DOCUMENTATION FOR TETRAHEDRALIZER VERSION 1.0.0

Useful Links:

Tetrahedralizer on GitHub: https://github.com/hanzemeng/Tetrahedralizer
Tetrahedralizer on Unity Asset Store:
Tetrahedralizer's online documentation:
https://docs.google.com/document/d/1TmDsCK4SiXGq7Y7xVC6l-rAW5G5WxgNKjIlzR9Algpc/edit?usp=sharing

Purpose:

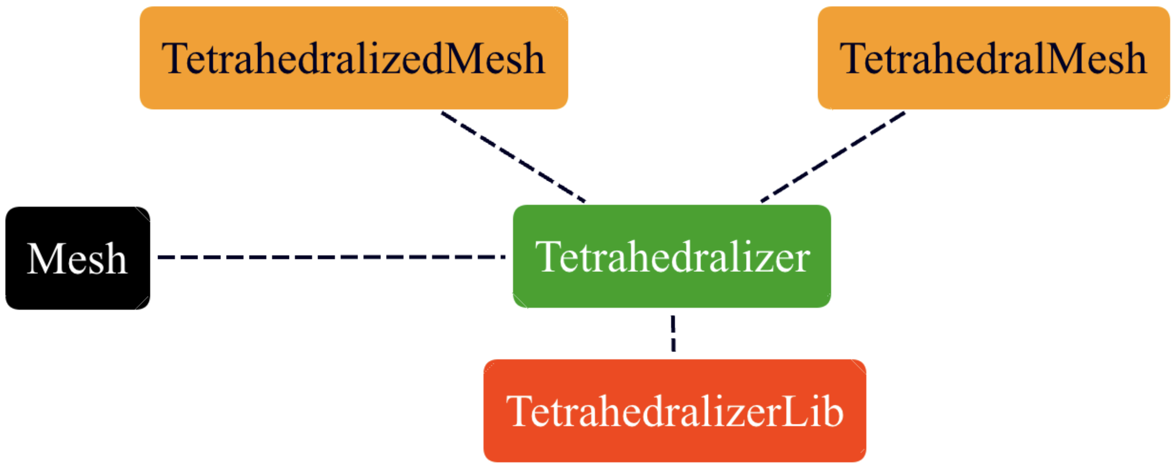
Tetrahedralizer converts triangle meshes into tetrahedral meshes. As the name suggests, a tetrahedral mesh represents a 3D model using tetrahedrons, which are more suitable for geometric computation than using triangles. Tetrahedralizer is able to convert arbitrarily complex triangle meshes that may include intersecting triangles, degenerate triangles, and non-manifold edges.

Tetrahedralizer ignores fields specific to skinned meshes and treats them as regular meshes.

Tetrahedralizer is a rather low-level package. The users should have prior experience working with meshes to make the best use of Tetrahedralizer.
I do my best to explain everything Tetrahedralizer has to offer, but some parts may still be confusing.

Roadmap:

The picture below describes the relationship between the major classes in this package.



The black class, Mesh, is from Unity Engine. It is the input to Tetrahedralizer.

The oranges classes are scriptable objects. They store results computed by Tetrahedralizer.

The green class, Tetrahedralizer, is the core of this package. It implements the algorithm used to convert Mesh into TetrahedralizedMesh and TetrahedralizedMesh to TetrahedralMesh.

The red "class", TetrahedralizerLib, is a dynamic library written in C++. It is used by Tetrahedralizer.

The expected workflow is that the user first create an instance of Tetrahedralizer,
then supply the mesh to convert to the tetrahedralizer and it will output an instance of TetrahedralizedMesh,
then supply the tetrahedralizedMesh to the tetrahedralizer and it will output an instance of TetrahedralMesh.
The tetrahedralMesh now contains the converted tetrahedral mesh. The rest is up to the users.

The rest of the document goes over the classes in more detail.

The Mesh Class:

Please refer to Unity Documentation.

The Tetrahedralized Mesh Class:

This class is a scriptable object that stores information needed to tetrahedralize a mesh.
The user may wish to skip this section if they are not planning to modify the package itself.

Below describes the fields of this class:

```
public Mesh originalMesh;

public List<ListInt32> originalVerticesMappings;
public List<NineInt32> newVerticesMappings;
public List<Int32> tetrahedrons;

/*
originalMesh is a reference to the original mesh that is being
tetrahedralized.


originalVerticesMappings stores indices of vertices from originalMesh
that share the same position.
Note that ListInt32 is a wrapper class for List<Int32>.

For example, say originalMesh's 2,39,444 vertices are in the same
position, then originalVerticesMappings[2] contains a ListInt32
object with its list containing {2,39,444}.

We can think of originalVerticesMappings as a map that only keeps the
unique positions in originalMesh.vertices.


newVerticesMappings stores new vertices generated by the algorithm.
The new vertices are represented using indices from
originalVerticesMappings (not from originalMesh).
Note that NineInt32 is a struct that contains nine Int32.

If 0xFFFFFFFF == newVerticesMappings[i][5], this means the ith new
vertex is the intersection of a line and a plane.
The line is defined as
newVerticesMappings[i][0],newVerticesMappings[i][1].
The plane is defined as
newVerticesMappings[i][2],newVerticesMappings[i][3],newVerticesMappin
gs[i][4].

If 0xFFFFFFFF != newVerticesMappings[i][5], this means the ith new
```

```
vertex is the intersection of three planes.
The first plane is defined as
newVerticesMappings[i][0],newVerticesMappings[i][1],newVerticesMappin
gs[i][2].
The second plane is defined using the same pattern but starts at 3.
The third plane is defined using the same pattern but starts at 6.


tetrahedrons stores the tetrahedrons that represent the tetrahedral
mesh.
For example, let 0==i%4, then
tetrahedrons[i+0],tetrahedrons[i+1],
tetrahedrons[i+2],tetrahedrons[i+3]
are the 4 indices that represent a tetrahedron.

If tetrahedrons[i] < originalVerticesMappings.Count, then
tetrahedrons[i] corresponds to one of the vertices from originalMesh.

If tetrahedrons[i] >= originalVerticesMappings.Count, then
tetrahedrons[i] corresponds to
newVerticesMappings[tetrahedrons[i]-originalVerticesMappings.Count].
*/
```

The Tetrahedral Mesh Class:

This class is a scriptable object that stores the tetrahedral mesh.
The tetrahedral mesh is stored in one of two ways:
Way one is storing only the shape information; we know the position of every vertex, but nothing else (no uv, color, etc.).
Way two is storing position as well as every type of vertex data that the original mesh has.

A tetrahedralMesh behaves very differently depending on which storing scheme is used, so please pay close attention to what we are about to read.
Below describes the fields of this class:

```
public List<Vector3> vertices;
public List<Int32> tetrahedrons;

public Mesh mesh;
```

```
/*
When storing scheme one is used, vertices and tetrahedrons are
populated with data, mesh is null.

When storing scheme two is used, vertices and tetrahedrons are null,
mesh is populated with data.



vertices stores the position of each vertex.

Note that a regular mesh often has multiple vertices at the same
position (this is to calculate normal and tangent), this is not the
case for vertices.
vertices may still contain duplicate values due to floating-point
rounding, but we don’t intentionally create multiple copies of the
same vertex.


tetrahedrons stores indices that represent tetrahedrons.
For example, let 0==i%4, then
tetrahedrons[i+0],tetrahedrons[i+1],
tetrahedrons[i+2],tetrahedrons[i+3]
are the 4 indices that represent a tetrahedron.

The tetrahedrons are oriented using the left-hand rule.
That is, if you curl your left hand around tetrahedrons[i+0],
tetrahedrons[i+1], and tetrahedrons[i+2], then your thumb points
toward tetrahedrons[i+3].
Note that it is possible to have degenerate tetrahedrons (4 vertices
on the same plane).


mesh is a new mesh generated based on the original mesh.

mesh’s vertices are organized such that
i == mesh.triangles[i] for every i.
```

```
Because a tetrahedron has 4 triangle faces,
every 12 vertices correspond to a tetrahedron.
For example, let 0==i%12, then
mesh.vertices[i+0],mesh.vertices[i+1],mesh.vertices[i+2]
is the position of the first face of the i/12 tetrahedron.
Suppose the original mesh has uv1, then
mesh.uv1[i+3],mesh.uv1[i+4],mesh.uv1[i+5]
is the uv1 of the second face of the i/12 tetrahedron.

Pop quiz:
how to access the color of the fourth face of the i/12 tetrahedron?
highlight the line below to check the answer.


Every kind of vertex data from the original mesh is generated
except boneWeights.

Note that mesh is a Mesh, meaning every time we use the dot operator
to access its vertex data, the entire array is copied.
*/
```

The Tetrahedralizer Class:

This class converts Mesh into TetrahedralizedMesh and TetrahedralizedMesh to TetrahedralMesh.

A tetrahedralizer can be created by:

```
Tetrahedralizer tetrahedralizer = new Tetrahedralizer();
```

To convert a mesh into a tetrahedralizedMesh:

```
tetrahedralizer.MeshToTetrahedralizedMesh(Mesh mesh,
TetrahedralizedMesh tetrahedralizedMesh);

// mesh is the input.
// tetrahedralizedMesh is cleared and has the output stored in it.
```

To convert a tetrahedralizedMesh into a tetrahedralMesh:

```
tetrahedralizer.TetrahedralizedMeshToTetrahedralMesh(
TetrahedralizedMesh tetrahedralizedMesh,
TetrahedralMesh tetrahedralMesh);

// tetrahedralizedMesh is the input.
// tetrahedralMesh is cleared and has the output stored in it.

// contact the author if you have a shorter name for this function.
```

A tetrahedralizer has settings that can be adjusted:

```
tetrahedralizer.SetSettings(Tetrahedralizer.Settings settings);

// where settings is defined as:
settings = new Tetrahedralizer.Settings(bool remapVertexData, double
degenerateTetrahedronRatio);

/*
If remapVertexData is true, then the output tetrahedralMesh will have
a mesh with every kind of vertex data from the original mesh.
Otherwise, the output tetrahedralMesh only has the position for every
vertex.

degenerateTetrahedronRatio is the ratio to remove tetrahedrons that
are too small.
Let the average volume of each generated tetrahedron be v_a. Then if
a tetrahedron's volume is less than v_a * degenerateTetrahedronRatio,
that tetrahedron is removed.
Set degenerateTetrahedronRatio to 0 if one wishes to keep every
generated tetrahedron.
*/
```

The TetrahedralizerLib:

TetrahedralizerLib is a dynamic library written in C++. Two versions of the library are included in the package.
One version supports macOS for both Intel 64-bit and Apple silicon chips.
The other supports Windows for Intel 64-bit chips.

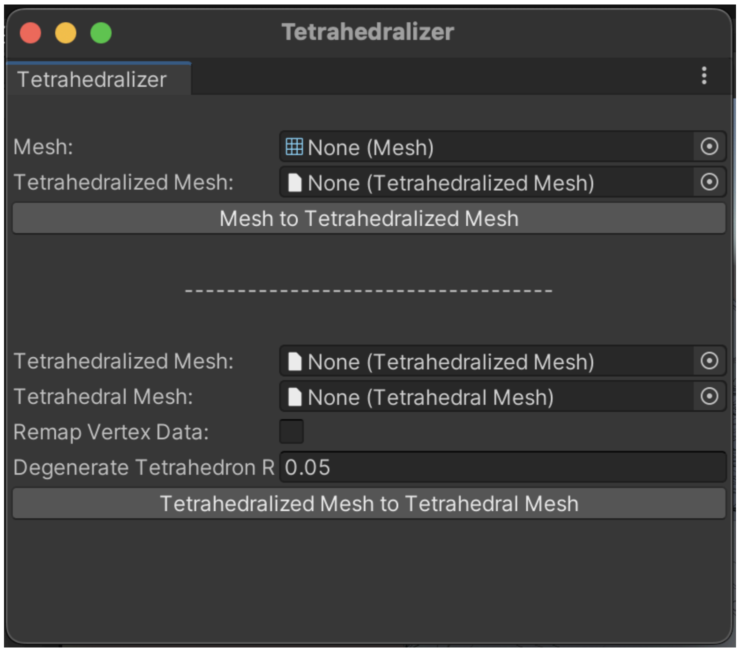
The source code is provided on GitHub. If one wish to compile the source code, here are the requirements:
Use C++20.

Use -ffp-model=strict, -ffp-contract=off, -frounding-math.
Optionally use -O2.

Unity Editor Modification:

The Unity Editor has been updated with the intention of simplifying development.

In the menu bar, select Window/Tetrahedralizer to open a window that looks like:



Mesh: None (Mesh)
Tetrahedralized Mesh: None (Tetrahedralized Mesh)
Mesh to Tetrahedralized Mesh

Tetrahedralized Mesh: None (Tetrahedralized Mesh)
Tetrahedral Mesh: None (Tetrahedral Mesh)
Remap Vertex Data: ☐
Degenerate Tetrahedron R 0.05
Tetrahedralized Mesh to Tetrahedral Mesh

The top part is about converting Mesh into TetrahedralizedMesh and
the bottom part is about converting TetrahedralizedMesh to TetrahedralMesh.

In the project window, right click and select
Create/ScriptableObjects/Tetrahedralizer/Tetrahedralized_Mesh to create a Tetrahedralized Mesh.
A Tetrahedral Mesh is created in the same way.

All editor scripts are in Scripts/Editor/.

Miscellaneous:

1. For tetrahedral meshes with every type of vertex data, only expect the position data to be correct. I have yet to find an algorithm that can properly generate other types of vertex data.
2. Tetrahedralizer's cover image is designed and created by artist michuudo. Please check him out on X and Bluesky.

3. Thank you for using Tetrahedralizer and reading its documentation. Consider reviewing it on the Unity Asset Store.

Author Information:

Email: hanzemeng2001518@gmail.com
LinkedIn: https://www.linkedin.com/in/han-zemeng-a227b3147/

References:

- Diazzi, L., & Attene, M. (2021). Convex Polyhedral Meshing for Robust Solid Modeling. ACM Transactions on Graphics (SIGGRAPH Asia 2021), 40(6).
- M. Attene. Indirect Predicates for Geometric Constructions. In Elsevier Computer-Aided Design (2020).
- Alec Jacobson, Ladislav Kavan, Olga Sorkine-Hornung. Robust Inside-Outside Segmentation using Generalized Winding Numbers. ACM Transaction on Graphics 32(4) [Proceedings of SIGGRAPH], 2013.
- Efficient Approximate Energy Minimization via Graph Cuts. Y. Boykov, O. Veksler, R.Zabih. IEEE TPAMI, 20(12):1222-1239, Nov 2001.
- What Energy Functions can be Minimized via Graph Cuts? V. Kolmogorov, R.Zabih. IEEE TPAMI, 26(2): 147-159, Feb 2004.
- An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. Y. Boykov, V. Kolmogorov. IEEE TPAMI, 26(9):1124-1137, Sep 2004.
- Many solutions from Stack Overflow, Unity Forum, Microsoft Learn, Unity Documentation are referenced.