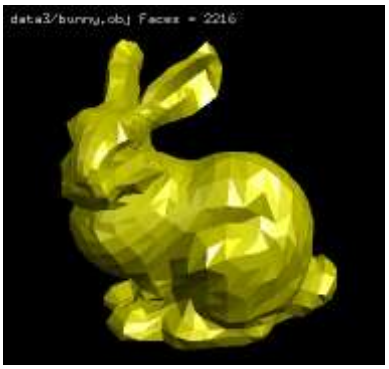
COMP 557 - Fall 2018 - Assignment 3
Mesh Simplification

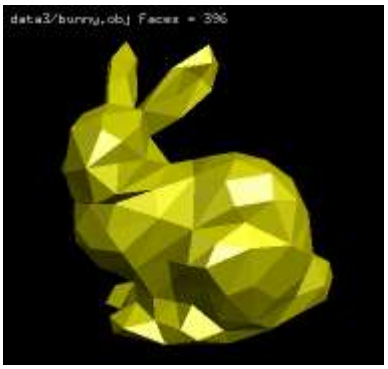
Available: Wednesday 17 October 2018

Due: 23:30 Monday 12 November 2018

Getting Started

In this assignment you will build half edge data structures from polygon soups, and simplify meshes using the quadratic error scheme described in class [Garland and Heckbert 1997]. We will only be working with triangle meshes.





The polygon soup loader has been written for you and is minimal. It will load the vertex and face data from an 'obj' file (though these files can also contain lots of other information too). We have also provided you with a number of sample triangle meshes that will be useful for testing your program.

Provided Code

Download the sample code from MyCourses. It is a working program that will load one of the sample polygon soups and will draw it. You will need to add the jogl, vecmath, and mintools jars to your project. The sample code zip file has the following contents.

- ***MeshSimplificationApp***

  The main application loads a polygon soup, will make calls to try to build a half edge data structure. The display call will draw the polygon soup mesh by default, and also draws a half edge of the half edge data structure (but only once you've written code to create this data structure). There are likewise options for drawing with back-face culling and wire-frame. You will be asked, and will find it useful, to visualize additional information in order to complete the assignment.

  There is a keyboard interface defined to let you test various parts of your implementation. You can walk the half edge data structure using space to go to the twin and N for the next half edge. Page up and page down will allow you to go forward and backward in the model list. Once you've completed various objectives in this assignment, pressing G will set the current half edge to the edge with lowest error, while pressing C will collapse the current edge, provided it does not cause the topological problems. See the control panel for the full list of keyboard controls.

- ***PolygonSoup***

  Contains a vertexList and a faceList, and code to load these from an obj file. There is also some simple drawing code used by the *MeshSimplificationApp* class to draw the loaded soup.

- ***HalfEdge***

  The half edge class is simple, and provided for you. It has a method for drawing a half edge to help you verify that your program is working. For your convenience, it also contains a *prev* method to get the previous half edge.

- ***Edge***

  The edge class is a common unique container for edge specific information common to the two half edges. It has members for storing the quadric error metric for collasping the edge, the optimal vertex position, and the

error. It also points back to one of its half edges. The class implements the Comparable interface so that you can use it with a java.util.PriorityQueue.

- ***Vertex***

  The vertex class is simply a container to hold the position of the vertex, and information for the quadric error metric.

- ***Face***

  The face class is a simple container for the face normal (for flat shading), and a pointer to a half edge. The constructor will compute the normal and set all the *leftFace* pointers of the half edge loop appropriately. Eventually, as part of the assignment objectives, the construction of a face will also compute important quantities for simplification.

- ***HEDS (you will likely start with this class)***

  HEDS is the Half Edge Data Structure, and simply contains a list of faces. You will need to implement the constructor which builds the structure given a polygon soup

- ***Sample Geometry***

  A number of sample obj files are included. All of the samples are *orientable manifolds with no boundaries*, except for the monkey head which is included as a bonus (i.e., you'll need to write code to deal with boundaries). Feel free to add your own models or modify these as you please.

Steps and Objectives (15 marks total)

1. ***Half Edge Data Structure (2 marks)***

   Write code that builds the half edge data structure. You will probably want to keep track of the half edges you create as you make them so that you can easily connect each half edge with its twin. One simple and somewhat inefficient approach would be to use *Map<String,HalfEdge>* where "i,j" is the key for the edge from vertex i to vertex j. An alternative would be to maintain a list of half edges leaving each vertex and match up the twins as you find them. You should also create the Edge objects as containers for edge specific information for mesh simplification.

   You can test that your structure is correct by using the interface that the application provides for walking the data structure (space for twin and N for next). Once you have your half edge data structure working, set the default value of the *drawPolySoup* parameter in the *A3App* to false (so that you can avoid unchecking this parameter in the interface on each subsequent run).

2. ***Edge Collapse (2 marks)***

   Implement code to collapse the current half edge when C is pressed. Given you've not implemented the quadric error metric yet, you can test by collapsing to the edge midpoint.

   Don't forget to remove the faces from the half edge data structure. After the collapse you will want to set the current half edge to be some other half edge. You may wish to choose a half edge that is adjacent to the new vertex, so you can easily walk the data structure with space and N to check that your data structure is correct.

   Note that you will later need to implement undo and redo methods, so you might want to think about how your collapse might be best compatible with these methods (i.e., make a new Vertex rather than reusing one of the two existing vertices so you can more easily restore the mesh with the old vertices).

3. ***Avoid Topological Problems (2 marks)***

   Write code to check if a collapse will cause topological problems using the following heuristics. That is, check if the 1-rings of the edge vertices have more than 2 vertices in common. You will also not want to collapse the edge if the mesh is already a tetrahedron (i.e., only 4 faces). If you do not maintain the mesh in a topologically valid state, then you will quickly encounter problems on subsequent edge collapse operations!

4. ***Quadric Error Minimum (3 marks)***

Write code to compute the quadric error matrix for faces, vertices and edges. Solve for the optimal *regularized* vertex position and error (note the regularization DoubleParameter has a default of 0.01).

To test (and demonstrate) that you have completed this objective, draw the optimal edge collapse location for the current edge (use a GL_POINT with glPointSize large enough to make it easy to see). Notice how this changes as you change the regularization parameter!

5. ***Mesh Simplification (2 marks)***

   Build a priority queue of edges based on their edge errors. When a mesh is loaded you should set the current half edge to be a half edge corresponding to the edge which is a the top of the queue.

   Fix your edge collapse code so that it updates the priority queue after the collapse. NOTE: the quadric error of the collapsed vertex must be set to the quadric error metric used for the edge! That is, for verticies i and j, Qi+Qj *without the regularization term*. Do not change the quadric metric for vertices in the 1-ring of the new vertex; even though the faces will have new planes, their current quadric error metrics will better reflect the shape of the original mesh.

6. ***Undo (2 marks)***

   Implement the undo edge collapse functions. Once you collapse and edge, there is a collection of 6 half edges that are excised from your half edge data structure. If you maintain a list of half edges you collapse, then it is possible to insert these half edges back into the HEDS. You will need to add back the two faces to the face set. You will also want to add the half edge and the Vertex to which it collapsed into the redo lists so you can later redo this collapse. This way you can leave all the quadric error metrics and priority queue in the state they were at the most simple state of the mesh, to revisit once you redo all undone edge collapses.

7. ***Redo (2 marks)***

   Implement the undo redo collapse functions. You should also have your normal collapse method call redo if the redo list is not empty. Note that the provided code already places the half edge into the undo list, but you need to write the rest of the method!

8. ***Boundaries Bonus (optional, 2 marks)***

   Extend you code to deal with boundaries, and test with the monkey head mesh. You will need to very carefully revisit your collapse edge code, as well as your undo and redo code. You will also need to create new tests to prevent topological problems. Be sure to document your implementation in your readme.txt or readme.pdf so that the TA will be aware of your efforts.

9. ***Readme File***

   Create a *readme.txt* or *readme.pdf* file to submit with your assignment. The readme should describe anything specific to your implementation, special features, or problems. Include any comments you have with respect to the assignment, and describes anything you deem to be noteworthy. Try to be brief!

Finished?

Great! Be sure your name and student number is in the window title, in your readme, and in the top comments section of each of your source files.

Submit your source code as a zip file via MyCourses. Include a *readme.txt* or *readme.pdf* file as indicated above. Be sure to check that your submission is correct by downloading your submission. You can not receive any marks for assignments with missing or corrupt files!

Note that you are encouraged to discuss assignments with your classmates, but not to the point of sharing code and answers. All code and written answers must be your own.