DOCUMENTATION FOR MESH SLICER FREE VERSION 2.1.0

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

Installation Note:

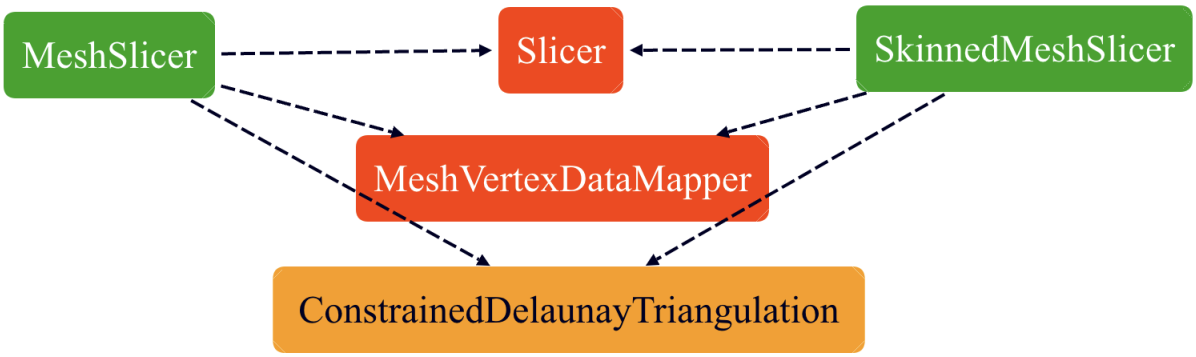
If another version of Mesh Slicer Free is already installed, please delete it first and then install the current version of Mesh Slicer Free.

Purpose:

Mesh Slicer Free slices meshes into exactly two meshes. Mesh Slicer Free can slice any mesh (concave, not closed, has holes, has interesting faces, has collinear triangles). Depending on the complexity of the input mesh, the performance varies widely. In general, meshes with less than 1000 vertices can be sliced within 100ms.

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 game objects:

```
(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 game objects:

```
(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.
*/
```

Input Mesh Requirements

Ideal meshes have the following properties:

1. All edges are manifold (every edge shares exactly two triangles).
2. None of the triangles are collinear or nearly collinear.
3. The triangles form one or multiple closed volumes.
4. The triangles do not intersect each other.

From my observation, most meshes satisfy none of the properties.

Applicability and Performance:

If an input mesh does not meet every input mesh requirement, Mesh Slicer Free may not slice properly.

Below lists options the users can adjust to expand the applicability of Mesh Slicer Free at the cost of performance.

```
#define CHECK_VERTEX_ON_EDGE
// Defined in ConstrainedDelaunayTriangulation/SegmentRecovery.cs.
// Define this symbol if previously crashed or not terminating.
// Do not define this symbol if slicing only ideal meshes.
// Huge time sink if this symbol is defined.
```

```
#define USE_WINDING_NUMBER
// Defined in ConstrainedDelaunayTriangulation/DomainCalculation.cs.
// Define this symbol if the intersection looks incorrect.
// Do not define this symbol if the intersection already looks good.
// Huge time sink if this symbol is defined.
```

```
#define USE_32_BIT_INDEX_FORMAT
// Defined in MeshVertexDataMapper.cs.
// Define this symbol if one of the sliced mesh contains more than
// 65536 vertices.
```

Note that **all of the symbols are defined by default**. The users may wish to comment out some of them to increase performance.

Miscellaneous:

1. To slice a mesh, Mesh Slicer Free needs to read the mesh data; this permission can be granted by checking Read/Write when importing the mesh.
2. The game object to be sliced should have a positive scale in every axis.

3. The hardware performing floating-point operations must conform with IEEE 754.
4. Thank you for using Mesh Slicer Free and reading its documentation. Consider reviewing it on the Unity Asset Store.

License:

Mesh Slicer Free is distributed under the terms of GNU GPL. The license can be found at: http://www.gnu.org/licenses/.

Contact Author:

hanzemeng2001518@gmail.com

Possible Improvements:

What is a reasonable scheme to assign vertex information (say uv coordinate) to vertices on the intersection face?
The intersection face may contain intersecting edges. By locating all of the edge intersections and including them in the triangulation, the slice result may look better.
How are generalized winding numbers calculated? Are they better at approximating the intersection than winding numbers?

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.
- David Sinclair, S-hull: a fast radial sweep-hull routine for Delaunay triangulation. 2016, https://doi.org/10.48550/arXiv.1604.01428.
- Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. ACM Trans. Graph. 32, 4, Article 33 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461916
- Files from https://github.com/govert/RobustGeometry.NET were taken directly.
- Files from https://github.com/AyazDyshin/Red-Black-Tree/tree/main were taken and modified.
- Many solutions from Stack Overflow, Unity Forum, Microsoft Learn, Unity Documentation are referenced.