CS3242 3D Modeling and Animation
Assignment 1: Mesh simplification

In this assignment, you are going to implement a mesh simplifcation algorithm that can reduce the number of faces in a triangle mesh. A simple C++ code template is provided for you to start. In the code template, a simple Mesh class with basic I/O functions that can read and write a .mesh data file is given.

The .mesh data file contains two types of information[i]:

- Lines starting with 'v' specify the x, y, z coordinates of each vertex.
- Lines starting with 'f' specify the index of vertices composing each triangular face.
- Lines starting with '#' are comments.

Your task is to implement

1. `simplifyMesh(const char* input, const char* output, int faceCnt)`
   which loads a mesh from the input file, simplify it into the specified number of faces `faceCnt` and store the result to the output file. You can choose either the Melax algorithm or the Error Quadrics method.
2. implement halfedge data structure *halfedge and its* components: HEVertex, HEEdge, and HEFace.
3. `convertMesh()` and `revertMesh()`
   which translate data between *indexed face set* and *halfedge*.
4. Helper methods
   `neighborVertices(HEVertex v)` which gets a vertex's neighboring vertices,
   `neighborFaces(HEVertex v)` which gets a vertex's neighboring faces,
   `adjacentVertices(HEFace f)` which gets the vertices of a face.
   These will come handy for your later algorithm implementation.

Feel free to write extra methods and classes. Example mesh data is provided in `data/` folder. You can use the provided Viewer.exe to view the model.

Requirements

You are free to use maths libraries (e.g. GLM, Eigen, or other matrix files downloaded from other sources), but you must implement *halfedge* data structure and the simplification algorithm on your own.

This assignment is worth **20%** of your final marks, which consists of:

- *Correctness* [8%]: the *halfedge* data structure shall be implemented correctly.
- *Effectiveness* [4%]: the algorithm correctly reduces the number of faces of the input mesh into `faceCnt`.
- *Accuracy* [8%]: the simplified mesh shall be an acceptable approximation of the original.

Submission

Due date: **Tuesday, March 8, 2016, 11:59AM (noon).**

Please zip your source code as well as the compiled executables, and rename it as YOURNAME_Asn1.zip.

Hints

The data structures provided in Mesh.h is *indexed face set*. You need to implement *halfedge* to make your mesh simplification algorithm efficient. Note that this step is very important as it is going to affect the way you implement the algorithm and its performance.

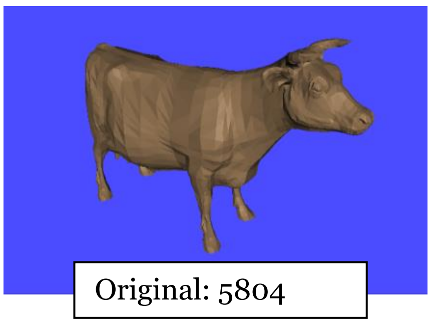
Implement one of the two algorithms discussed in class that can simplify your mesh: Melax algorithm or Error Quadrics. Make sure that any adjacency data is still maintained and consistent after performing the simplification operation.

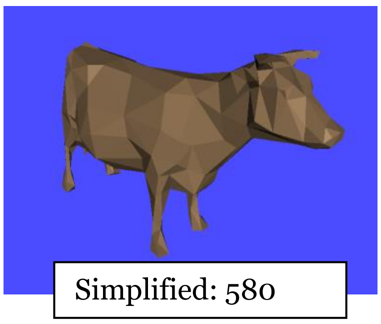
Before you implement the better algorithms to improve accuracy, you can test the correctness of your code with naive random half edge collapses, i.e., randomly pick an edge and remove it. You may want to test your implementation with a few simple meshes first.

When all the above building blocks are ready, code your algorithm and test it with the provided mesh data. With the provided meshes, a proper algorithm should take at most a few tens of seconds to finish. Remember to compile your executable in release mode.

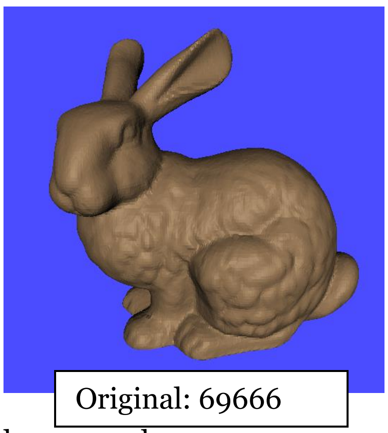
Sample output[ii] (you should try the bones example last as it is more challenging.)

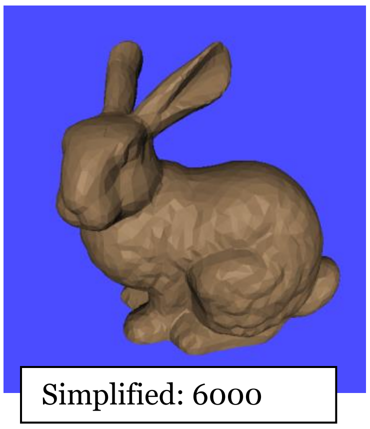
cow.mesh

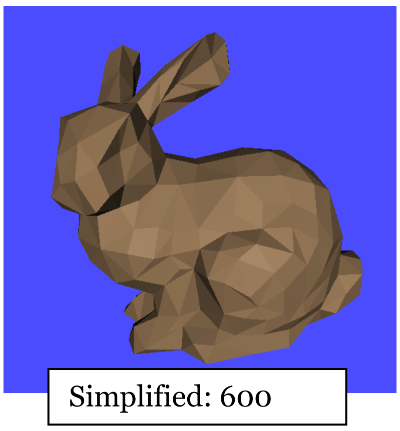




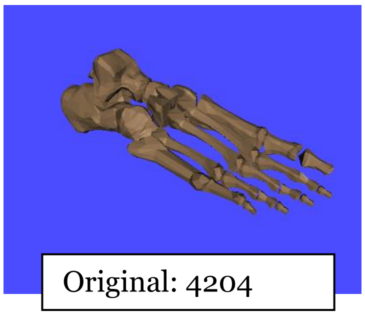
bunny.mesh

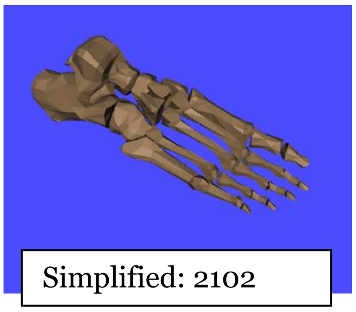


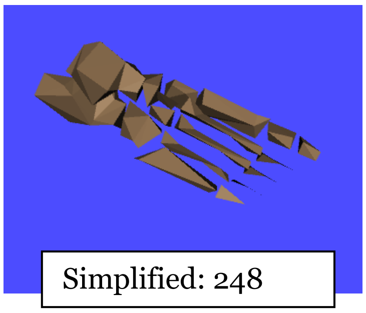




bones.mesh







[i] In fact, the .mesh file is very close to the Wavefront OBJ file format. You can change the extension to .obj and open the file in MeshLab for viewing and examining the mesh vertices and faces.
[ii] Depending on your algorithm implemented, the simplified mesh might be slightly different from the above examples.