Branching Modeling Application User's Guide

August 10, 2016

By

Dar-jen Chang
Computer Engineering and Computer Science
University of Louisville

E-mail: djchan01@louisville.edu

1 Introduction

The branching model (TriMesh_GUI) application is a Windows application, made with Unity 3D, whose main function is lets the users open tube and graph files to create triangular mesh 3D models of branching network structures defined in the input files. A screenshot of the branching modeling application is shown in Figure 1. The model shown in the display window in Figure 1 is a simulated bipolar retinal cell triangular mesh model created from the tube file, ret12_mod.tub, located in the application folder's directory FileTubes/Retinal Objs.

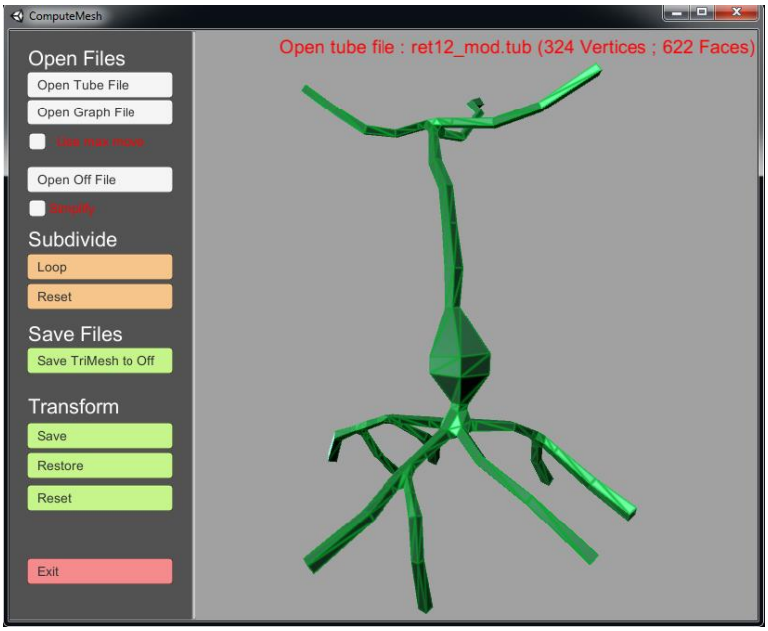


Figure 1. A Screenshot of Unity Windows Application for Branching Modeling

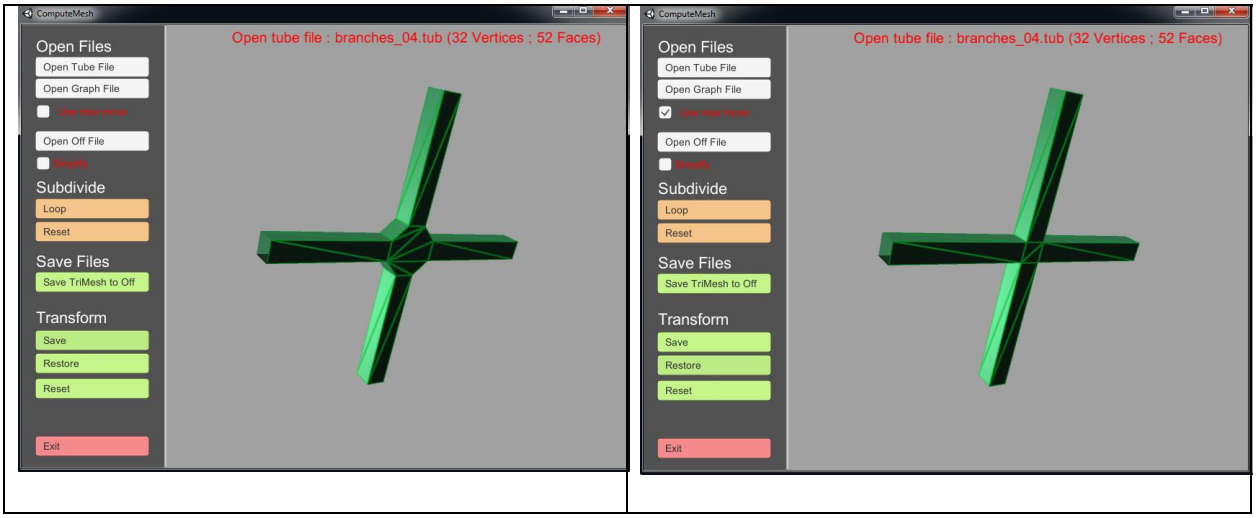
You can open the given tube, graph, or OFF files in the application folder to view the resulting triangular mesh models shown in the display window and save them in OFF files for further use. Furthermore you can prepare your own tube and graph files manually using a text editor from scratch, modify the given files, create new files procedurally using your algorithms, or make input files from your acquired data using image processing techniques, for example.

2 User Interface

The user interface is quite straightforward. Buttons and check boxes are grouped into these categories: Open Files, Subdivide, Save Files, and Transform located on the left margin of the window. There is an Exit button near the bottom of the window lets you exit (close) the application.

Open Files Group

The Open Files group allows you to open a tube file, graph file, or OFF file and check Use max move or Simplify checkbox. Using the Use max move checkbox you can control how the 3D construction method makes the triangular mesh around the center of the branch. Figure 2 illustrates the difference between two triangular mesh models constructed without (Figure 2(a)) and with (Figure 3(b)) the Use max move checked; 2(a)'s branch center is more tubby and 2(b)'s is more trim. Note that the model in Figure 2(a) appears to have more vertices and faces than those of the model in Figure 2(b) but both models show having the same vertices count (32) and faces count (52). The reason is the model in Figure 2(b) has overlap vertices and degenerated faces, which are not visible. They are not desirable if subdivide is needed to smooth the model. In such a case, the Simplify checkbox can be checked to merge overlap vertices and remove degenerated faces. The resulting model with Simplify check has 24 (vs. 32) vertices and 36 (vs. 52) faces. Using the Simplify option with care since the Simplify operation is time consuming (the algorithm implemented is quadratic time-complexity).



(a) Use max move unchecked (b) Use max move checked

Figure 2. Open Tube File: FileTubes ->Branches->brnaches_04.tub

The "Open Off File" button lets you open an OFF file. At this time, only triangular mesh OFF files are supported. If the Simplify is checked, after the desired OFF file is read into an internal OFF data structure, the system will perform the Simplify operation on the OFF data structure.

Subdivide Group

The Subdivide group lets you perform Loop subdivision on the open 3D model and reset to the original model. Each time you click the Loop menu, it will apply Loop subdivision once on the current mesh (the original or last mesh after Loop subdivision). You can click the Reset button to return back to the original mesh constructed from the input file. The Loop subdivide operation is

also time consuming and increases vertices and faces count exponentially. Although the system can handle a mesh having up to 2^{32}-1 vertices and faces computationally, but to graphically render the mesh it only allows a mesh having up to 2^{16}-1 (65,535) vertices, which is a Unity 3D mesh's limitation, and up to 21,000 faces, which is a limitation imposed by the wireframe shading library used to draw the wireframe view of the model.

Save Files Group

You can click "Save TriMesh to Off" button to save the current mesh (the original mesh constructed from the input file or after Loop subdivision) to an OFF file.

Transform Group

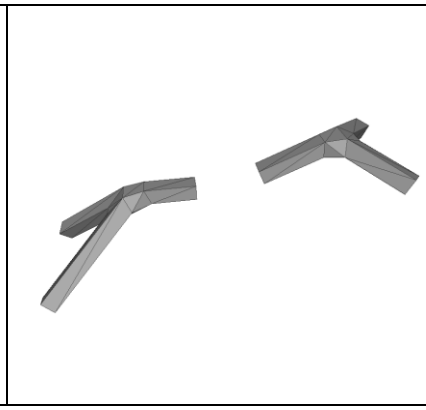
To manipulate the mesh model display, you can use left mouse button to rotate, mouse wheel to zoom in or out, and right mouse button to translate (pan) the model. These transformations are applied to the model not to the camera. Often it takes time to adjust the transformation to put model display in a desired view. To save a desired view you can click the Save button to save the current transform and click the Restore button to set the transformation to the last saved transform (for example when the model is invisible because it is outside of the display window, you may like to restore the last saved transform). Finally you can click the Reset button to back to the original transform when the input file is opened.

3 Tube and Graph file formats

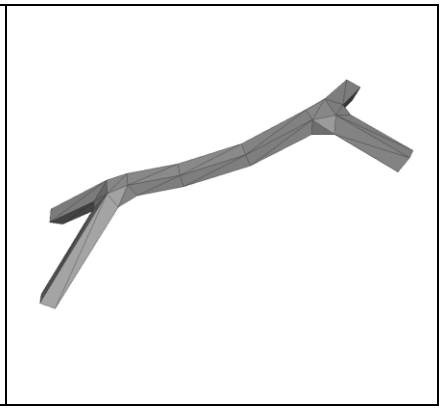
This section explains the file formats of input tube and graph files. File 3 shows an example of tube files and resulting models created from tube file input.

```
1       # bill3.tub
2       node 1 1.0 -5  10  15
3       node 2 1.0  5  10  15
4       node 3 1.0  0   5   5
5       node 4 1.0  0   2   0
6       node 5 1.0 -5 -10 -10
7       node 6 1.0  5 -10 -10
8
9       branch 1   0 10 10
10      1 2
11      3
12      branch 2   0 0 -5
13      4
14      5 6
15
16      connect 3 4
```

(a) Input file



(b) Two disconnected branches (branch 1, lines 9-11, on the right and branch 2, lines 12-14, on the left)



(c) Connected branches (out-node 3 connected to in-node 4, line16)

Figure 3. Tube Structure Input File and 3D Branching Model Created

Tube File Format

The tube file format is basically a line text (i.e. using ASCII character set coding) file format, where each line defines a record or empty including white-space characters for readability. We will use the file in Figure 3 to explain the file format (note the line numbers are not part of the file).

- The special character # begins an in-line comment, which will be skipped by the parser. For example, since line 1 begins with a #, the remaining of the line has no input to the system. You can use comments just like in any programming languages to document your input files.
- Any number of empty or white-space character lines can be inserted (maybe for readability purpose) between record lines.
- There are three sections in the file: first section defines nodes, second defines branches, and third defines connection between nodes. In Figure 3, lines 2-7 define six nodes, lines 9-14 define two branches, and Line 16 defines a connection between two nodes. These three sections must be given in the order as shown in the file given in Figure 3(a).
- A node definition line has the following format:

  node *id r x y z*

  where *id* is a non-negative integer defining the unique id of the node, *r* is a floating-point number defining the radius of the node, and (*x, y, z*) are three floating-point numbers defining the 3D position of the node. Nodes are classified exclusively as in-nodes or out-nodes of a branch, or connect nodes (used to connect branch nodes) and branch nodes (in-nodes or out-nodes) cannot be shared between two different branches as explained latter.
- A branch definition has the following format:

  branch *id* [*r*] *x y z*
  <*list-of-in-node-id*>
  <*list-of-out-node-id*>

  where *id*, [*r*], *x*, *y*, and *z* have the same meanings as the node record, except here the radius *r* is optional (denoted by []). The <*list-of-in-node-id*> lists the ids of all the branch's in-nodes, which must be defined in the node section. In-nodes are those branch nodes that serve as input to the branch center. If <*list-of-in-node-id*> just contains a -1, there are no in-nodes to the branch. The <*list-of-out-node-id*> lists out-node ids similar to the <*list-of-in-node-id*> for in-nodes. For example, lines 9-11 in Figure 3(a) define a branch with id 1 center at (0, 10, 10) and of in-nodes 1 and 2, and of out-node 3.
- A connect definition has the following format:

  connect *node1-id inode2-id* [<*list-of-node-ids*>]

  where *node1-id* cannot reference to a branch in-node, and the last node-id in the list

cannot reference to a branch out-node. For example, line 16 in Figure 3(a) defines a connection between node 3 (an out-node of branch 1) and node 4 (an in-node of branch 2). Note the following line input

```
connect 1 2 3 4
```

has the same meaning as the following three-lines input

```
connect 1 2
connect 2 3
connect 3 4
```

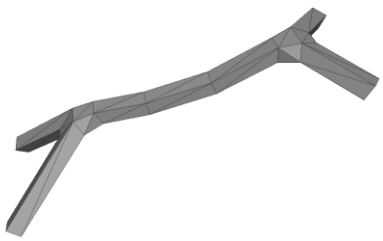
As you can see from the tube file format and its semantics, the tube data structure defines *directional* tube structure, in which each branch is self-contained and can be easily removed and inserted without affecting other branches. This is one of the most important considerations of the tube structure design. Connected nodes are used to connect branches. It seems it is kind of tedious to define the tube structure. However if you do not care about in and out direction of branch nodes, you can use simpler undirected graph files to input to the system. Internally, there is tool that converts *undirected* graph into the *directional* tube structures.

Graph File Interface

A graph file defining nodes including their radius and (x, y, z) position and edges (i.e. connectivity of nodes) can be input to the system to construct a desired 3D branching model as shown in Figure 4. As you can see from Figure 4(a), the graph file format is very straightforward and similar to the tube file format. As matter of fact, the graph file format only allows two sections: node and connect and it has no branch section in-between them.

```
1       # bill3.gra
2       node 1 1.0 -5  10  15
3       node 2 1.0  5  10  15
4       node 3 1.0  0   5   5
5       node 4 1.0  0   2   0
6       node 5 1.0 -5 -10 -10
7       node 6 1.0  5 -10 -10
8
9       node 10 1.0 0 10 10
10      node 20 1.0 0  0 -5
11
12      connect 1 10
13      connect 2 10
14      connect 3 10
15      connect 4 20
15      connect 5 20
15      connect 6 20
16
17      connect 3 4
```

```
1       # bill3.grap.tub
2       node 1 1.0 -5  10  15
3       node 2 1.0  5  10  15
4       node 3 1.0  0   5   5
5       node 4 1.0  0   2   0
6       node 5 1.0 -5 -10 -10
7       node 6 1.0  5 -10 -10
8
9       branch 10 1  0 10 10
10      -1
11      1 2 3
12      branch 20  1 0 0 -5
13      4
14      5 6
15
16      connect 3 4
```



(a) Graph input file (b) Converted tube structure file (c) Constructed 3D branching model

Figure 4. From Graph Input File (a) to Tube File (b) to 3D Branching Model (c)

It is interesting to compare the directional tube structures defined in Figure 3(a) and Figure 4(b), respectively. Branch 10 defined in Figure 4(b) is a source branch (i.e. everything flows out the branch center), whereas the corresponding Branch 1 in Figure 3(a) is not. This difference is due to the algorithm we used to convert undirected graph structures to directional tube structures. As far as 3D modeling is concerned, they created identical mesh model (see Figure 3(c) and Figure 4(c)). However, with the graph file interface, because it is easier and graphs are so commonly used in many application domains, it opens up many techniques, some straightforward and some algorithmically interesting, to construct 3D tube models using the graph interface to the system.

Tube File Format in Coco/R Extended Backus–Naur Form (EBNF)

The following lists the EBNF used in Coco/R definition of the tube file format syntax.

```
Tube = { nl } Nodes Branches Connects .
Nodes = Node { Node } .
Node = "node" intcon   Number Number Number Number nl { nl } .
Branches = { Branch } .
Branch = "branch" intcon Number Number Number [ Number ] nl
           // Branch in nodes; a -1 means no in nodes
           intcon { intcon } nl
           // Branch out nodes; a -1 means no out nodes
           intcon { intcon} { nl } .
Connects = { Connect } .
Connect = "connect" intcon intcon { intcon } { nl } .
Number = intcon | floatcon .
```

4 Sample Files Provided

The sample data files given are organized in the application folder called TriMesh_GUI_Data in three subdirectories as listed below:

FileTubes	which contains tube files
FileGraphs	which contains graph files
FileOffs	which contains OFF files

You can explore these subdirectories and open files contained in them to view the created mesh models. Only examples of tube files contained in the FileTubes folder will be explained further. The tube files in the FileTubes folder are grouped to several subdirectories with directory name reflecting the kinds of models that can be created form the tube files contained in the subdirectory. For example, the Trees subdirectory contains tube files that create tree branch like models. These subdirectories in the directory, FileTube, are explained in the following paragraphs.

Branches Subdirectory

This subdirectory contains tube files we created manually to show different branch structures. Figure 5 shows a branch with 10 nodes (tubes) of different radii. Note the open file name and vertices and face count are displayed near top of the display window.

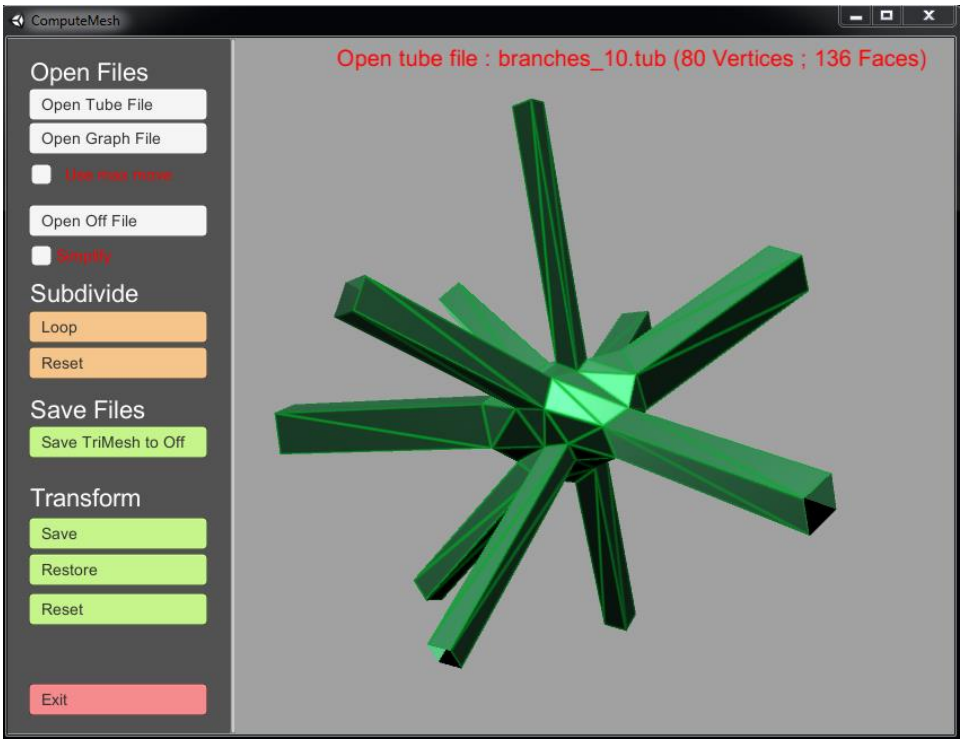
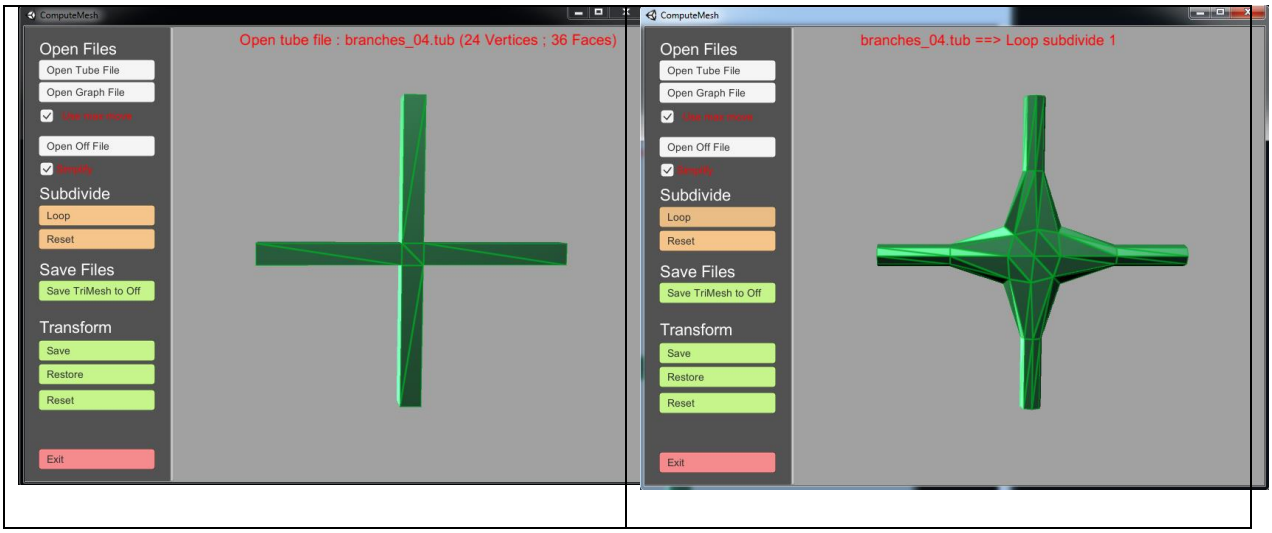


Figure 5 Model created from the tube file branches->branch_10.tub

To understand the system, it is a good exercise to use a text editor to modify the file, branch_10.tub. For example, change the radii and positions of the branch nodes or add a new branch node to the file, and open the modified file to view the resulting model created from the file input and compare the differences in the model shown in Figure 5 and your model.

Figure 6 shows the model created form the open tube file, branches_04.tub, with Use max move and Simplify checked and the model after one Loop subdivide operation. If Simplify is unchecked, the model after one loop subdivide will be very different. Try it out. Do you know why with and without Simplify checked they look so much different?

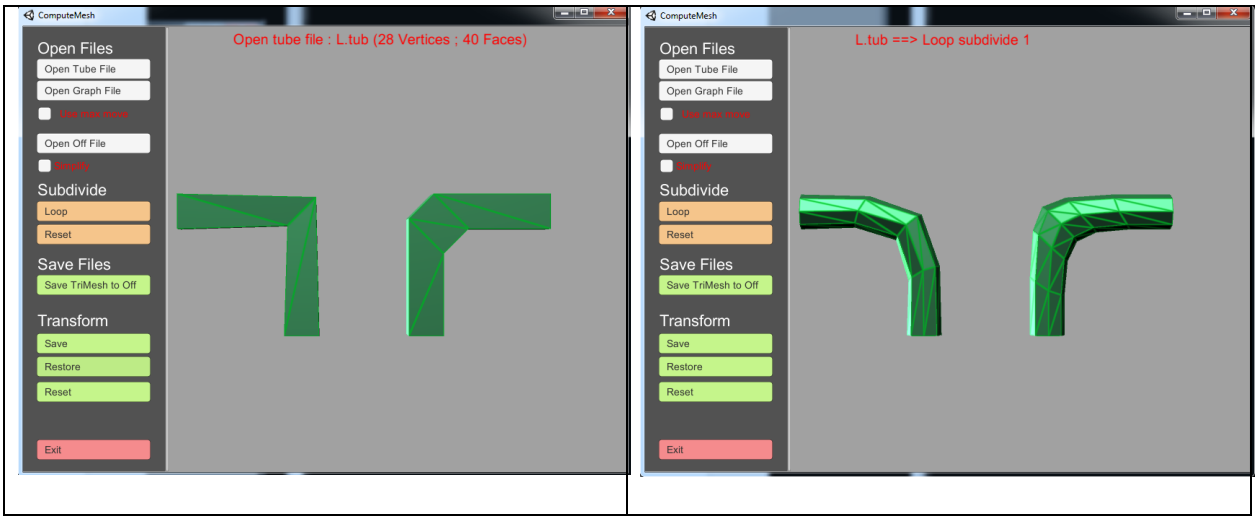


(a) Model from input tube file, branches_04.tub (b) Model after one Loop subdivide

Figure 6. Open with Use max move and Simplify checked

Branch_vs_Extrusion

This subdirectory contains tube files demonstrating the difference in modeling using branch and extrusion, respectively. Figure 7 shows the modeling result from the tube file, L.tub.



(a) Model created from L.tub (b) Model after one Loop subdivide

Figure 7 Branch vs extrusion modeling

Lattices

This subdirectory contains tube files which creates lattice like models. These tube files are procedurally generated. Figure 8 shows the model created from the file, lattice_5_5_4.tub.

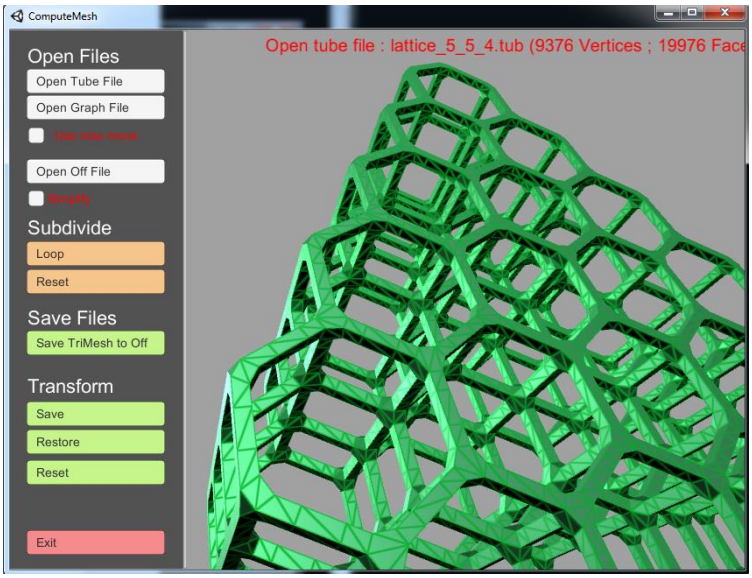


Figure 8. A lattice model created from the tube file lattice_5_5_4.tub

Polyhedra

This subdirectory contains tube files, which create polyhedra skeleton structures. These tube files are generated from polyhedron 3D data set to graph files with estimated radii at each node first and then these graph files are converted to tube files. Figure 9 shows a Dodecahedron skeleton structure from the tube file Dodecahedron.tub in the subdirectory. You can experiment with the different radii assigned to the branch nodes and connect nodes and examine the differences in the created models.

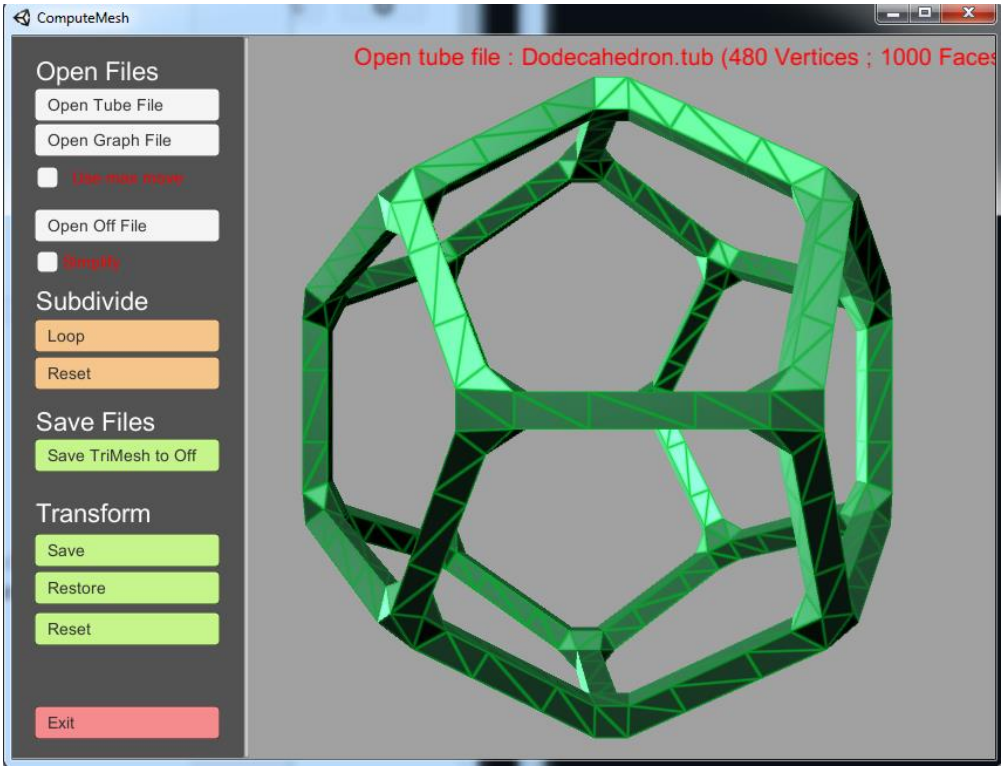


Figure 9. A Dodecahedron skeleton structure created form the tube file Dodecahedron.tub

PS Subdirectory

This subdirectory contains tube files generated from surface parametric equations such as sphere, torus, and Mobius band. Figure 10 shows the Mobius band model created from the tube file Mobius_16_5_v2.tub in the subdirectory.

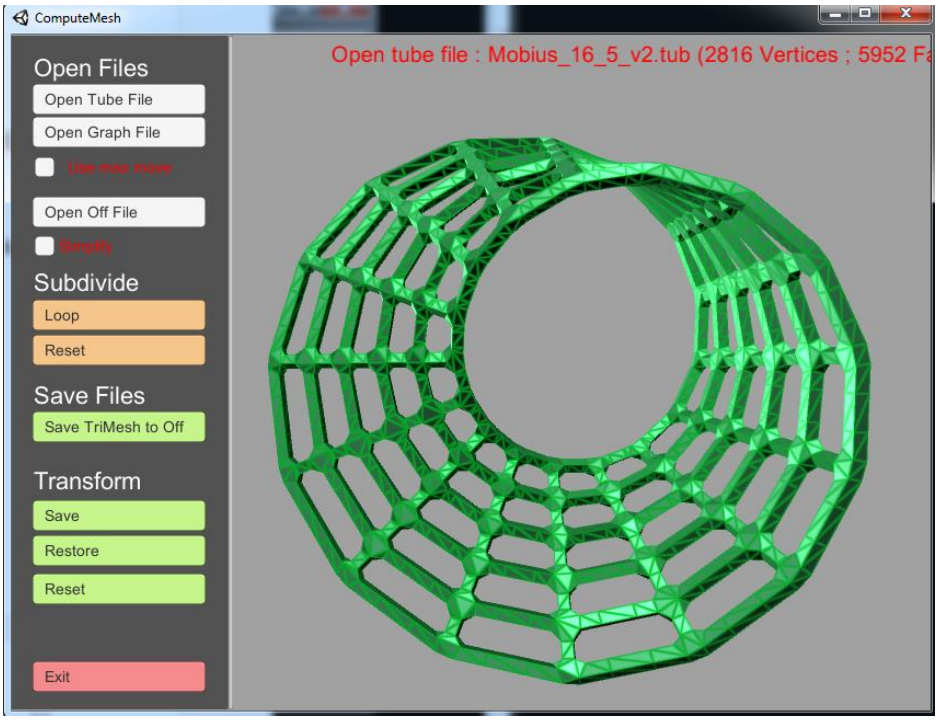


Figure 10. A Mobius band model created from the tube file Mobius_16_5_v2.tub

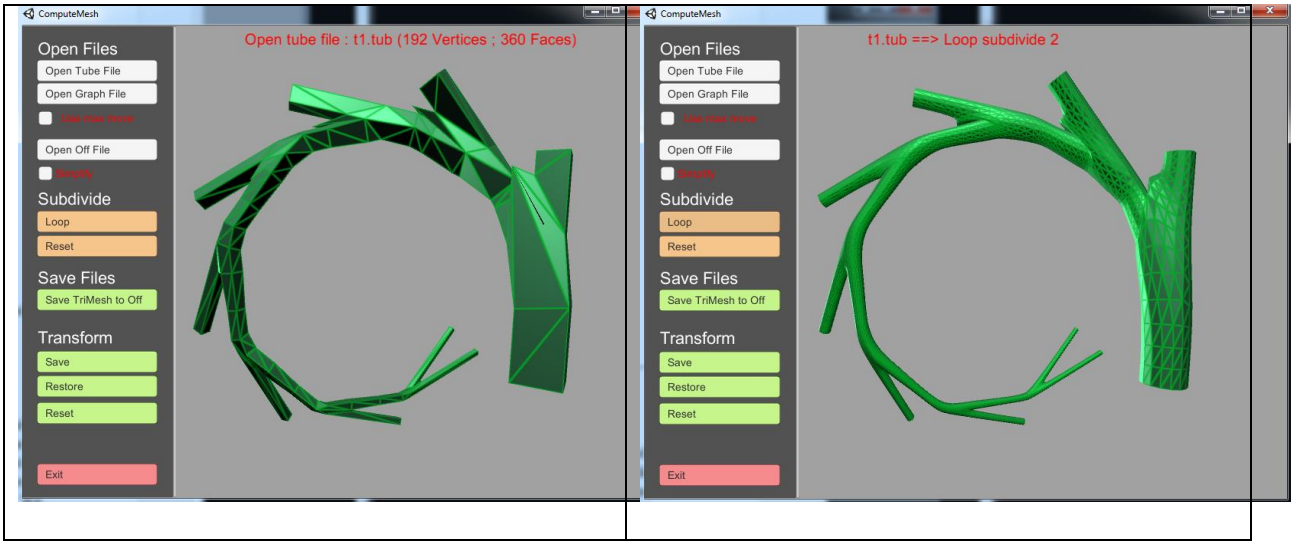
Similarly, from any 3D triangular mesh model, we can create a graph file from the mesh's vertices and edges with estimated radius assigned to each vertex and then use the graph file to input to the system to generate a kind of skeletal 3D model of the given mesh model. In fact, in our system library, there is a method does exactly like we have just talking about from an OFF file input.

Retinal Objs Subdirectory

This subdirectory contains the tube files used by us to simulate bipolar retinal cell triangular mesh model (see Figure 1). These tube files are prepared with the help a 3D modeling package to create splines exported to graph files and then they are converted to tube files.

Tree Subdirectory

This directory contains tube files that create tree branch like mesh models. Some of these tube files are created manually and some are created procedurally using L-system or particle system. Usually we create graph files first then convert them into tube files. Optionally the graph files can directly input to the system by clicking the "Open Graph File" button. One of the challenges we encounter when we create these graph files is how to assign radii to graph nodes so that we can end up with good tree branch models. We are continuing working on the radius assigning problem. Figure 11 shows a simple tree branch model created form the tube file, t1.tub, in the subdirectory and the model after two Loop subdivisions.



(a) The model created from the input (b) The model in (a) after 2 Loop subdivides

Figure 11. The tree branch model created from ti.tub and its Loop subdivide result

5 Limitations and Performance Issues

Up-to 65,535 vertices is imposed by Unity mesh and up-to 21,000 triangular faces is imposed by the wireframe shaders used in the graphics rendering of the Windows application. When you see a button or checkbox turned to red color, it indicates the application is busy doing your requested command. The mesh Simplify and Loop subdivide operations are very time consuming for large models. As a result, sometimes you may need to wait for a long time watching the clicked button or checkbox stayed in red. Once the system completes the command, the button or checkbox will return to its normal color. Our advice for using these time-consuming commands is be patient when the system is busy and use them with care.

Final Notes

I hope you enjoy using our system. Please let us know any problems you find and share with us your cool models and explain how you create them.