Useful Links:

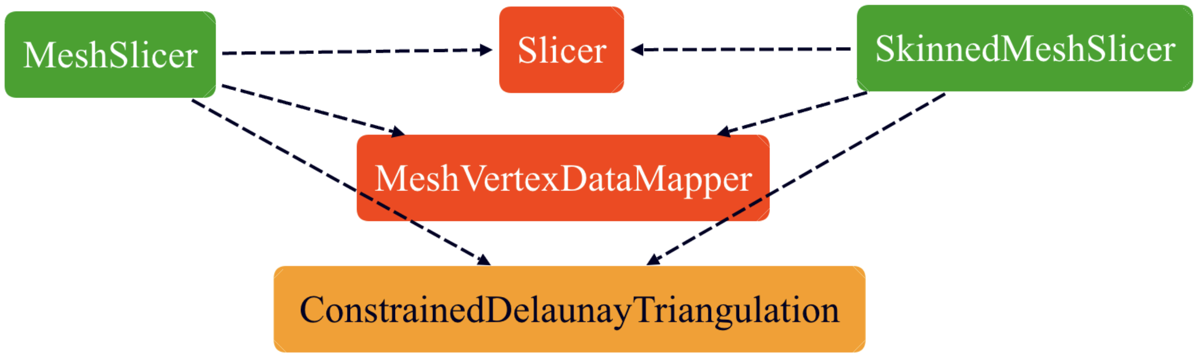
Mesh Slicer Free on GitHub: https://github.com/hanzemeng/MeshSlicerFree
Mesh Slicer Free on Unity Asset Store: https://assetstore.unity.com/packages/slug/283149
Mesh Slicer Free's online documentation:
https://docs.google.com/document/d/1Muqt7BsIIzq-GR4BRaghu5qdzuZcEfcTd27qJ-Gr0GU/edit?usp=sharing

Purpose:

Mesh Slicer Free slices meshes into exactly two meshes. Mesh Slicer Free can handle some pretty complex meshes, such as those with holes. In general, as long as the mesh forms a closed volume and every triangle intersects other triangles only at the vertices, Mesh Slicer Free will slice correctly and, relatively, efficiently.

Roadmap:

The picture below describes the relationship between the major classes in Mesh Slicer Free.



The users are intended to directly use the green classes. The users can also directly use the orange class, although this is unintended. The red classes are intended for internal use only and are not documented.

The Mesh Slicer Class:

A MeshSlicer object can be created by:

```
MeshSlicer meshSlicer = new MeshSlicer();
```

A slice can be performed by:

```
(GameObject, GameObject) res =
meshSlicer.Slice(targetGameObject, slicePlane, intersectionMaterial);

/*
```

```
res contains the two sliced game objects. res == (null, null) if the
slice plane does not intersect the targetGameObject.

targetGameObject is the game object to be sliced. targetGameObject is
assumed to have a MeshFilter and MeshRenderer attached.
targetGameObject is duplicated twice during the slice, so it is
recommended to remove irrelevant scripts before slicing
targetGameObject.

slicePlane is of type (Vector3,Vector3,Vector3). slicePlane contains
the 3 points (in world position) that form the slice plane. The 3
points must not be collinear or the result is undefined.

intersectionMaterial is the material to fill the intersection. If
intersectionMaterial == null, then the intersection will be filled
using the last material in the MeshRenderer.
*/
```

A slice can be performed asynchronously by:

```
(GameObject, GameObject) res = await
meshSlicer.SliceAsync(targetGameObject,slicePlane,intersectionMaterial);

// parameters are the same as the synchronous version
```

A slice can be performed at a lower level without creating gameobjects:

```
(Mesh, Mesh) res = meshSlicer.Slice(slicePlane, targetMesh,
targetTransform, createSubmeshForIntersection);

/*
res contains the two sliced meshes. res == (null, null) if the slice
plane does not intersect the targetMesh.

slicePlane is of type (Vector3,Vector3,Vector3). slicePlane contains
the 3 points (in world position) that form the slice plane. The 3
points must not be collinear or the result is undefined.

targetMesh is the mesh to be sliced.
```

```
targetTransform is the transform that calculates the world positions
of targetMesh's vertices. Usually targetTransform is the transform of
the MeshFilter that contains targetMesh.

If createSubmeshForIntersection is true, then each of the sliced mesh
will have a new submesh corresponding to the intersection face. If
false, then the intersection face is combined with the last submesh
from targetMesh.
*/
```

A slice can be performed asynchronously at a lower level without creating gameobjects:

```
(Mesh, Mesh) res = await meshSlicer.SliceAsync(slicePlane, targetMesh,
targetTransform, createSubmeshForIntersection);

// parameters are the same as the synchronous version
```

The MeshSlicer class has a constant member:

```
public const int MAX_SUBMESH_COUNT = 16;
/*
Must be strictly less than the number of submeshes in the target game
object, even if not creating a new submesh for the intersection face.
Value defined in MeshSlicer.cs.
*/
```

The Skinned Mesh Slicer Class:

A SkinnedMeshSlicer object can be created by:

```
SkinnedMeshSlicer skinnedMeshSlicer = new SkinnedMeshSlicer();
```

A slice can be performed by:

```
(GameObject, GameObject) res = skinnedMeshSlicer.Slice
(targetGameObject, skinnedMeshRendererIndex, rootIndex,
 slicePlane, intersectionMaterial);

/*
res contains the two sliced game objects. res == (null, null) if the
slice plane does not intersect the targetGameObject.
```

```
targetGameObject is the direct parent of the game object that contains
the SkinnedMeshRender, and the game object that contains the bones.

skinnedMeshRendererIndex is the child index of the game object that
contains the SkinnedMeshRender.

rootIndex is the child index of the game object that contains the bones
(this game object should have dozens of nested children).

slicePlane is of type (Vector3,Vector3,Vector3). slicePlane contains
the 3 points (in world position) that form the slice plane. The 3
points must not be collinear or the result is undefined.

intersectionMaterial is the material to fill the intersection. If
intersectionMaterial == null, then the intersection will be filled
using the last material in the MeshRenderer.
*/
```

A slice can be performed asynchronously by:

```
(GameObject, GameObject) res = await skinnedMeshSlicer.Slice
(targetGameObject, skinnedMeshRendererIndex, rootIndex,
 slicePlane, intersectionMaterial);

// parameters are the same as the synchronous version
```

The SkinnedMeshSlicer class has a constant member:

```
public const int MAX_SUBMESH_COUNT = 16;
/*
Must be strictly less than the number of submeshes in the target game
object, even if not creating a new submesh for the intersection face.
Value defined in SkinnedMeshSlicer.cs.
*/
```

The Constrained Delaunay Triangulation Class:

A ConstrainedDelaunayTriangulation object can be created by:

```
ConstrainedDelaunayTriangulation cdt = new
```

```
ConstrainedDelaunayTriangulation();
```

A constrained Delaunay triangulation can be performed by:

```
List<int> res = cdt.Triangulate(vertices, edges);

/*
res is the list of triangles produced by triangulation. Every 3
elements describe each of the triangles (if i%3==0, then res[i+0],
res[i+1], res[i+2] is a triangle).

vertices is a list of Vector2 that describes the position of each
vertex.

edges is a list of int that describes the edges that should be
preserved in triangulation. Every 2 elements describe each of the
edges (if i%2==0, then edges[i+0], edges[i+1], is an edge).
If edges is an empty list or is null, then Delaunay triangulation is
performed.
*/
```

The result of constrained Delaunay triangulation can be stored in a user-supplied list:

```
cdt.Triangulate(vertices, edges, res);
/*
res is cleared and then has the triangulation result stored in it.

All other parameters are the same.
*/
```

The ConstrainedDelaunayTriangulation class has a constant member:

```
public const float INPUT_VERTICES_RANGE = 10000f;
/*
Due to the algorithm used in triangulation, every vertex in the
vertices must be strictly inside a large triangle. The large triangle
is defined by:
(-INPUT_VERTICES_RANGE, -INPUT_VERTICES_RANGE)
( INPUT_VERTICES_RANGE, -INPUT_VERTICES_RANGE)
( 0f,                    INPUT_VERTICES_RANGE)
```

```
The value is defined in ConstrainedDelaunayTriangulation/Public.cs.
*/
```

Contact Author:

hanzemeng2001518@gmail.com

License:

References:

- S.W. Sloan, A fast algorithm for generating constrained delaunay triangulations, Computers & Structures, Volume 47, Issue 3, 1993, Pages 441-450, ISSN 0045-7949, https://doi.org/10.1016/0045-7949(93)90239-A.
- Jonathan Richard Shewchuk, Lecture Notes on Delaunay Mesh Generation, 2012, https://people.eecs.berkeley.edu/%7Ejrs/meshpapers/delnotes.pdf.
- Richard Shewchuk, J. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates . Discrete Comput Geom 18, 305–363 (1997). https://doi.org/10.1007/PL00009321.
- Files from https://github.com/govert/RobustGeometry.NET were taken directly.
- Many solutions from Stack Overflow, Unity Forum, Microsoft Learn, Unity Documentation are referenced.