Mesh Simplification Using Vertex Clustering

Alex Bechanko
atbechan@mtu.edu

Table of Contents

Building This Project

Dependencies

Base dependencies:
The project expects that you are compiling this on a Linux machine, where a version of Make is available for this project to use for making compilation easy. The project also requires that the GCC compiler for the C++ programming language is installed. The version of the compiler should not matter, but the only version this project has been tested with is GCC version 4.4.5.

Libraries:
The libraries this project depends on GLUT, GLU, and the base OpenGL libraries. If you have the ability to compile and run OpenGL programs, chances are very good that you can compile and run this project.

How to Compile

To compile all the executables available to this project, change the directory to the top directory of the project *mesh_refine*, and run the command *make* with no parameters to compile all the executables. In a command line form, this would look like:

```
[user@pc] cd path/to/mesh_refine
[user@pc] make
```

If you wish to only compile one of the three executables made in this project, run make one of the following commands: *viewer, obj, simplify*. Running these commands will only build a specific executable from this project.

To build only Viewer:

```
[user@pc] cd path/to/mesh_refine
[user@pc] make viewer
```

To build only Obj:

```
[user@pc] cd path/to/mesh_refine
[user@pc] make obj
```

To build only Simplify:

```
[user@pc] cd path/to/mesh_refine
[user@pc] make simplify
```

Running Viewer

Description

Viewer is an application designed to make viewing meshes much easier. The way it accomplishes this is by using the mouse to and the buttons on the mouse to move the mesh in ways that make sense. Viewer also uses quaternions to make rotating the mesh in the virtual space much easier.

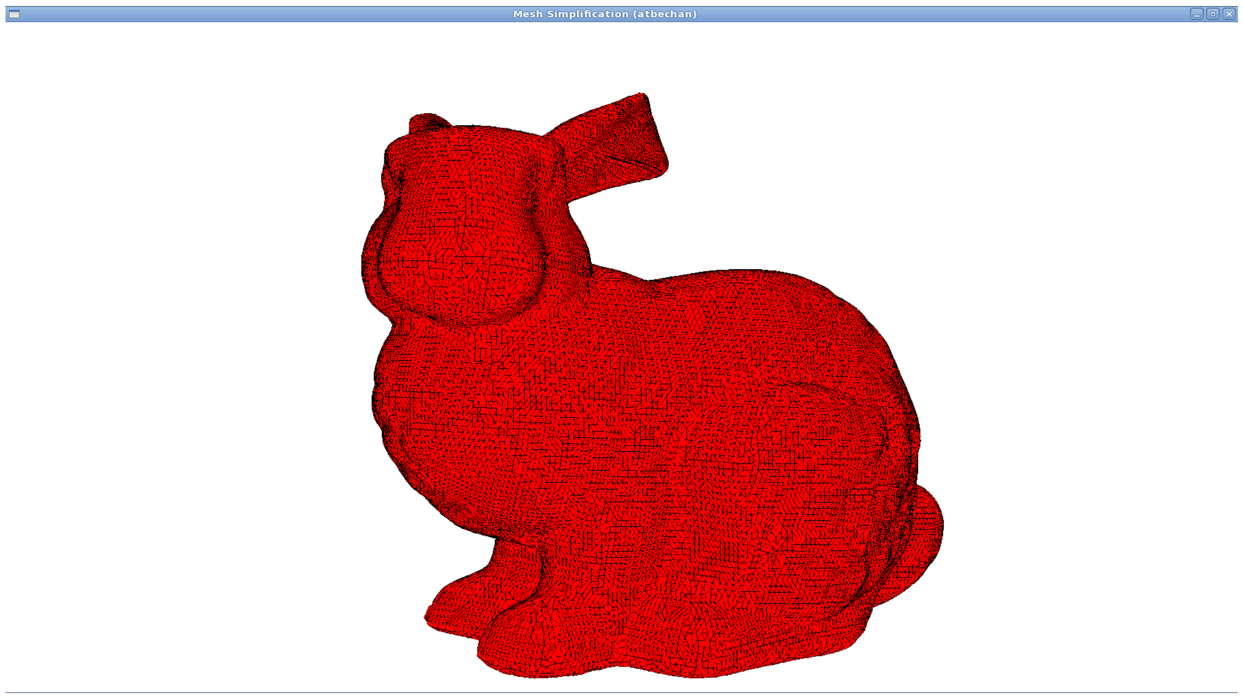
Commands

To run *viewer*, you must have access to a terminal window. Change the directory so that you are in top directory *mesh_refine* of this project, and then change the directory to *bin*, which is a subdirectory of the *mesh_refine top directory. Then run the program viewer in the bin* directory with a specific OBJ mesh file that you wish to run. In the command line window, it would look like this:

```
[user@pc] cd path/to/mesh_refine
[user@pc] cd bin
[user@pc] ./viewer path/to/mesh.obj
```

Interface

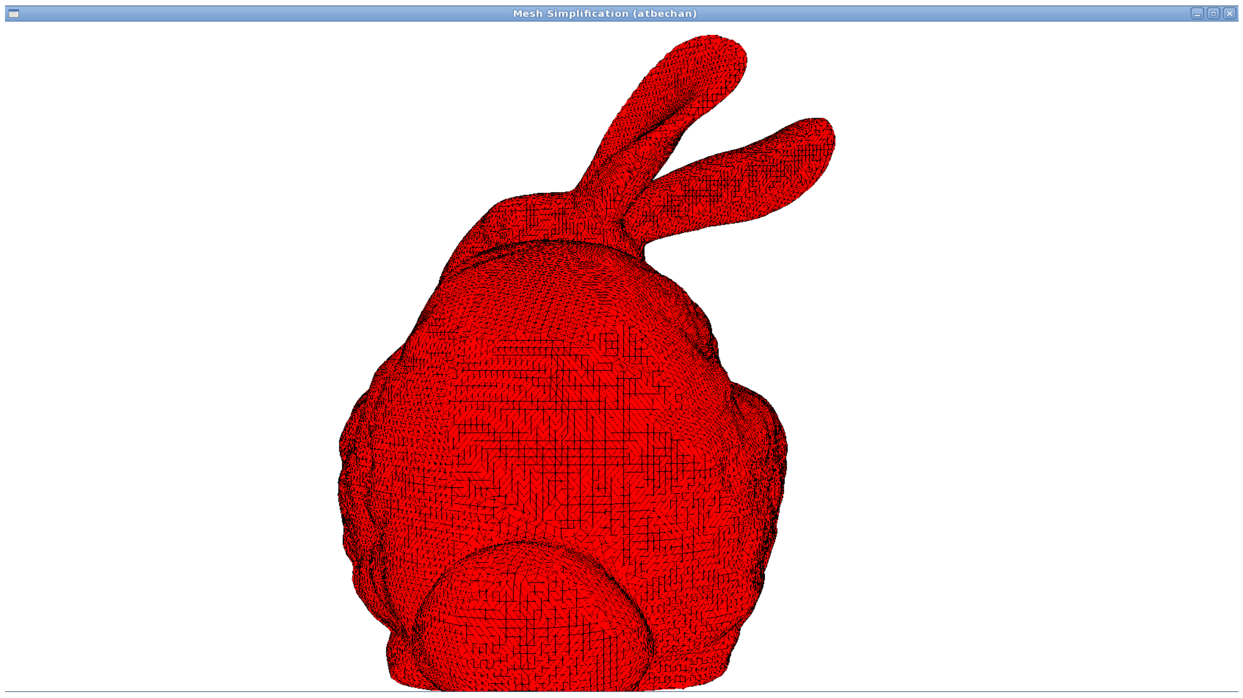
The interface of the Viewer application is really simplistic. There are no buttons, or anything on screen besides the mesh that you wish to view. A picture is below.



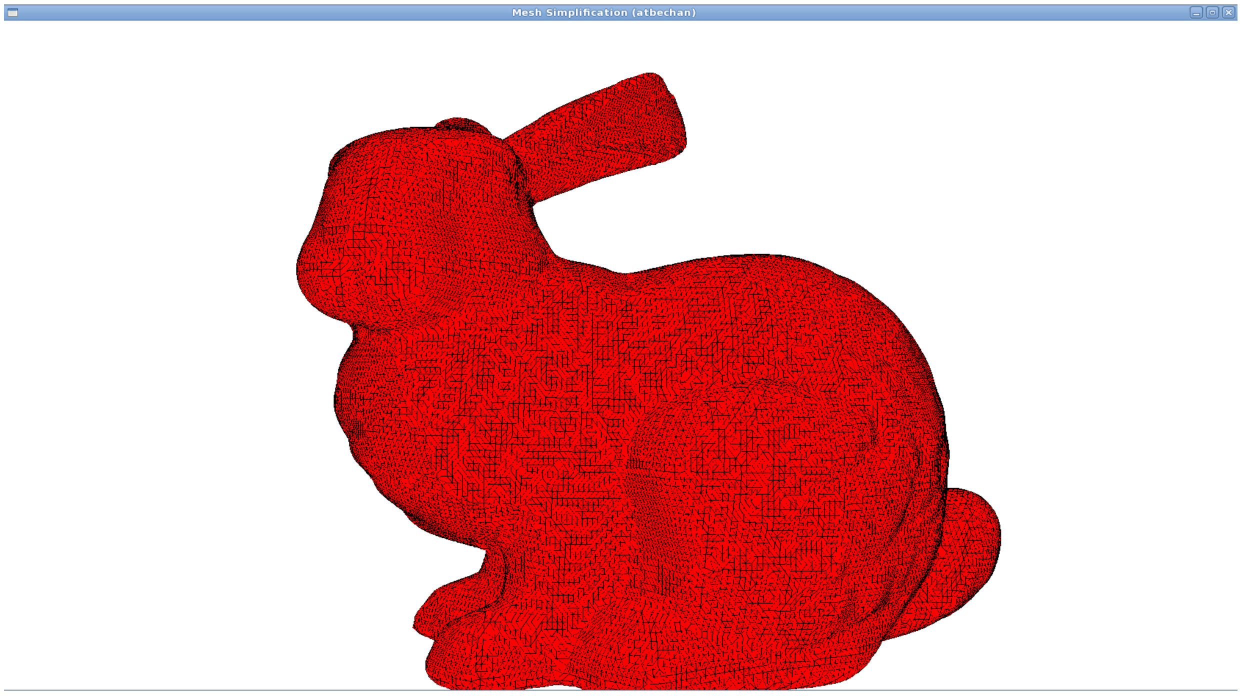
The mesh itself can be seen by the outlines given to each polygon used in the defining of the mesh. The *viewer* can handle any OBJ file formatted meshes, including those that use different kinds of polygons to draw the mesh.

Moving the Mesh

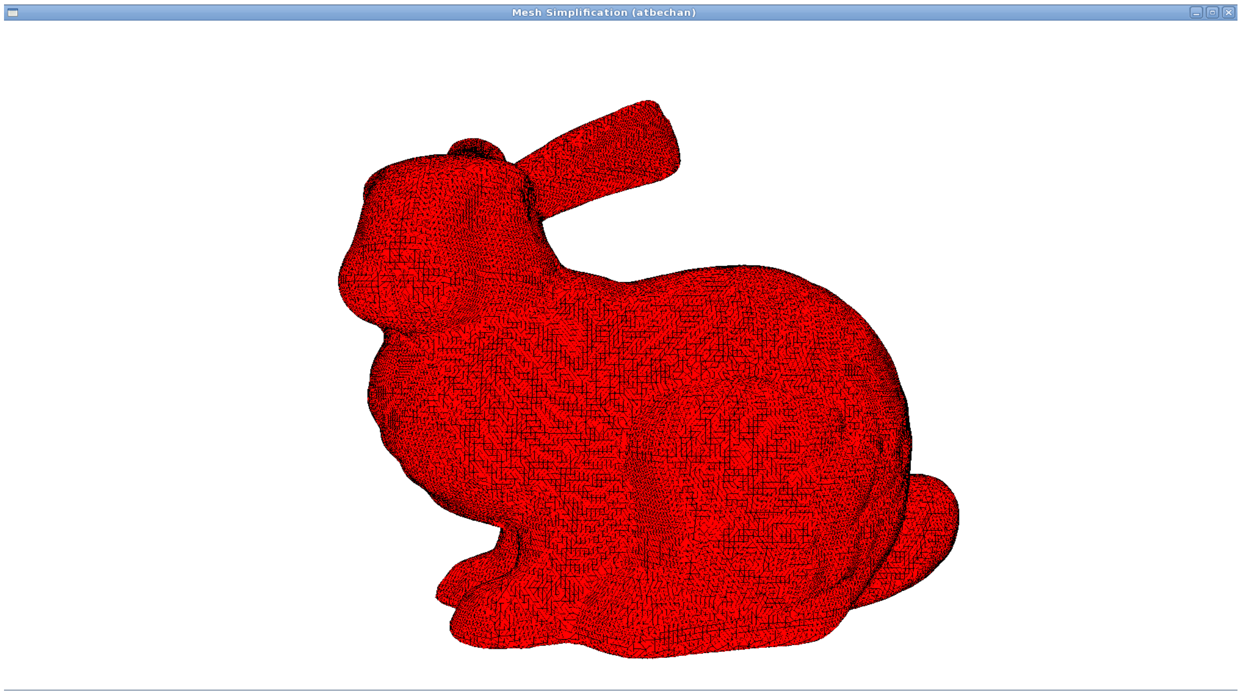
To move meshes in the screen, there are three main ways, you can rotate them left/right, rotate them up/down, and you can zoom in or zoom out on a given mesh. To start, we will show how we positioned the Stanford bunny in the picture above. Upon opening the Stanford bunny mesh, *bunny.obj*, we can see that the default positioning of the bunny looks like this:



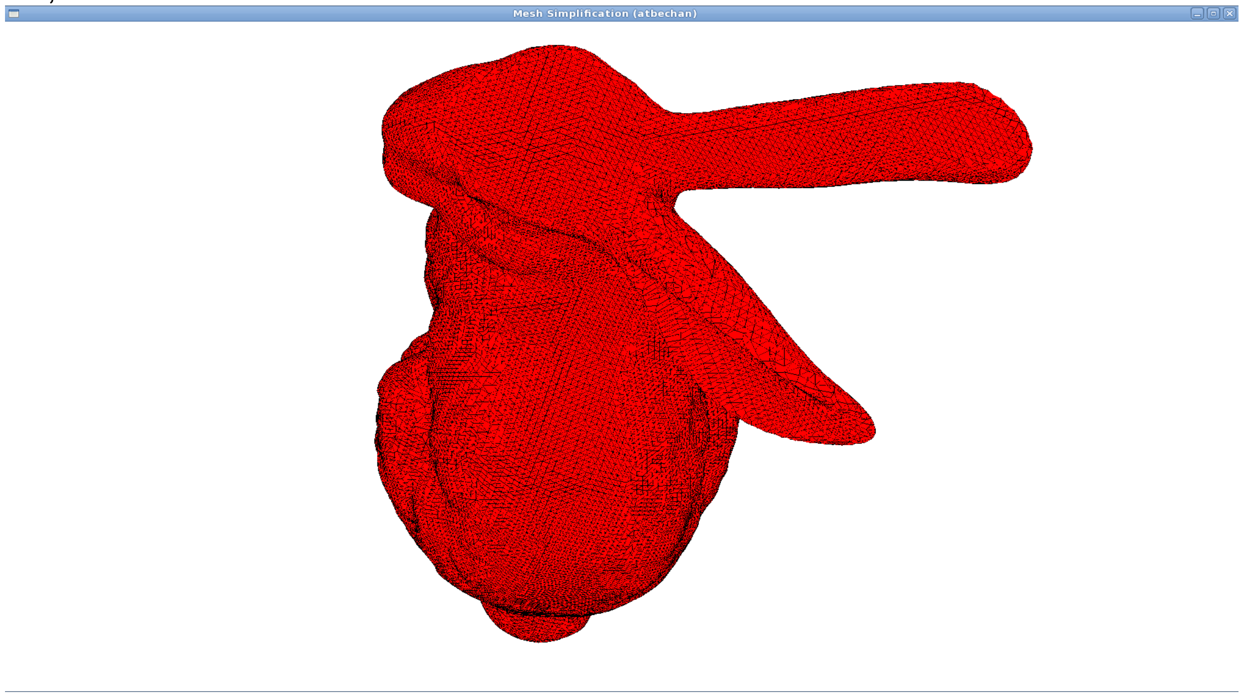
First we will rotate the bunny right so that the bunny is facing us. To do this press the left mouse button and move the mouse to the right. This gives us:



Next we will zoom out a little bit so that the bunny is in the screen fully. To do this move the mouse wheel towards you. This will move the bunny further back in the screen, making it look like this:



This will roughly reproduce the picture we made previously. If you wanted to get a better look at the bunny’s head, this will require us to rotate the bunny towards us. To do this, click the left mouse button and drag the mouse down slowly. This will rotate the bunny so that its head will be closer to us than the rest of his body. If we had done this as soon as we opened the viewer, this is what we would see:



Running Simplify

Description

The *simplify* program is the heart of this project. This program is what simplifies the mesh into a different mesh with a lower face count. The *simplify* program does this by performing a vertex clustering algorithm using a specific refinement measurement to refine the mesh. The program then outputs this mesh into a specified file so that the mesh may be used later or viewed.The algorithm used is described in a later section.

Commands

To run the *simplify* program, you must have a command line window open. First change the directory so that you are in *mesh_refine*, the top level directory of the project. Then change the directory once more so that you are in *bin*, a subdirectory of the *mesh_refine* directory. Once you have done this, run the *simplify* program found in the *bin* directory with the given OBJ mesh file you wish to run the refinement algorithm on, and the level of refinement you would like to refine the mesh with, and then finally the name you would like to give the OBJ file this program outputs. In a command line window, you would see:

```
[user@pc] cd path/to/mesh_refine
[user@pc] cd bin
[user@pc] ./simplify -r level path/to/mesh.obj path/to/output.obj
```

This command will refine *mesh.obj* with the *level* amount of refinement, and then output the resulting mesh into *output.obj*. The *level* of refinement describes how big of a partition should be used when performing vertex clustering. So if you have a mesh that is described using coordinates that are very small, a very very small level of refinement is recommended, otherwise you will refine the mesh too much.

Running OBJ

Description

The *obj* program is a simple program that was used to test the OBJ file format loading and saving mechanisms that are used in the *viewer* and the *simplify* programs. This program is typically run from the command line window, and gets passed the file path to a OBJ mesh file. The program then will output some basic information about the mesh, and if you give it an extra parameter the program will also save the mesh to a certain filename.

Commands

As stated above, the program *obj* is run from the command line window. To run it, first change the directory so that you are in *mesh_refine*, the top level directory of the project. Then change the directory to *bin*, a subdirectory of the *mesh_refine* directory. The run the *obj* program found in this directory by giving a path to a OBJ mesh file. If you wish to make a copy of the mesh using this utility, call the program with the parameter *-p*, followed by the path you wish to save the mesh to, followed by the path to the mesh you wish to use. In the command line window, you would see:

```
[user@pc] cd path/to/mesh_refine
[user@pc] cd bin
[user@pc] ./obj -p path/to/save/mesh.obj path/to/mesh.obj
```

The commands run above will save the *mesh.obj* found in *path/to*, and save that mesh to the path *path/to/save/mesh.obj*,

Project Summary

Algorithms

The main algorithm that was used is a slightly different version of vertex clustering. Normally, the way vertex clustering is performed is that the space the mesh resides in gets partitioned into many subspaces, and in those subspaces all the vertices that are in that subspace get mapped to a single vertex that is also in that same subspace. The algorithm this project uses is often called a floating partition vertex clustering. First the vertices are sorted according a measurement of how important they are as a feature to the mesh. Then we pick the most important vertex, and create a subspace around it that is a given size, then map all the vertices we find in that subspace to that important vertex. The advantage to using this algorithm instead of the fixed partitions algorithm is that this produces better results more often, due to the fact that the angles and shapes in the mesh are being taken into account.

Difficulties

The difficulties in this project were relatively minor. The main difficulty that was had was that I had originally written this in the Python scripting language, but when I stress tested the program the amount of time that it took to view a mesh of a reasonably large size was way too long. To fix this problem I had to rewrite the program in a lower level language, C++. Thankfully the transition was relatively easy though, with very few problems in translating between the two languages. The other main difficulty was that testing of the simplification process could not really be done until I had a working version. This meant that I could not do incremental testing that I would normally do for projects. Thankfully again, this was another area that went fairly smooth once the problems were found. Lastly, I had difficulty in making the run time complexity of my version of the floating partition vertex clustering algorithm the same as the run time complexity described in the paper. My algorithm technically does performs in $O(n^2)$ time complexity, while the algorithm described performs at $O(n \log n)$. The reason they have a better time complexity for the algorithm is because their version stores the information regarding proximity of vertices into a table before running the algorithm. Unfortunately I had difficulty in getting this algorithm to work for a reasonably sized mesh on my development computer, because I was unable to allocate the space necessary for that table. So in order to run with bigger meshes on a machine with a low amount of memory, I altered the algorithm a little so that it computed the closeness of vertices as needed, instead of storing all values. This allowed me to run bigger meshes, and the run time of the program was still very low, taking about a couple seconds for a mesh with seventy thousand vertices.

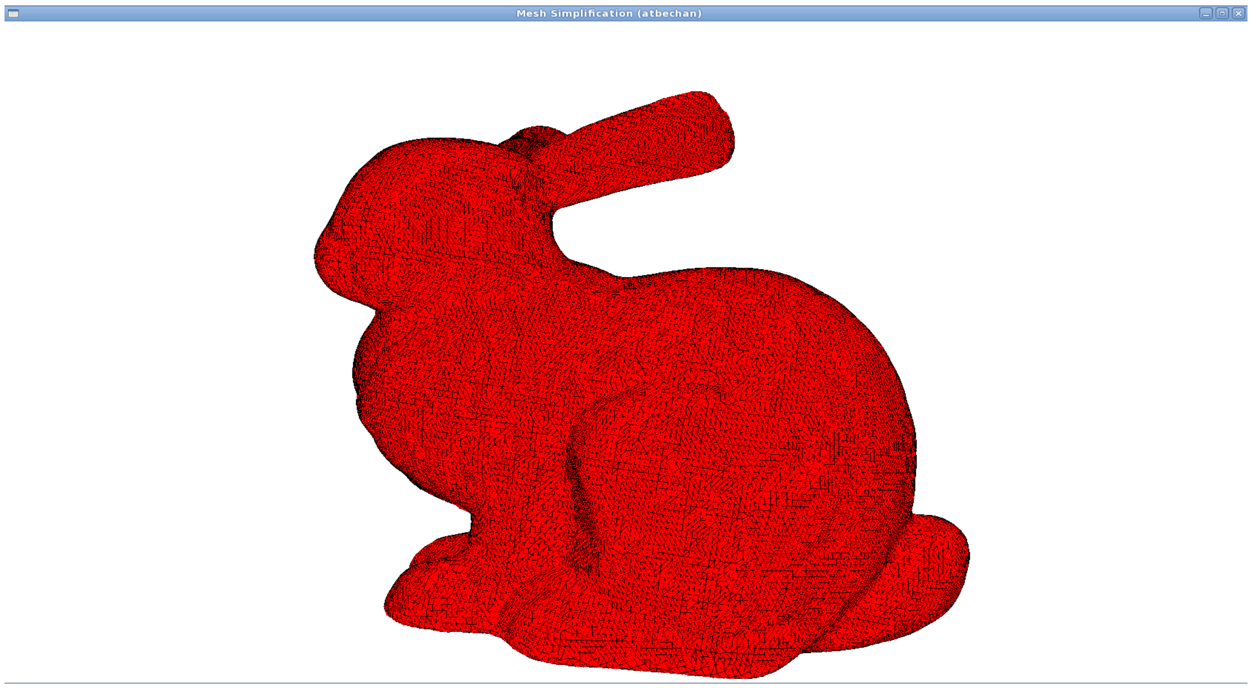
Areas to Improve

There are of course, many areas in which this project can be improved. The first thing I would improve is the algorithm I use. I would make it very easy to switch between using the version I have and the faster but more memory intensive version described in the paper. The next thing I would improve is the mesh viewing program *viewer*. I believe that this could be improved by making it easier to see changes by adding the ability to reload a mesh, as well as refine the movement controls for the mesh a bit more. There is also a lot of room for code refinement. In particular I would move the code I use to triangulate the mesh into a different file, or even perhaps into the OBJ file loader library that I use. This would clean up the way the code looks even more. Also, I would try to use less standard library containers and functions. These classes and functions are very handy when programming this, but I believe if I did not use them I would make it easier for me to run bigger meshes on low memory computers.

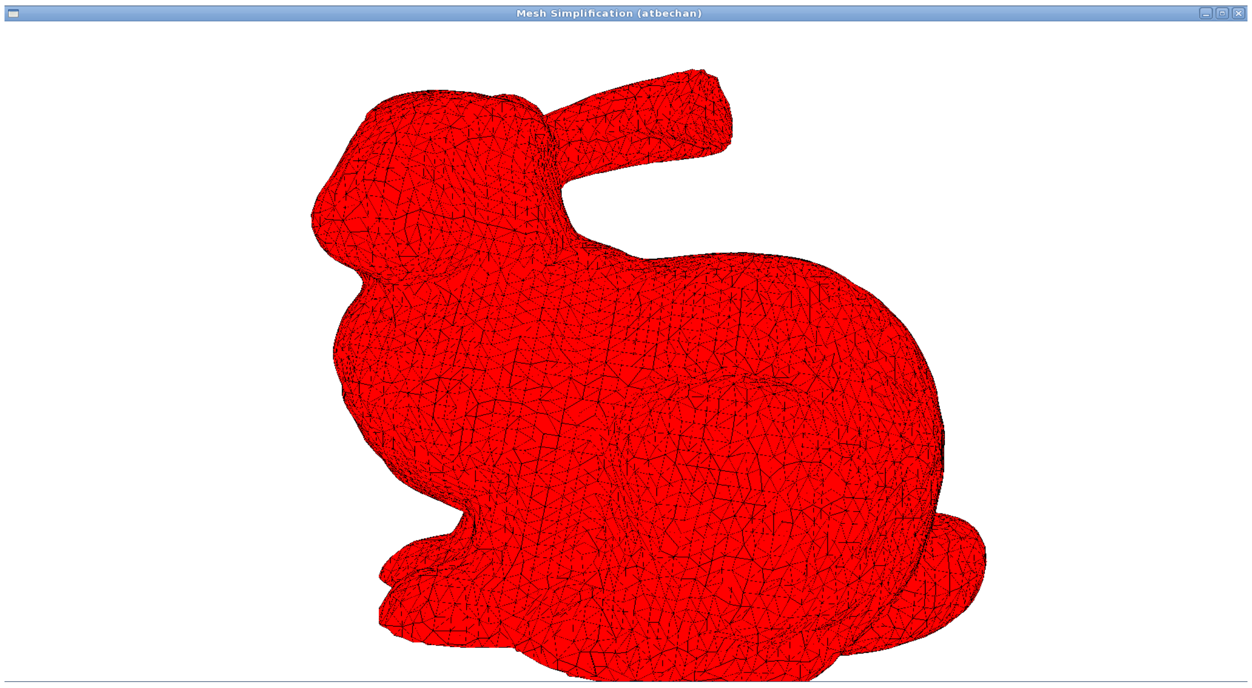
Examples

Stanford Bunny

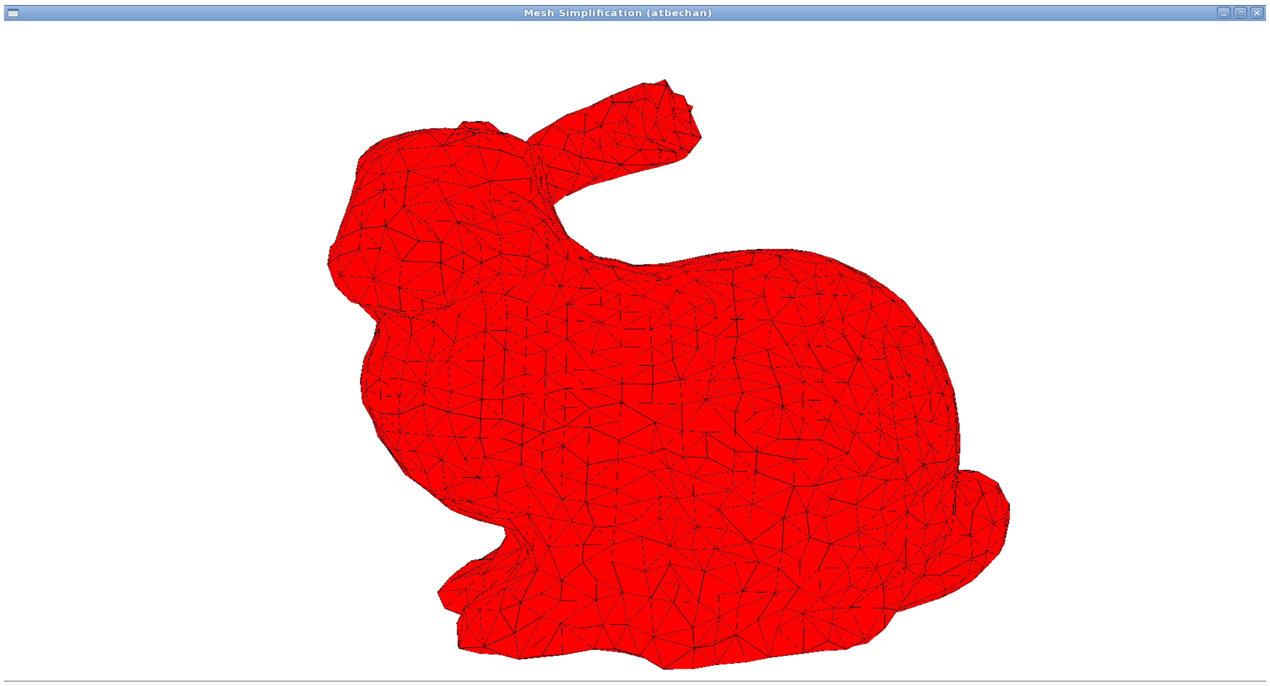
Before:



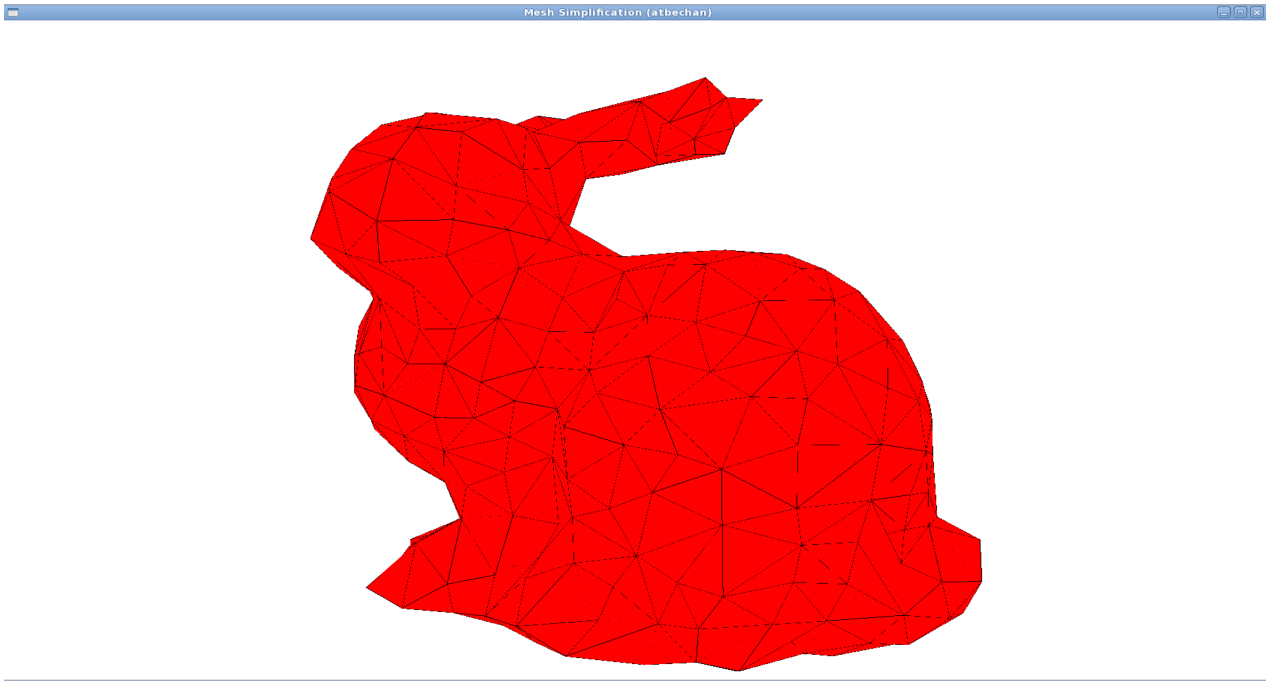
After simplifying with partition radius of 0.1:



After refining with partition radius of 0.25:

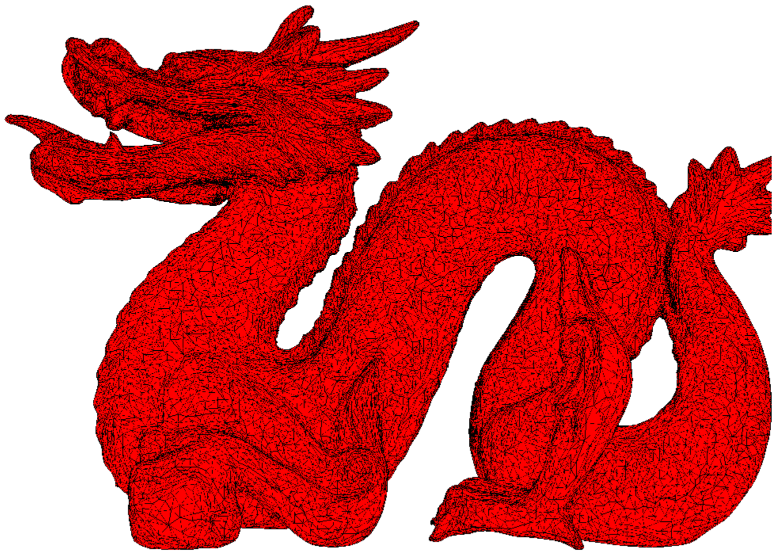


After refining with partition radius of 0.5:

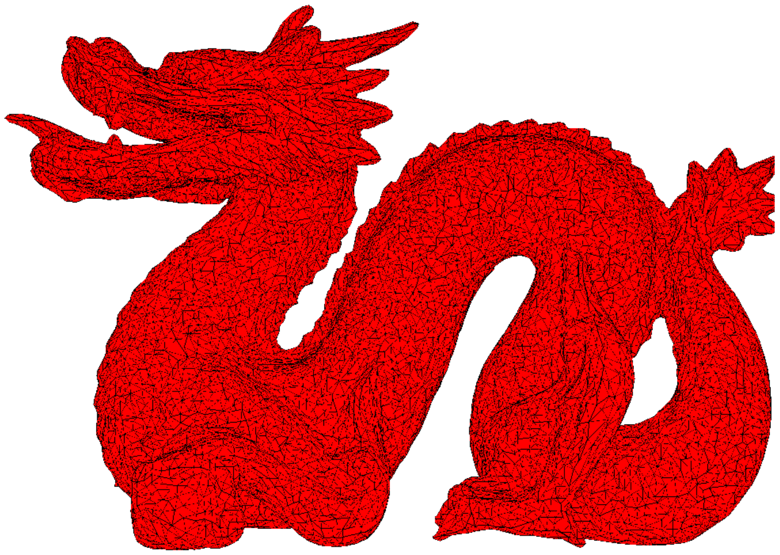


Dragon

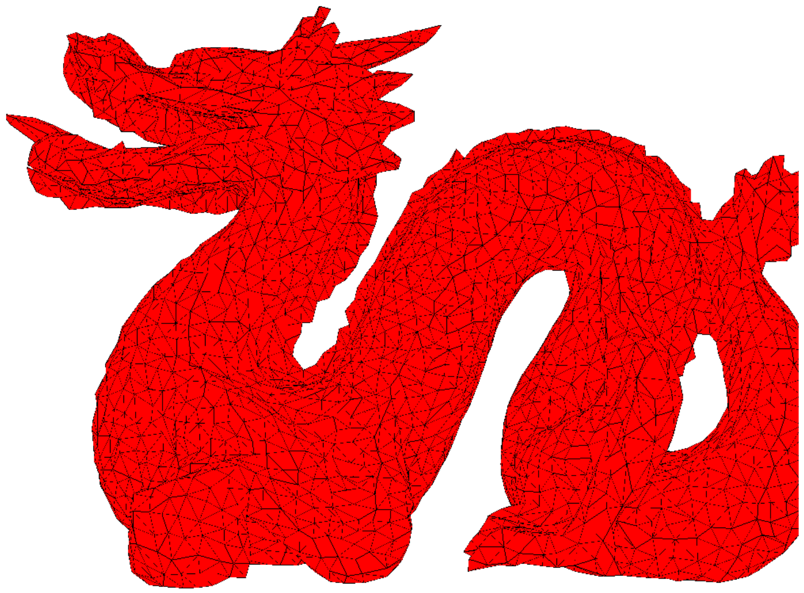
Before:



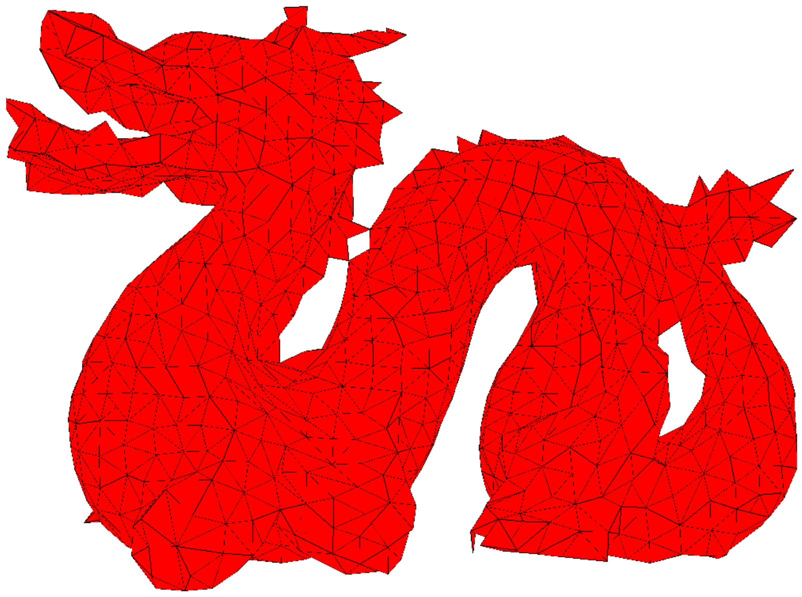
After refining with partition radius 0.01:



After refining with partition radius 0.025:

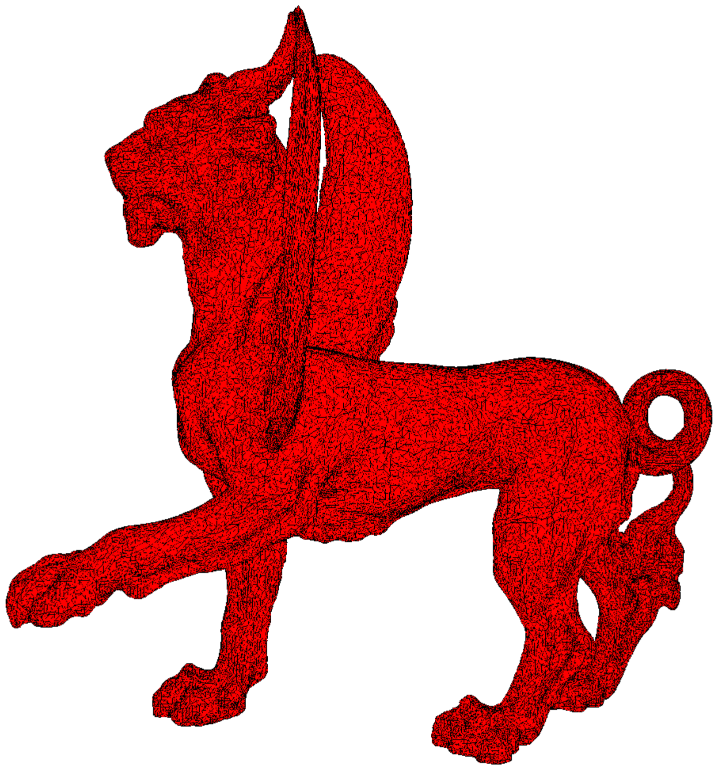


After refining with partition radius 0.05:

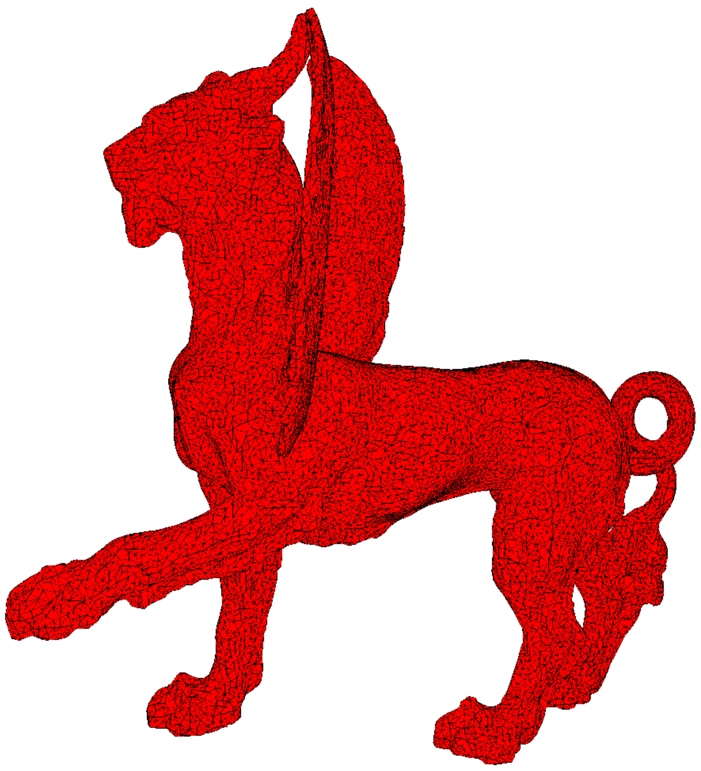


Feline

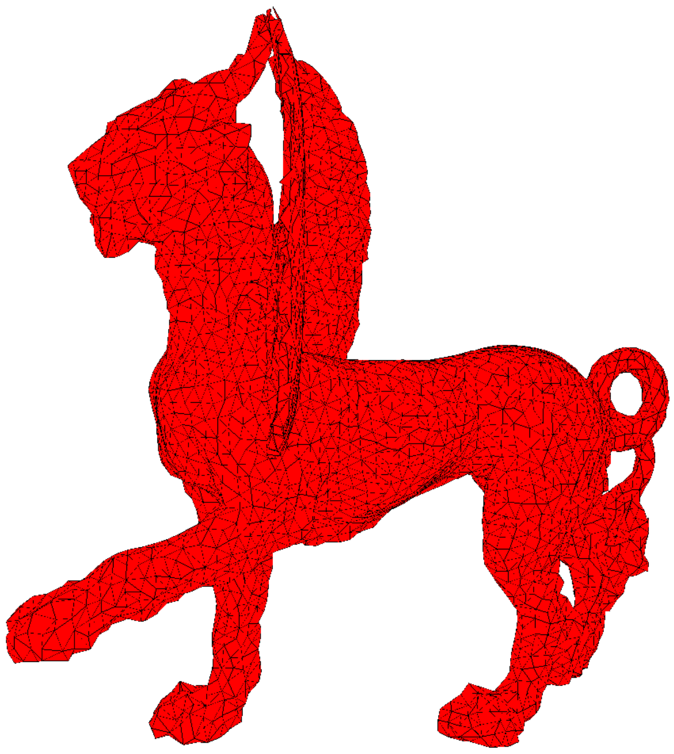
Before:



After refining with partition radius 0.01:



After refining with partition radius 0.025:



After refining with partition radius 0.05: