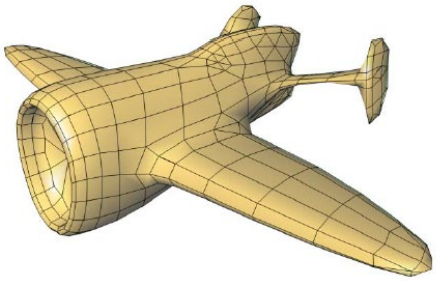
Single component, closed, triangular, orientable manifold



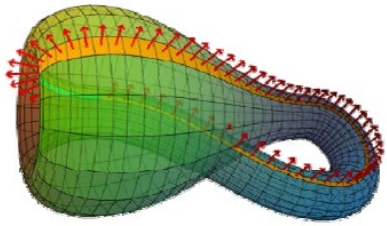
With boundaries



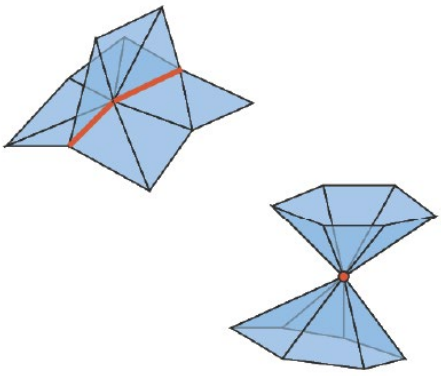
Multiple components



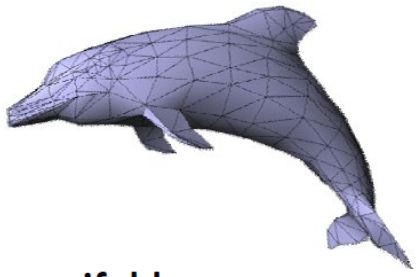
Not only triangles



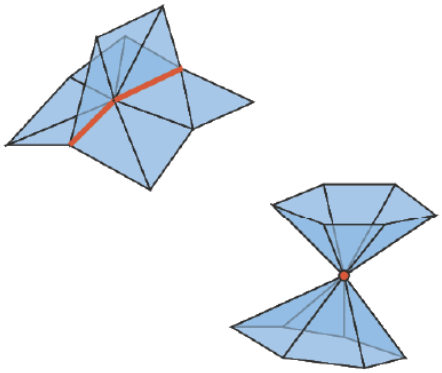
Not orientable



Non manifold



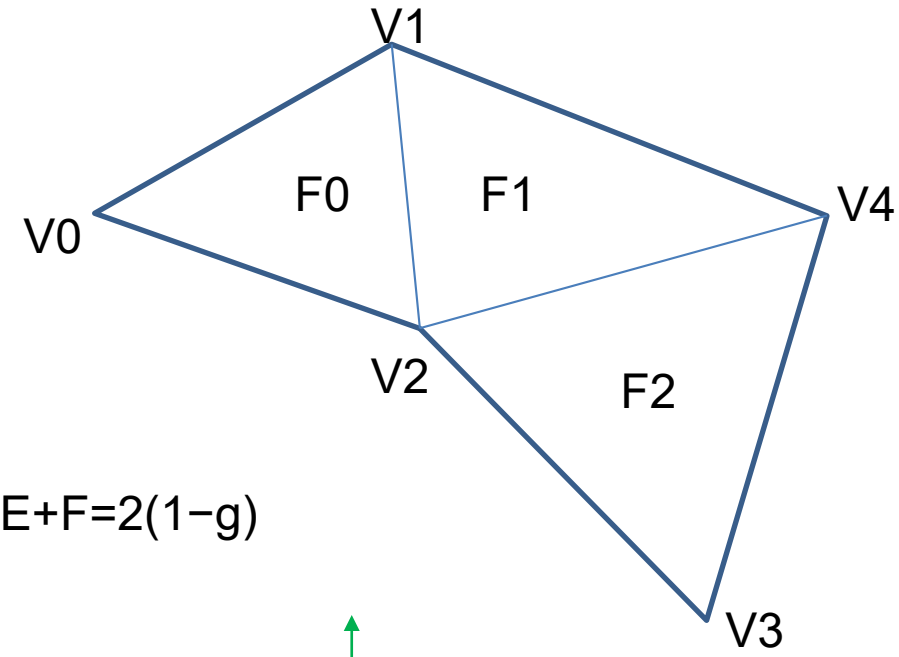
manifold



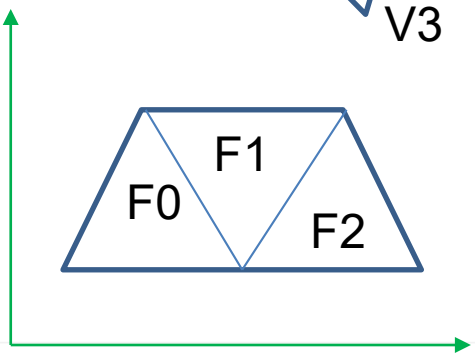
Non manifold

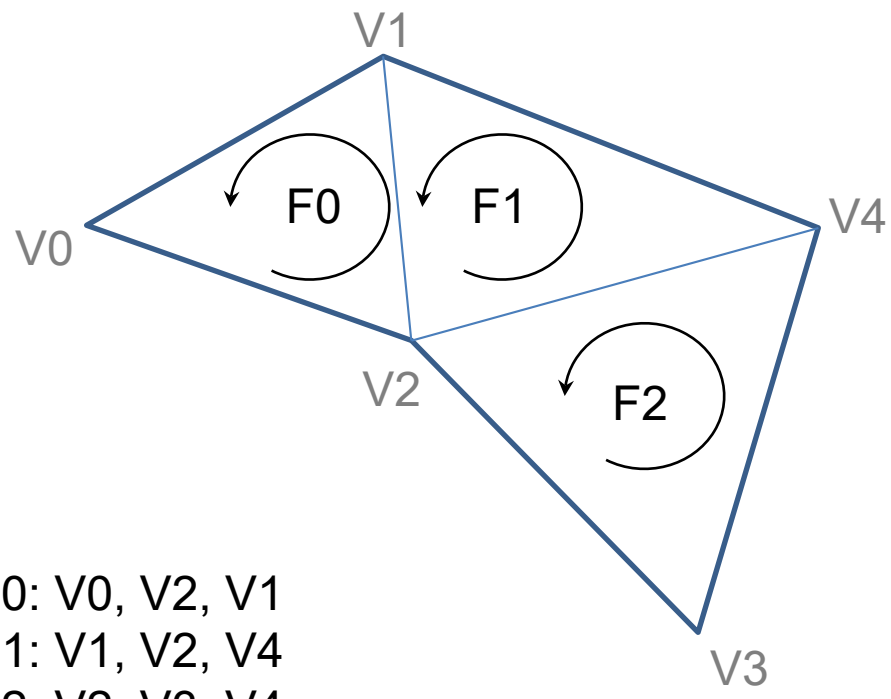
In mathematics, a **manifold** is a topological space that locally resembles Euclidean space near each point. More precisely, an n-dimensional manifold, or *n-manifold* for short, is a topological space with the property that each point has a neighborhood that is homeomorphic to an open subset of n-dimensional Euclidean space.

One-dimensional manifolds include lines and circles, but not lemniscates. Two-dimensional manifolds are also called surfaces. Examples include the plane, the sphere, and the torus, and also the Klein bottle and real projective plane.



Euler's formula: V−E+F=2(1−g)





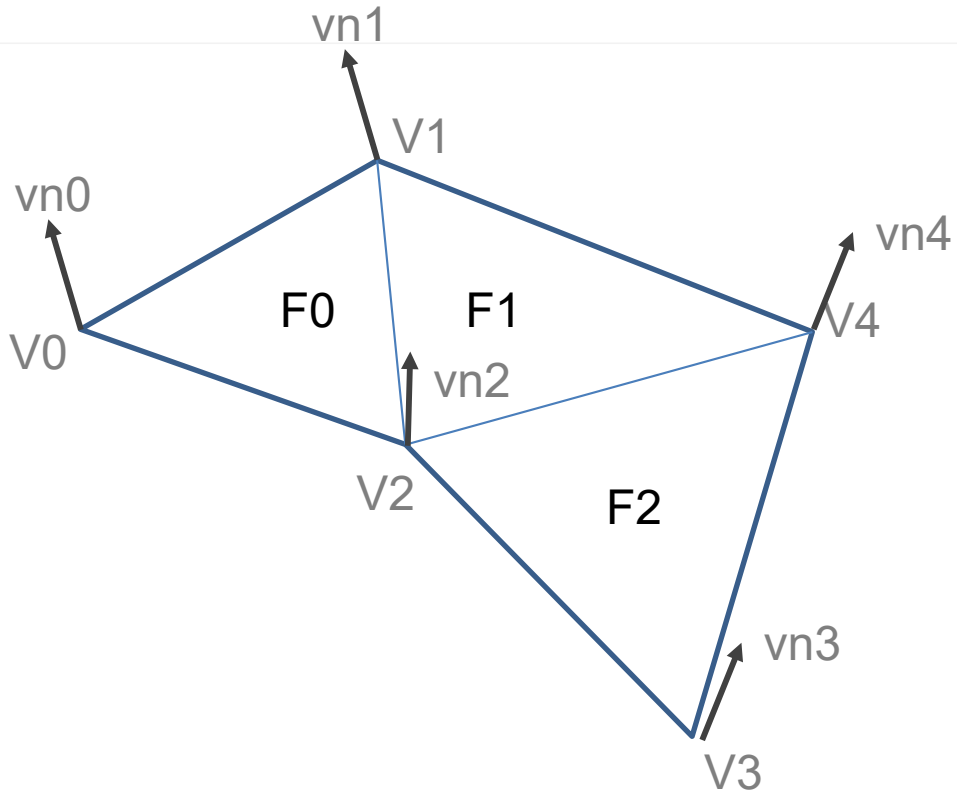
F0: V0, V2, V1
F1: V1, V2, V4
F2: V2, V3, V4

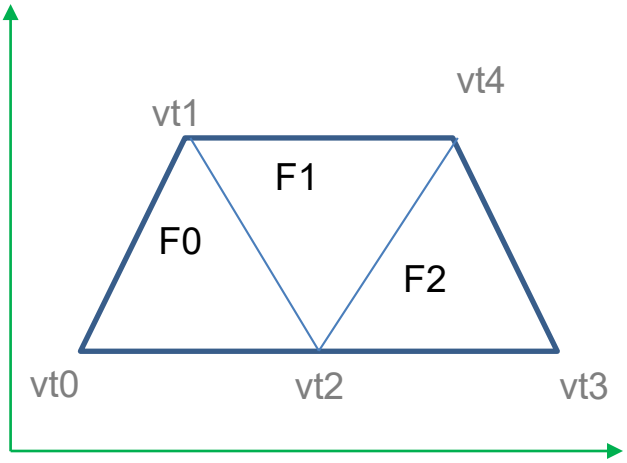
Euler's formula:
V−E+F=1 // for planar graphs

V−E+F=2(1−g)

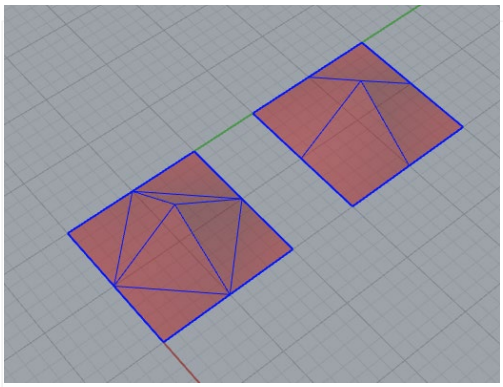
V = Vertices
E = Edges
F = Faces
G = Genus (# holes)

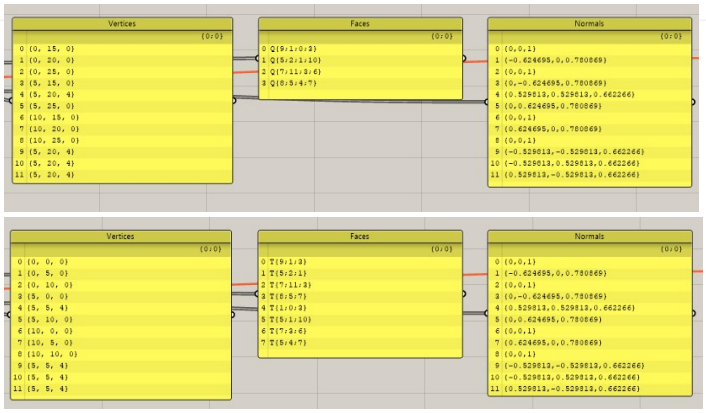
5 – 7 + 3 = 1





F0: V0, V2, V1
F1: V1, V2, V4
F2: V2, V3, V4





mesh01.obj

C: > Users > denni > iCloudDrive > _ACADEMIC > _COURSES > Design Programming II 2023 > designProgramming2Course > CourseNotes

```
# Rhino

mtllib mesh01.mtl
usemtl diffuse_0
v 0 0 0
v 0 5 0
v 0 10 0
v 5 0 0
v 5 5 4
v 5 10 0
v 10 0 0
v 10 5 0
v 10 10 0
v 5 5 4
v 5 5 4
v 5 5 4
vt 0 0
vt 0 0.5
vt 0 1
vt 0.5 0
vt 0.5 0.5
vt 0.5 1
vt 1 0
vt 1 0.5
vt 1 1
vt 0.5 0.5
vt 0.5 0.5
vt 0.5 0.5
vn 0 0 1
vn -0.6246950626373291 0 0.78086882829666138
vn 0 0 1
vn 0 -0.6246950626373291 0.78086882829666138
vn 0.52981293201446533 0.52981293201446533 0.66226619482040405
vn 0 0.6246950626373291 0.78086882829666138
vn 0 0 1
vn 0.6246950626373291 0 0.78086882829666138
vn 0 0 1
vn -0.52981293201446533 -0.52981293201446533 0.66226619482040405
vn -0.52981293201446533 0.52981293201446533 0.66226619482040405
vn 0.52981293201446533 -0.52981293201446533 0.66226619482040405
f 10/10/10 2/2/2 1/1/1 4/4/4
f 6/6/6 3/3/3 2/2/2 11/11/11
f 8/8/8 12/12/12 4/4/4 7/7/7
f 9/9/9 6/6/6 5/5/5 8/8/8

```

mesh01.obj | mesh01triangles.obj

C: > Users > denni > iCloudDrive > _ACADEMIC > _COURSES > Design Programming II 2023 > designProgramming2Course > CourseNotes > S

```
# Rhino

mtllib mesh01triangles.mtl
usemtl diffuse_0
v 0 0 0
v 0 5 0
v 0 10 0
v 5 0 0
v 5 5 4
v 5 10 0
v 10 0 0
v 10 5 0
v 10 10 0
v 5 5 4
v 5 5 4
v 5 5 4
vt 0 0
vt 0 0.5
vt 0 1
vt 0.5 0
vt 0.5 0.5
vt 0.5 1
vt 1 0
vt 1 0.5
vt 1 1
vt 0.5 0.5
vt 0.5 0.5
vt 0.5 0.5
vn 0 0 1
vn -0.6246950626373291 0 0.78086882829666138
vn 0 0 1
vn 0 -0.6246950626373291 0.78086882829666138
vn 0.52981293201446533 0.52981293201446533 0.66226619482040405
vn 0 0.6246950626373291 0.78086882829666138
vn 0 0 1
vn 0.6246950626373291 0 0.78086882829666138
vn 0 0 1
vn -0.52981293201446533 -0.52981293201446533 0.66226619482040405
vn -0.52981293201446533 0.52981293201446533 0.66226619482040405
vn 0.52981293201446533 -0.52981293201446533 0.66226619482040405
f 10/10/10 2/2/2 4/4/4
f 6/6/6 3/3/3 2/2/2
f 8/8/8 12/12/12 4/4/4
f 9/9/9 6/6/6 8/8/8
f 2/2/2 1/1/1 4/4/4
f 6/6/6 2/2/2 11/11/11
f 8/8/8 4/4/4 7/7/7
f 6/6/6 5/5/5 8/8/8

```

3_3_3: Meshes

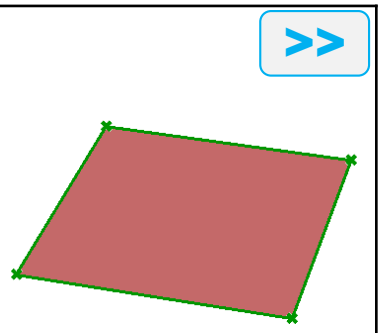
Meshes represent a geometry class that is defined by faces and vertices. The mesh data structure basically includes a list of vertex locations, faces that describe vertices connections and normal of vertices and faces. More specifically, the geometry lists of a mesh class include the following.

Mesh geometry lists	Description
Vertices	Of type MeshVertexList - includes a list of vertex locations type Point3f.
Normals	Of type MeshVertexNormalList - includes a list of normals of type Vector3f.
Faces	Of type MeshFaceList - includes a list of normals of type MeshFace.
FaceNormals	Of type "MeshFaceNormalList" - includes a list of normals of type Vector3f.

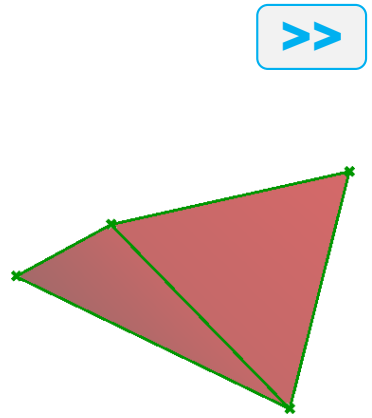
Create mesh objects:

You can create a mesh from scratch by specifying vertex locations, faces and compute the normal as in the following examples.

```
//Create a simple rectangular mesh that has 1 face
//Create a new instance of a mesh
Rhino.Geometry.Mesh mesh = new Rhino.Geometry.Mesh();

//Add mesh vertices
mesh.Vertices.Add(0.0, 0.0, 0.0); //0
mesh.Vertices.Add(5.0, 0.0, 0.0); //1
mesh.Vertices.Add(5.0, 5.0, 0.0); //2
mesh.Vertices.Add(0.0, 5.0, 0.0); //3

//Add mesh faces
mesh.Faces.AddFace(0, 1, 2, 3);

//Compute mesh normals
mesh.Normals.ComputeNormals();

//Generate any additional mesh data
mesh.Compact();
```




```
//Create simple triangular mesh that has 2 faces
//Create a new instance of a mesh
Rhino.Geometry.Mesh mesh = new Rhino.Geometry.Mesh();

//Add mesh vertices
mesh.Vertices.Add(0.0, 0.0, 0.0); //0
mesh.Vertices.Add(5.0, 0.0, 2.0); //1
mesh.Vertices.Add(5.0, 5.0, 0.0); //2
mesh.Vertices.Add(0.0, 5.0, 2.0); //3

//Add mesh faces
mesh.Faces.AddFace(0, 1, 2);
mesh.Faces.AddFace(0, 2, 3);

//Compute mesh normals
mesh.Normals.ComputeNormals();

//Generate any additional mesh data
mesh.Compact();
```



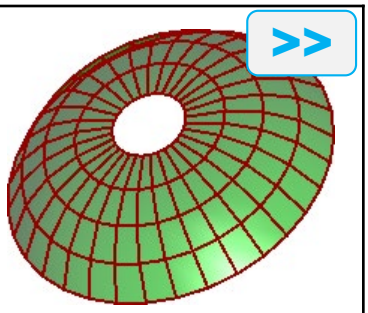
Creating meshes

The Mesh class includes many ***CreateFrom*** static methods to create a new mesh from various other geometry. Here is the list with description as it appears in the ***RhinoCommon*** help.

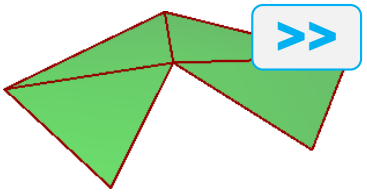
CreateFromBox(BoundingBox, Int32, Int32, Int32)	Constructs new mesh that matches a bounding box.
CreateFromBox(Box, Int32, Int32, Int32)	Constructs new mesh that matches an aligned box.
CreateFromBox(IEnumerable<Point3d>, Int32, Int32, Int32)	Constructs new mesh from 8 corner points.
CreateFromBrep(Brep)	Constructs a mesh from a brep.
CreateFromBrep(Brep, MeshingParameters)	Constructs a mesh from a brep.
CreateFromClosedPolyline	Attempts to create a Mesh that is a triangulation of a closed polyline.
CreateFromCone(Cone, Int32, Int32)	Constructs a solid mesh cone.
CreateFromCone(Cone, Int32, Int32, Boolean)	Constructs a mesh cone.
CreateFromCurvePipe	Constructs a new mesh pipe from a curve.
CreateFromCylinder	Constructs a mesh cylinder
CreateFromLines	Creates a mesh by analizing the edge structure. Input lines could be from the extraction of edges from an original mesh.
CreateFromPlanarBoundary(Curve, MeshingParameters)	Do not use this overload. Use version that takes a tolerance parameter instead.
CreateFromPlanarBoundary(Curve, MeshingParameters, Double)	Attempts to construct a mesh from a closed planar curve.RhinoMakePlanarMeshes
CreateFromPlane	Constructs a planar mesh grid.
CreateFromSphere	Constructs a mesh sphere.
CreateFromSurface(Surface)	Constructs a mesh from a surface
CreateFromSurface(Surface, MeshingParameters)	Constructs a mesh from a surface
CreateFromTessellation	Attempts to create a mesh that is a triangulation of a list of points, projected on a plane, including its holes and fixed edges.

Here are a couple examples to show how to create a mesh from a brep and a closed polyline.

```
//Set mesh parameters to default
var mp = MeshingParameters.Default;

//Create meshes from brep
var newMesh = Mesh.CreateFromBrep(brep, mp);
```




```
//Create a new mesh from polyline
var newMesh = Mesh.CreateFromClosedPolyline(pline);
```



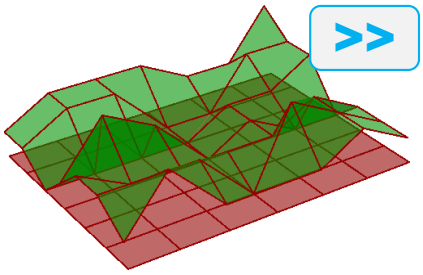
Navigate mesh geometry and topology:

You can navigate mesh data using ***Vertices*** and ***Faces*** properties. The list of vertices and faces are stored in a collection class or type with added functionality to manage the list efficiently. ***Vertices*** list for example, is of type ***MeshVertexList***. So if you need to change the locations of mesh vertices, then you need to get a copy of each vertex, change the coordinates, then reassign in the vertices list. You can also use the ***Set*** methods inside the ***MeshVertexList*** class to change vertex locations.

The following example shows how to randomly change the ***Z*** coordinate of a mesh using two different approaches. Notice that mesh vertices use a ***Point3f*** and not ***Point3d*** type because mesh vertex locations are stored as a single precision floating point.

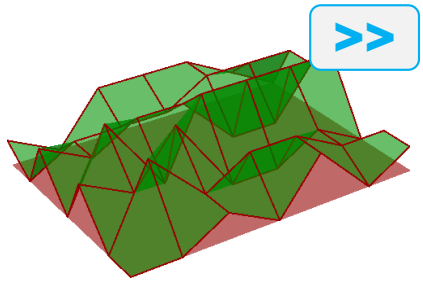
Change the location of mesh vertices by assigning the new value to the ***Vertices'*** list

```
//Change mesh vertex location
int index = 0;
Random rand = new Random();

foreach (Point3f loc in mesh.Vertices) {
    Point3f newLoc = loc;
    newLoc.Z = rand.Next(10) / 3;
    mesh.Vertices[index] = newLoc;
    index = index + 1;
}
```



Change the location of mesh vertices using the ***SetVertix*** method

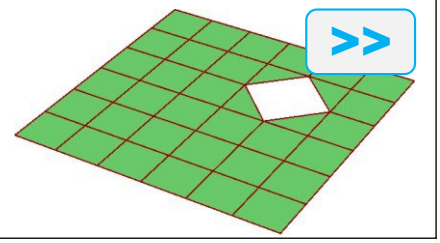
```
Mesh mesh = … // from input

//Create a new instance of random generator
Random rand = new Random();
for (int i = 0; i <= mesh.Vertices.Count - 1; i++){
    //Get vertex
    Point3f loc = mesh.Vertices[i];
    loc.Z = rand.Next(10) / 3;
    //Assign new location
    mesh.Vertices.SetVertex(i, loc);
}
```



Here is an example that deletes a mesh vertex and all surrounding faces

Delete mesh faces randomly

```
Mesh mesh = … // from input
int index = 32;
//Remove a mesh vertex (make sure index falls within range)
if (index >= 0 & i < mesh.Vertices.Count)
{
    mesh.Vertices.Remove(index, true);
}
```

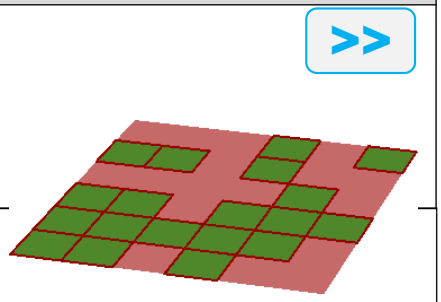


Navigate mesh geometry and topology

You can also manage the ***Faces*** list of a mesh. Here is an example that deletes about half the faces randomly from a given mesh.

Delete mesh faces randomly

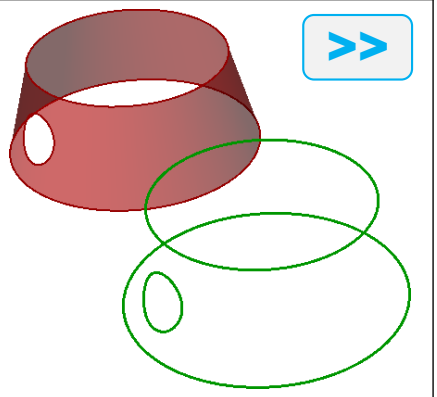
```
List<int> faceIndices = new List<int>();
int count = mesh.Faces.Count;

//Create a new instance of random generator
Random rand = new Random();

for (int i = 0; i <= count - 1; i += 2){
    int index = rand.Next(count);
    faceIndices.Add(index);
}

//delete duplicate indexes
List<int> distinctIndices = faceIndices.Distinct().ToList();

//delete faces
mesh.Faces.DeleteFaces(distinctIndices);
```



Meshes keep track of the connectivity of the different parts of the mesh. If you need to navigate related faces, edges or vertices, then this is done using the mesh topology. The following example shows how to extract the outline of a mesh using the mesh topology.

Mesh topology example: extract the outline of a mesh

```
// Get the mesh's topology
Rhino.Geometry.Collections.MeshTopologyEdgeList  meshEdges =
mesh.TopologyEdges;

List<Line> lines = new List<Line>();

// Find all of the mesh edges that have only a single mesh face
for (int i = 0; i <= meshEdges.Count - 1; i++) {
   int numOfFaces = meshEdges.GetConnectedFaces(i).Length;

   if ((numOfFaces  == 1)) {
      Line line = meshEdges.EdgeLine(i);
      lines.Add(line);
   }
}
```

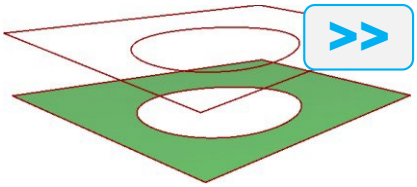


Mesh methods

Once a new mesh object is created, you can edit and extract data out of that mesh object. The following example extracts naked edges out of some input mesh.

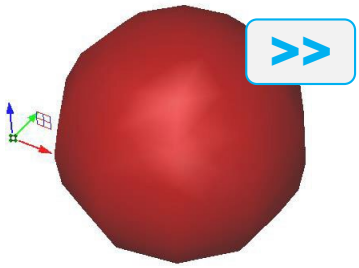
Extract the outline of a mesh

```
Mesh mesh = … // from input

//Declare an array of polylines
Polyline[ ] naked_edges = { };

//Create a new mesh from polyline
nakedEdges = mesh.GetNakedEdges();
```



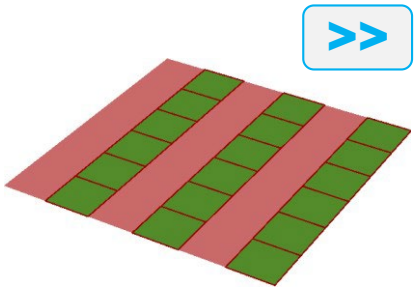
Test if a given point is inside a closed mesh

```
Mesh mesh = … // from input
Point3d pt = … // from input

//Test if the point is inside the mesh
mesh.IsPointInside(pt, tolerance, true);
```



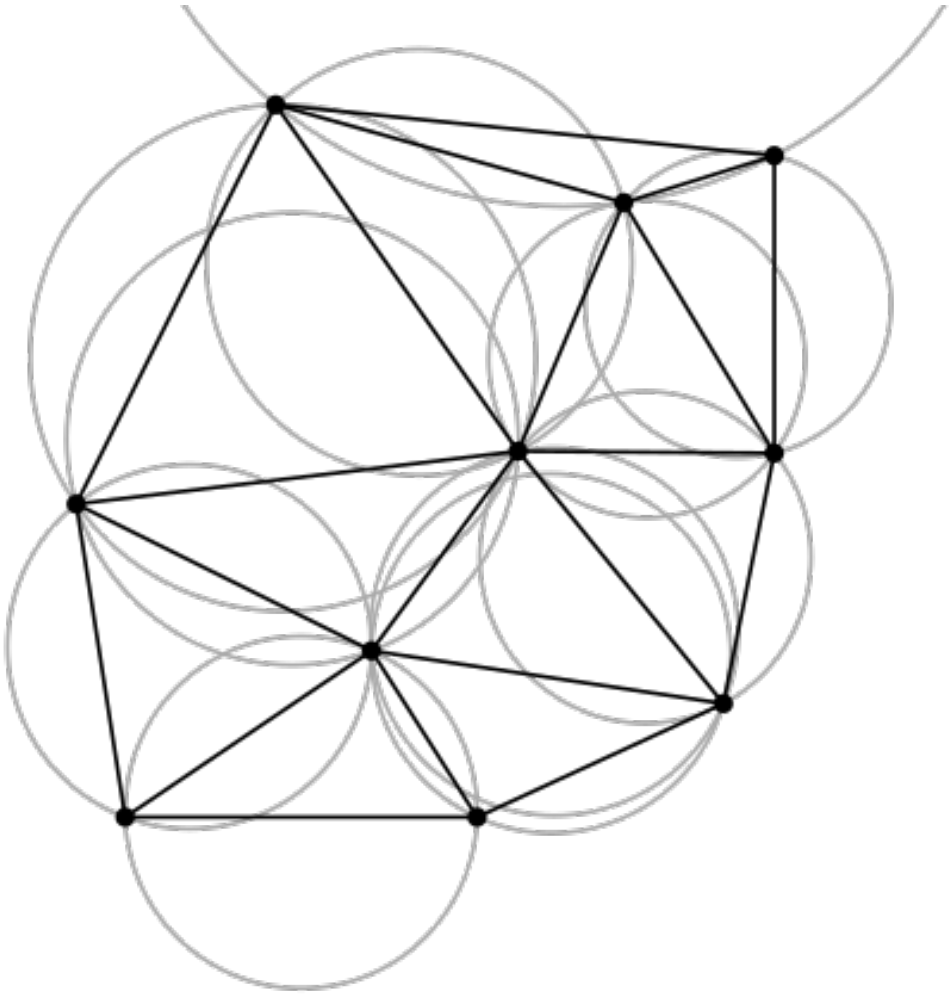
Delete every other mesh face, then split disjoint meshes

```
Mesh mesh = … // from input
List<int> faceIndeces = new List<int>();

//Loop through faces and delete every other face
for (int i = 0; i <= mesh.Faces.Count - 1; i += 2) {
    faceIndeces.Add(i);
}
//delete faces
mesh.Faces.DeleteFaces(faceIndeces);
//Split disjoint meshes
Mesh[ ] meshArray = mesh.SplitDisjointPieces();
```

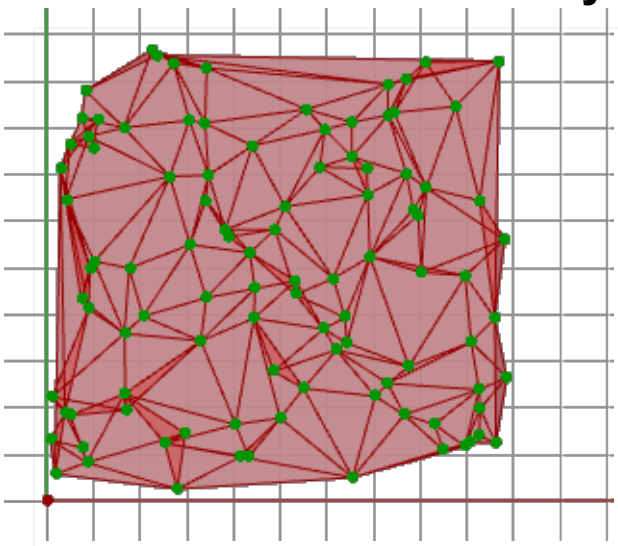


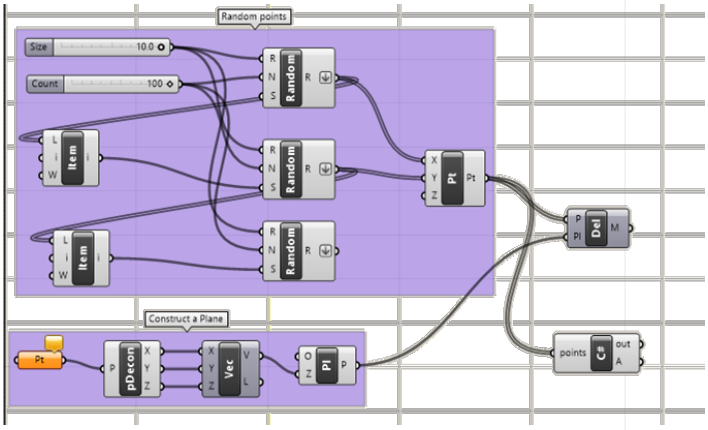
Delaunay Triangulation

A Delaunay Triangulation in a plan is a mesh such that no vertex is contained in the circle circumscribed by the 2 points of any given triangle.



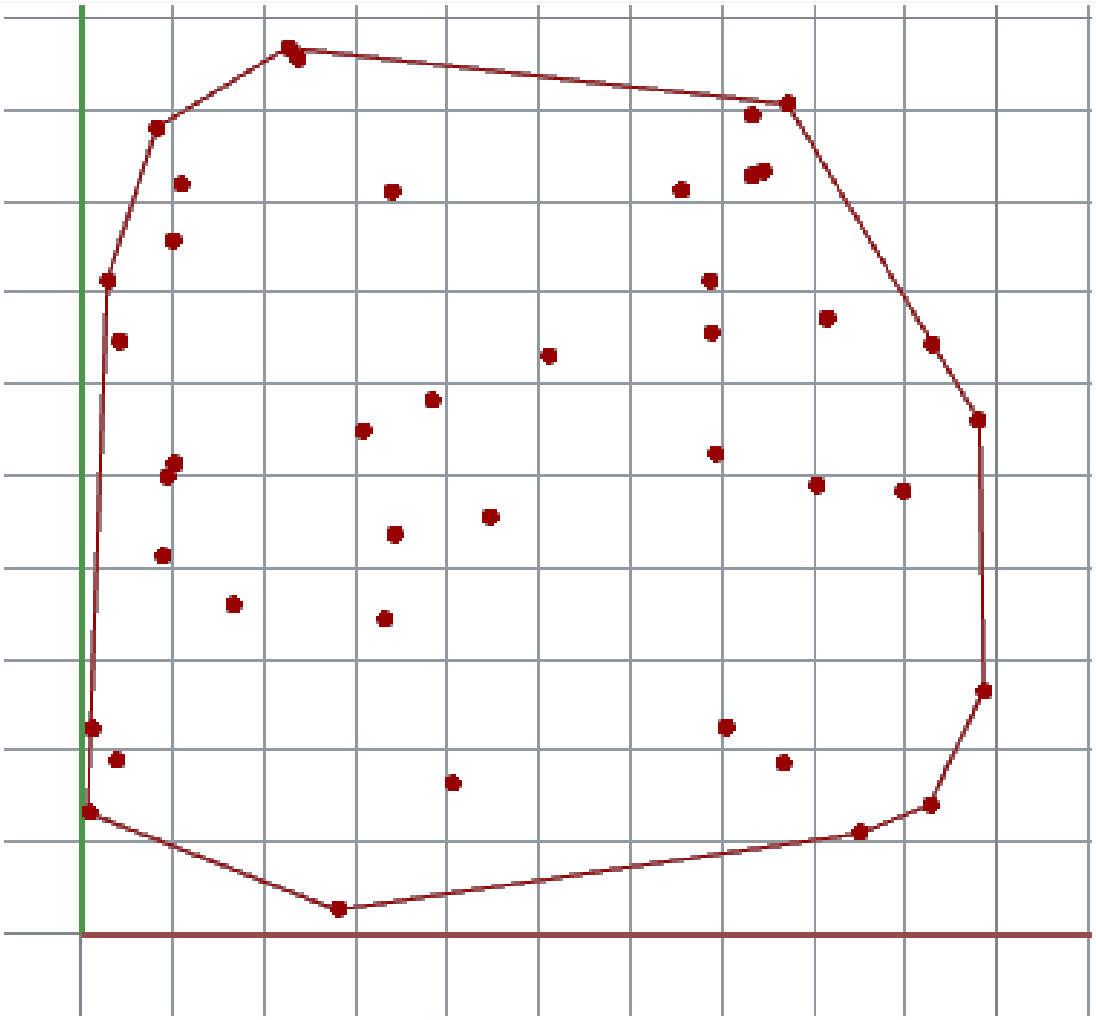
Delaunay Triangulation



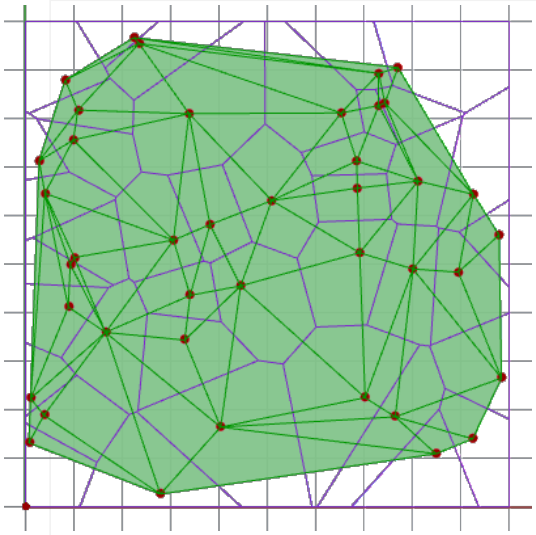


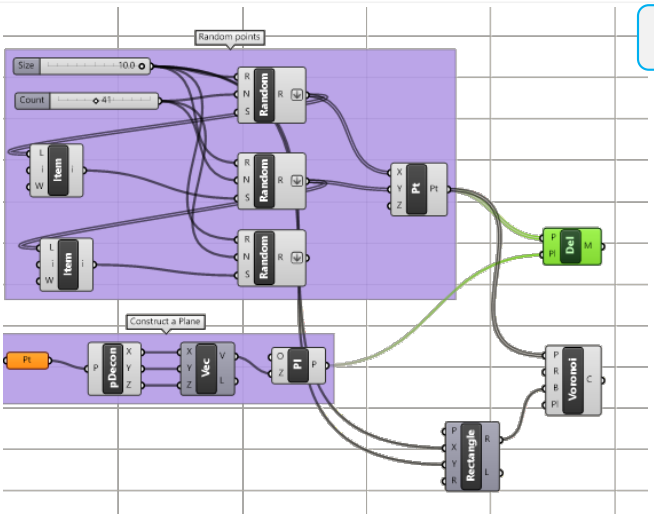

```
private void RunScript(List<Point3d> points, ref object A)
{


  //convert point3d to node2
  //grasshopper requres that nodes are saved within a Node2List for Delaunay
  var nodes = new Grasshopper.Kernel.Geometry.Node2List();
  for (int i = 0; i < points.Count; i++)
  {
    //notice how we only read in the X and Y coordinates
    //  this is why points should be mapped onto the XY plane
    nodes.Append(new Grasshopper.Kernel.Geometry.Node2(points[i].X, points[i].Y));
  }

  //solve Delaunay
  var delMesh = new Mesh();
  var faces = new List<Grasshopper.Kernel.Geometry.Delaunay.Face>();

  faces = Grasshopper.Kernel.Geometry.Delaunay.Solver.Solve_Faces(nodes, 1);

  //output
  A = Grasshopper.Kernel.Geometry.Delaunay.Solver.Solve_Mesh(nodes, 1, ref faces);
}
```

Convex Hull



Vornoi




Random points
Size
10.0
Count
41
Random
Item
Pt
Del
Construct a Plane
pDecon
Vec
Pl
Voronoi
Rectangle