Homework 8 : Mesh Subdivision & Simplification

Project Report

Sarah Gross, student 2017280160

1. Work structure

In order to write the subdivision code, I followed the great tutorial provided by Carnegie Mellon University which explains step by step the subdivision process.
https://github.com/cmu462/Scotty3D/wiki/Loop-Subdivision

I realized reading this tutorial that I would need a framework supporting HalfEdge implementation. I tried without success to use the one provided by CMU, then I found another provided by Snrao310 on Github.

This framework, located in Core/ folder in my project, gives helpful implementation of HalfEdges, Edges, Vertices and Faces. It also provides convenient iterators to loop through the edges, vertices or faces of the mesh. It took me some time to read and understand this framework structure, but then I could concentrate on implementing the Subdivision/Simplification functions.

All the code I wrote is in src/ folder (plus some additional code I implemented in Core/ to support Simplification, see 2.).
The Subdivision and Simplification functions are located in file src/MeshOperator.

To build the code, create a folder “build” at the root and execute cmake:
`mkdir build, cd build, cmake .., make`

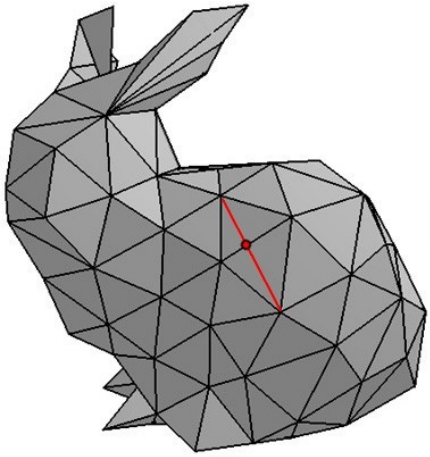
Then to run the code : `./MeshEdition`

2. Mesh Subdivision

I wrote the Subdivision code following the CMU tutorial :

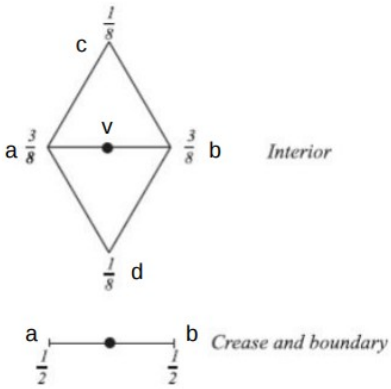
1. First, I copy all vertices and and all edges and store them in arrays. Since we want to make the loop subdivision only on the original faces and not the ones created during the process, this step is important.

2. We get the face with the maximum Id in order to use the function `edgeSplit` provided by the framework. Then, we iterate over the original edges and split them.



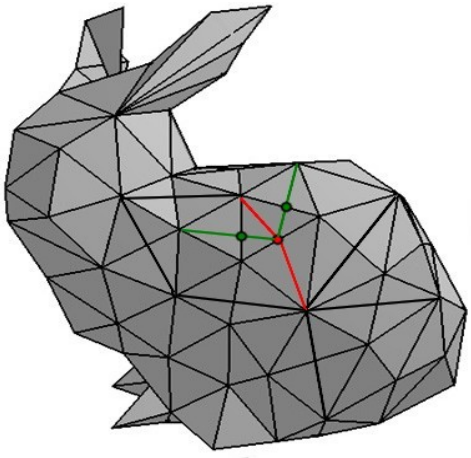
During this loop, `edgeSplit` returns the new vertex created by the Edge splitting (red point on above figure). We position the new vertex using the following formula :

- v = 1.0/2.0*(a + b) for a boundary vertex
- v = 3.0/8.0*(a + b)+ 1.0/8.0*(c + d) else



The formula is well explained slide 10 of this lesson
http://www.cs.cmu.edu/afs/cs/academic/class/15462-s14/www/lec_slides/Subdivision.pdf

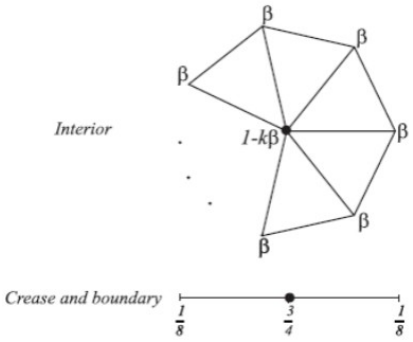
After a few iterations, we get a similar figure :



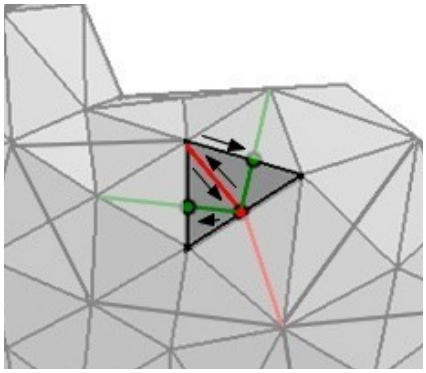
The green edges are created by splitting some old edges, however they cut newly created faces. This is why we will only need to flip only one edge (see 1.4).

We need to finish making all the edge splits before re-positioning the old edges (the ones that were not created by the splitting) because their position depends on the position of their neighbors (some of which are newly created by the splitting).

3. We loop though the old vertices and re-position them using the formula :
    - v = 1.0/8.0*(a + b) + 3.0/4.0*(v) for a boundary vertex
    - v = v*(1-k*BETA) + (sum of all k neighbor vertices)*BETA



4. We loop through the newly created edges (by the splitting) which need to be flipped. Each face will have a similar configuration:



The green edges are correctly positioned on this face. Only the red one, the first created, needs to be flipped (swapped) to being on the two green vertices, which we can easily access using the halfEdges associated with this edge. We use the `edgeSwap` function provided by the framework:

```
edgeSwap(e,
((e→halfedge(0))→he_next())→target(),
((e→halfedge(1))→he_next())→((e->halfedge(1))->he_next())->target())
```

3. Mesh Simplification

Likewise, I followed the great explanation given by CMU on Quadric Error metrics simplification
https://github.com/cmu462/Scotty3D/wiki/Simplification

In order to follow their tutorial, I made some changes in the Core framework :

- I added a class EdgeRecord in Core/Edge to keep track of the cost for collapsing the edge
- I added an attribute quadric in Core/Vertex and Core/Face

Then I could successfully implement each step in src/MeshEdition:

1. Iterate over the faces, compute quadrics for each face and store it in face->quadric(). The quadric computation detail is explained in the tutorial. We calculate the doubles (a,b,c,d) forming the plane equation ax+by+cz+d=0 of the face using the face normal and one vertex. Then we have the quadric of the face :

```
a^2  ab  ac  ad
ab  b^2  bc  bd
ac  bc  c^2 cd
ad  bd  cd  d^2
```

2. Iterate over the vertices, compute the quadric for each vertex by adding up the quadrics at all the faces touching that vertex and store them in vertex→quadric().

3. Iterate over the edges. For each edge, create an EdgeRecord and insert it into one global MutablePriorityQueue.

   This step is completed in several sub-steps:
   1. Compute a quadric K for the edge as the sum of the quadrics at endpoints.
   2. Find the optimal point by solving the 4x4 linear system Ax=b. This step is well explained the original paper page 3: https://www.cs.cmu.edu/~./garland/Papers/quadrics.pdf
   3. Store the optimal point in EdgeRecord::optimalPoint.
   4. Compute the corresponding error value = x^T K x and store it in EdgeRecord::cost.
   5. Store the edge in EdgeRecord::edge.

4. Until a target number of faces is reached, collapse the best/cheapest edge. This step is completed in several sub-steps:
   1. Get the cheapest edge from the queue.
   2. Remove the cheapest edge from the queue by calling pop().
   3. Compute the new quadric by summing the quadrics at its two endpoints.
   4. Remove any edge touching either of its endpoints from the queue.
   5. Collapse the edge.
   6. Set the quadric of the new vertex to the quadric computed in Step 3.
   7. Iterate of any edge touching the new vertex. Create new edge records for each of them and them insert them into the queue.

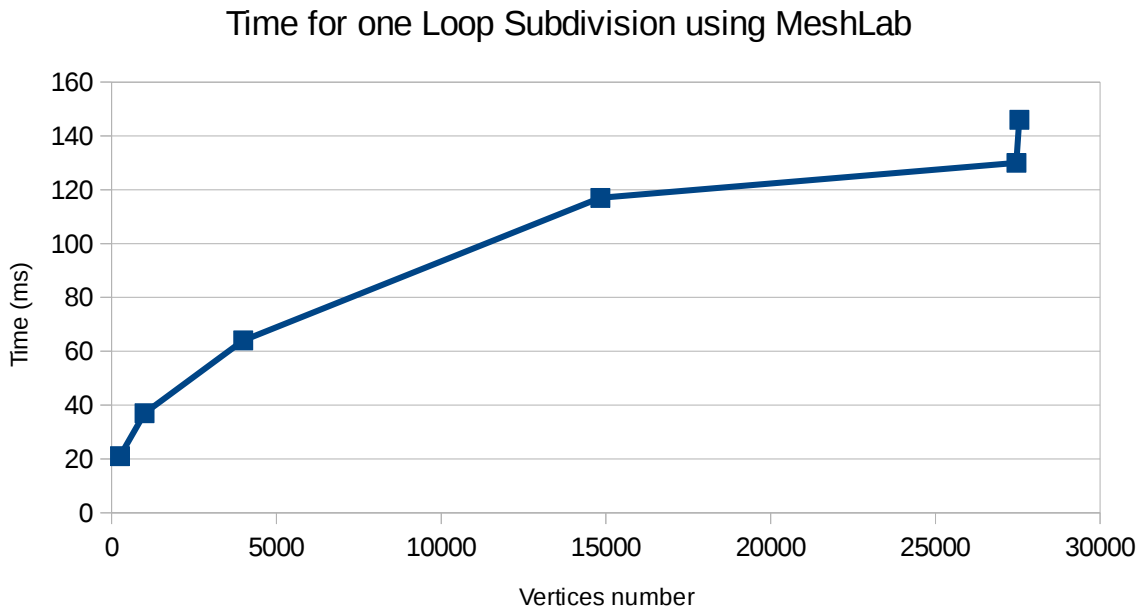
   I chose the target number of faces to be ¼ of the former number, as advised by the tutorial.

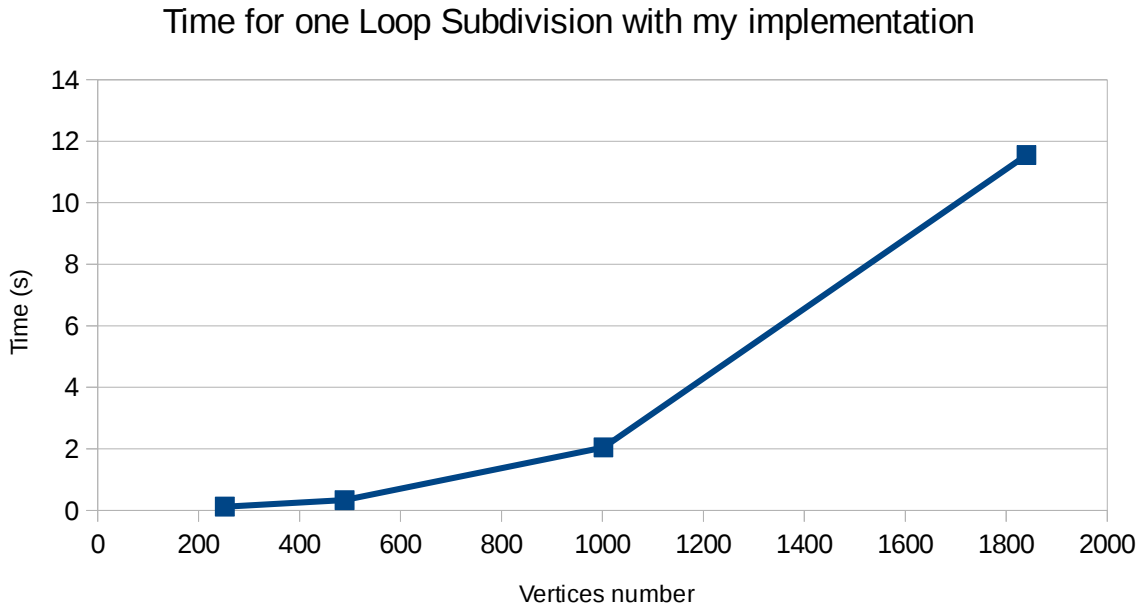
4. Analysis

I used the software MeshLab to compare my results with.

- **Time performance**

I tried performing one Loop Subdivision on meshes with increasing number of vertices using both my implementation and MeshLab.





The performance of MeshLab is excellent. The time increases logarithmically until it reaches

around 27 500 vertices, where only the time increases with very few new vertices added by the Loop Subdivision. We can consider then that 27 500 vertices is the limit of MeshLab, still the time performance is overall under 0.1 s.

On the opposite my solution is very time-consuming. The curve increases almost exponentially which makes my solution unusable for a high number of vertices. My program would freeze when trying to perform Loop Subdivision on a mesh with 4000 vertices.

Therefore **my program's Loop Subdivision performance in time is very bad** compared to the one implemented in MeshLab.

However, for the Simplification, the programs perform almost the same. I tested both of the on bunny.obj, which has 1002 vertices. My program performed the operation in 52,7 ms, compared to 51 ms for MeshLab.

Therefore **my program's Simplification performance in time is pretty good** since it can compare with a professional software.

Since Simplification performs much better then Subdivision, we can emit the hypothesis that the bad performance could come from `edgeSplit` and `edgeSwap` functions from the framework, since they are not used in Simplification.

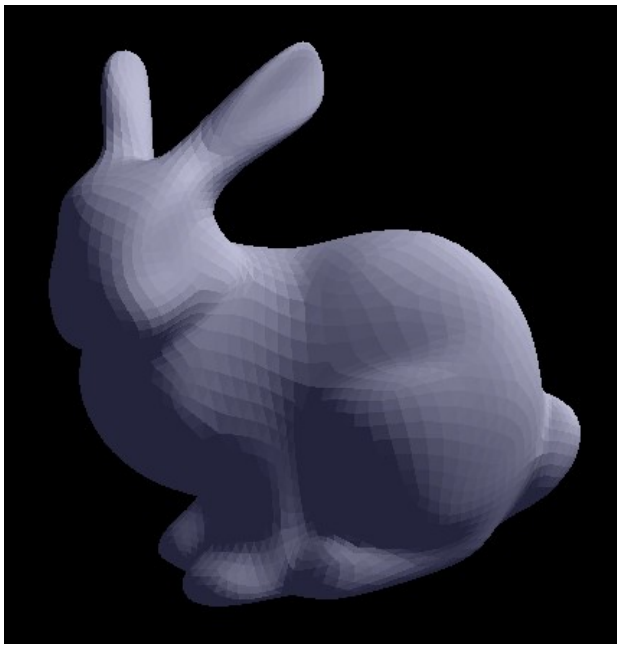
- **Space performance**

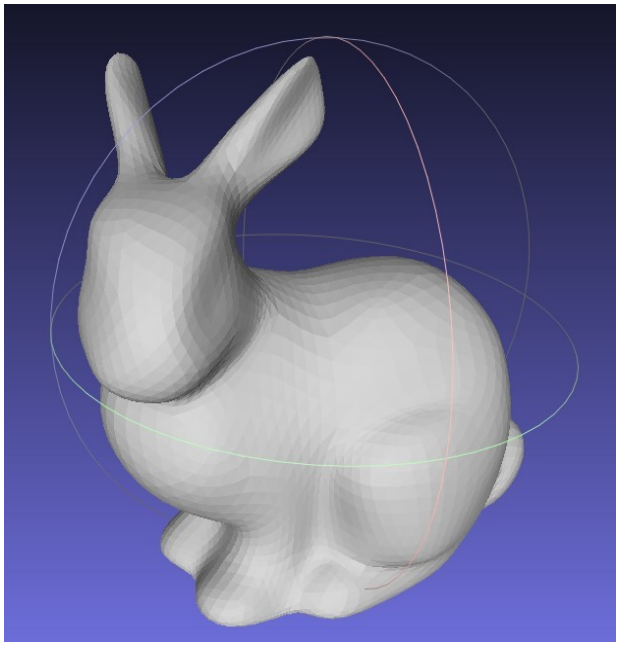
I did not manage to know the space perfomance on MeshLab, so I could not make a comparison on that factor.

- **Visual performance**

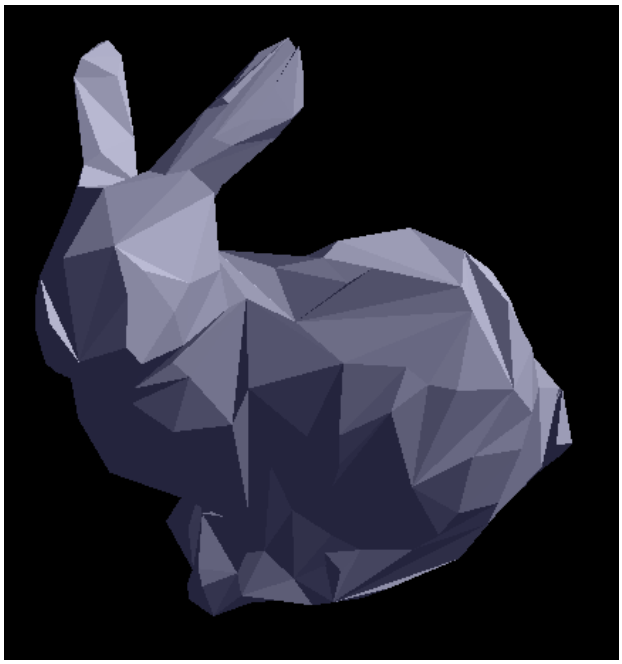
We can see that the results from my solution (left) are very similar from Meshlab. On that regard, they are very satisfying.

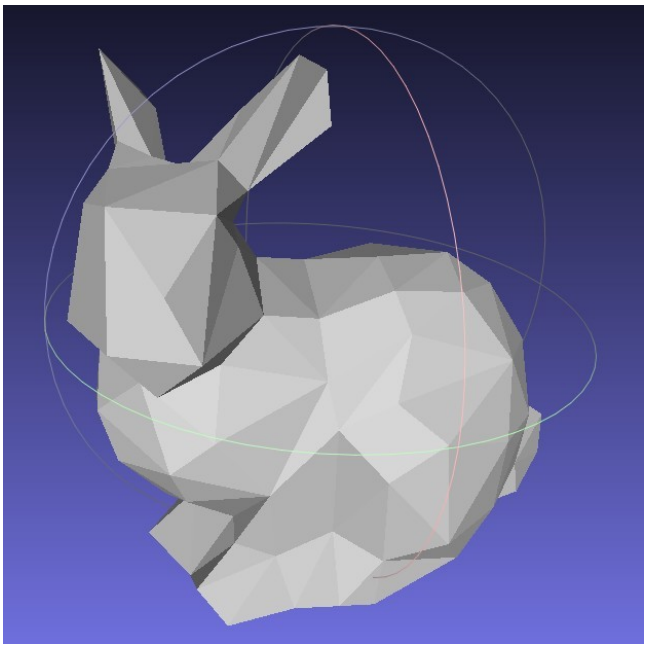
Mesh subdivision on bunny.obj





Mesh simplification on bunny.obj





5. Bonuses

On top of displaying the lighted mesh, I implemented an entire GUI for Mesh Editing :

- Rotation of the object, zoom in/out with the mouse
- Change of shading flat/smooth by pressing key “s” (I implemented Vertex normals in Core/Vertex file)
- Menu with right click for performing loop division, simplification, saving the mesh in a .obj file
- In-time information displayed on the top of the screen : number of vertices, confirmation and time for performing the operations.

A part of the GUI is implemented in src/interactionManager as in my previous projects, another part is in the main file src/main.