

August 23, 2011

MESH SIMPLIFICATION VIA EDGE COLLAPSE

A REPORT ON THE DESTRUCTION OF
THE IMAGE OF
A HORSE USING COLLAPSING EDGES

WIM LOOMAN (*92692734*)
wgl18@uclive.ac.nz

This report details the implementation and testing of a Mesh Simplification algorithm. The specific algorithm chosen for simplifying the mesh was edge collapse, where an edge has its two end points collapsed together into a single vertex.

1.1 Edge Collapse

Edge collapse is on its face a very simple algorithm, an edge is chosen based on a set of error metrics, one end point of the edge is moved halfway towards the other then the edges extending from the unmoved end point are updated to be connected to the moved end point. The error metrics chosen have a very large effect on how much deformation of the mesh results when some fraction of the edges are collapsed, a good set of error metrics will initially choose edges that are going to produce the least change in the model. The flip side of this is that the better the error metrics are the longer it is likely to take to calculate.

1.1.1 Error Metrics

The simple error metric used was mainly based around two key variables. The length of the edge and the angle difference between the normals of the faces on each side. The length of the edge is relevant because the longer the edge the further the endpoint is going to be moved. The difference between the normals is used as an indicator of the curvature of the local area, the greater the angle the more curved it is assumed that the area is. Once these are found they are simply added together.

The next error metric was based on the one described by Stan Melax [1] developed for his work at Bioware. This also has a component based on the length of the edge and one based on the local curvature, however the local curvature component is based on a larger area of the surface to be collapsed.

2.1 Model Storage

The model was stored in a structure based off the standard half-edge data structure. Each model is stored as a series of edges, vertices and faces that cross-reference each other. Each face knows one edge that is a part of it; each vertex knows one edge that points to it and its point in space; and each edge knows the vertex it points to, the face it's part of, the next edge around the face, the previous edge around the face and the edge that points in the opposite direction to it along it's edge. It is this storage of each real edge in two *edge* structures that gives rise to the name half-edge.

Along with this standard data required to completely describe a mesh in a half-edge structure these structs contain some additional information. For performance reasons faces contain a memoized copy of their normal and edges contain a memoized copy of error metric result. Obviously when an edge gets collapsed it will affect the normals and error metrics of the faces/edges around it so the two-ring of the collapsed edge has these values reset to be calculated again.

An additional bit of information contained within all the structs is the deletion status of the face/edge/vertex, this is primarily used to simply determine which of the faces to draw during the display and which edges to consider for collapse, but is also used as a sanity check during certain operations to ensure deleted edges/vertices are never encountered during the normal processing.

2.2 Model Loading

The only format supported to load models from is the Object File Format (OFF), this is a simple list of vertices and faces composed of those vertices. This means reading in the majority of the data required is very simple, load the vertices into an array then for each face add a series of edges representing the face to the edge array and the face to the face array.

One limitation of the simple edge collapse algorithm used is it only supports triangular meshes. This isn't much of a problem as any mesh can be represented as a triangular mesh, so the loading code simply complains and quits if any polygon with more than 3 vertices is detected.

Once the initial set of vertices, edges and faces is loaded from file the rest of the data in the half-edge structure has to be calculate. The vertex and face data is fully contained in

the OFF format along with the edges previous and next pointers, the edges do however need their pair edges calculated. This is done by simply iterating through the edge list and finding two edges (a and b) where the vertices before and after one (a->vert and a->prev->vert) are the same as the vertices after and before the other (b->prev->vert and b->vert) respectively.

This takes care of the majority of the cases, however the OFF format doesn't contain any information about boundary edges so for those edges that don't have a pair found we must construct a new edge and set that as the edge, this pair will have its face set to NULL to denote that it is a boundary edge and will have to have both its previous and next edges (both also boundary edges) found. To find these the vertices at either end of the new edge are rotated around (widdershins for the vertex the edge points at and clockwise for the other) until either another boundary edge is encountered or there are no more edges. If another boundary edge is encountered then the two are joined as next and previous, otherwise it will be joined later when the missing boundary edge is added.

2.3 Error Metrics

Before actually calculating the error metrics a few simple tests are performed to weed out edges we don't care about. First the memoized value of the edge is checked and returned if it is valid, after the first iteration this will be set for the vast majority of edges and can be easily returned. Next the edge and its pair is checked if their face is NULL, if so it is a boundary edge and is skipped. Next the one ring neighbourhood of both end points are checked for any boundary edges and if found the edge is skipped. Some of these edges are collapsible, but there are a lot of special cases that aren't and others that require additional work to collapse so it is easier to just ignore this set of edges.

Finally the intersection of the one ring neighbourhood of the end points is checked to ensure it only contains 2 vertices, otherwise the removal of this edge will cause at least four other triangles to be collapsed into lines.

Both error metrics only needed two values calculable, the length of an edge and the angle between two face normals. The length is trivial to find, it is simply the length of the vector between the points at the vertices of the edge, this was taken care of by the geometry library. The angle between the two face normals was similarly simple to calculate by using the relationship $\theta = \cos^{-1}(m_1 \cdot m_2)$ where m_1 and m_2 are the unit normals of the two faces. To simplify the calculation of this angle a mapping of $o = \frac{1 - m_1 \cdot m_2}{2}$ was used, this provides o which follows a similar shape as θ and has output between 0 and 1 as seen in Fig. 2.1.

For the simple error metric the length of the edge was simply added to the angle between the two faces on either side of the edge.

For the error metric described by Stan Melax there was a bit more processing involved. As stated in his paper the cost formula was:

$$\text{cost}(u, v) = \|u - v\| \times \max_{f \in T_u} \left(\min_{n \in T_{uv}} \left(\frac{1 - m_f \cdot m_n}{2} \right) \right)$$

Where u and v are the end points of the edge, T_u is the set of faces that contain u and T_{uv}

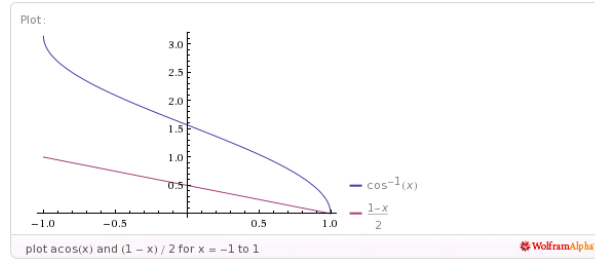
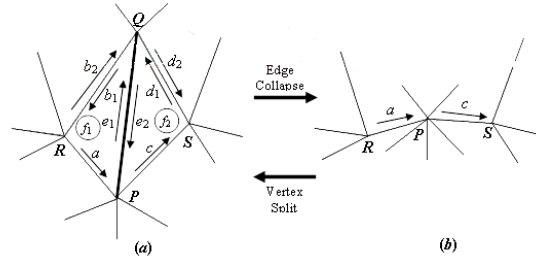
Figure 2.1: θ against o in the range $m_1 \cdot m_2 = [-1, 1]$ 

Figure 2.2: The edge collapse operation

is the set of faces that contain both u and v . In all cases that we care about T_{uv} contains only two faces, those adjacent to the edge \overline{uv} . So to calculate this we simply need to iterate around all edges terminating at u and find the minimum of the angle between the normals of the corresponding face and each side of the \overline{uv} edge. The maximum value from this iteration is then taken and multiplied by the length of the edge and used as the cost.

2.4 Edge Collapse

Once an edge to collapse has been found the actual edge collapse must happen. The identifiers used in the next section refer to those in Fig. 2.2.

The first operation is to connect a and c to the other vertices in their new faces, this is done by setting a 's previous and next to b_2 's previous and next and similarly for c and d_2 . One special case is when b_2 and d_2 are part of the same triangle, in this case a 's next is set to c and c 's prev to a .

Next if P 's associated edge was e_2 then we set it to a to ensure $\mathbf{p} \rightarrow \mathbf{edge}$ will be usable in the future. We then traverse the edges of Q widdershins between d_1 and b_2 and set them to point at P instead.

We then connect the next and previous of b_2 and d_2 to point a and c respectively to finalise those triangles connections. The faces associated with the triangles along with R and S also have their edges set to a and c to ensure they point at current edges.

Then depending on the error metric used P is either moved to the midpoint of the line between P and Q or left where it is.

Finally the edges $e_1, e_2, b_1, b_2, d_1, d_2$, the faces f_1, f_2 and the point Q all have their deleted flag set to ensure they won't be used in future operations.

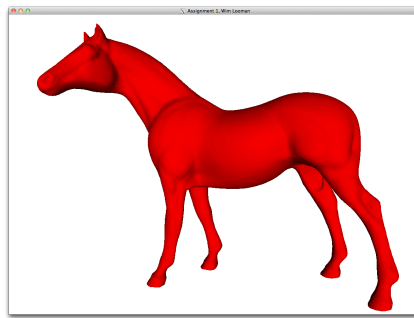
The reference image used for these results is Horse from the Shapes Repository [2]. This is a fully connected mesh with 59547 edges, 39698 faces and 19851 vertices.

The results of the two error metrics are presented below. Figure 3.1 shows the results of applying both error metrics to the model 3 times reducing the number of polygons by 50% each time. Both models are decent at 50% polygon reduction. The simple error metric has produced a lot more obvious deformities, but it would be good enough for use with a nice normal map. After removing another 50% of the polygons however the story has changed a bit. There are a lot of obvious deformities introduced in the legs of the horse especially. These are likely the result of the very simple local curvature check that is performed, the angle across the single edge will be very small but between one of those triangles and the next will be large.

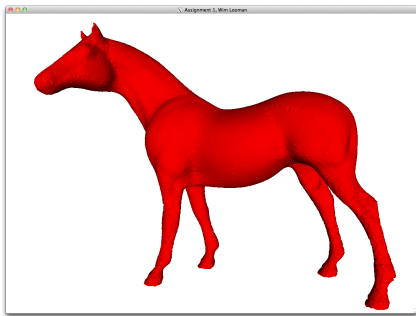
Finally by only 12.5% of the polygons remaining the model is horribly deformed (Fig. 3.1f). The error metric is obviously far too simplistic when compared to the Stan Melax's error metric's model at the same polygon level.

Figure 3.2 then shows the results of applying Stan Melax's error metric another 3 times, reducing the number of polygons to 50% then to 25% the next two times. Alongside these it shows the image with a pre-calculated normal map added. The results shown here are quite impressive, at only 6.2% of the original polygons the model is still very well rendered with flat shading and only has a slight loss of muscle definition when normal mapped. The biggest differences are around the nose and eyes, very high detail areas that would have had most of the detail destroyed to get to such a low polygon count.

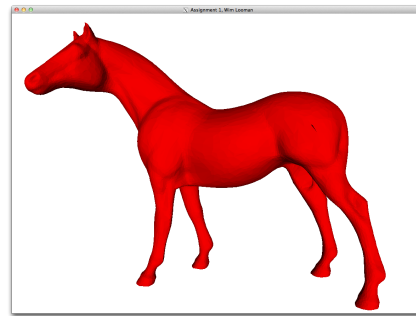
Even with over 98% of the polygons removed the model is still quite recognisably a horse. It even still has recognisable calfs and ears. Once we reduce the model down to 0.4% of the initial polygons the shape is mostly gone. It is still recognisable as a four legged animal but exactly what it is is almost impossible to guess. This level of detail is probably too low for use as anything but a very long distance model in any modern graphics application. The 618 polygon model with normal map and an applied texture however could still be useful as a long distance low fidelity model.



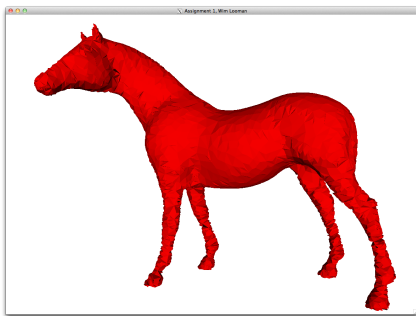
(a) Initial Model - 39698 polygons - 100%



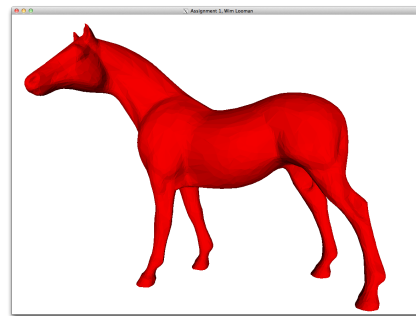
(b) Simple Error Metric - 19848 polygons - 50%



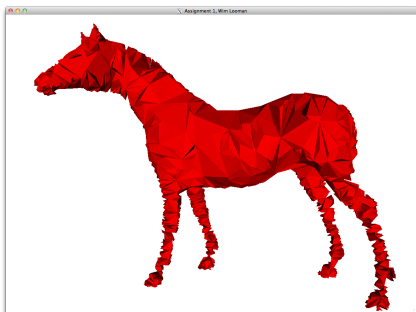
(c) Stan Melax's Error Metric - 19848 polygons - 50%



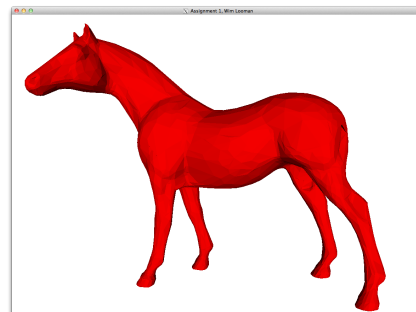
(d) Simple Error Metric - 9922 polygons - 25%



(e) Stan Melax's Error Metric - 9922 polygons - 25%

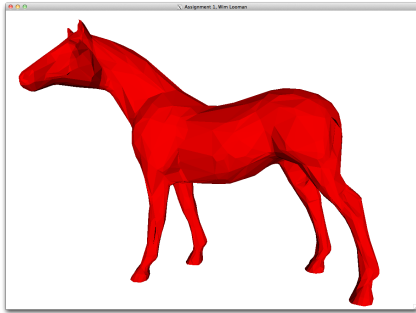


(f) Simple Error Metric - 4960 polygons - 12.5%

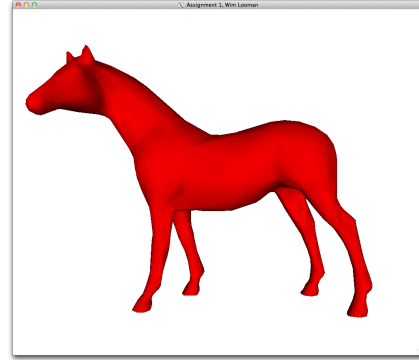


(g) Stan Melax's Error Metric - 4960 polygons - 12.5%

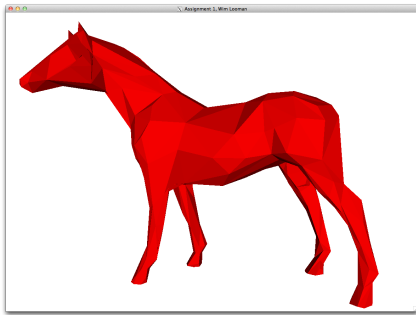
Figure 3.1: Comparison of the mesh simplification



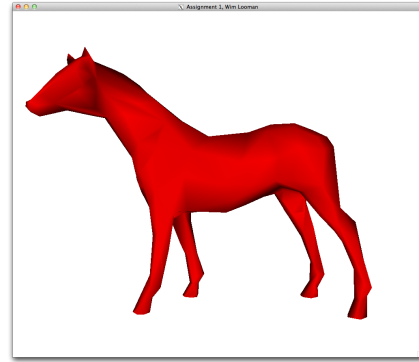
(a) 2478 polygons - 6.2%



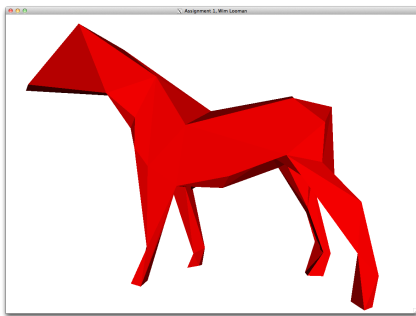
(b) Normal mapped - 2478 polygons - 6.2%



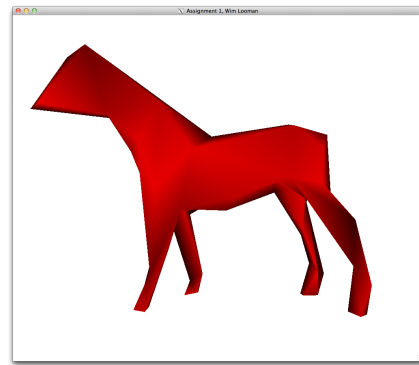
(c) 618 polygons - 1.6%



(d) Normal mapped - 618 polygons - 1.6%



(e) 152 polygons - 0.4%



(f) Normal mapped - 152 polygons - 0.4%

Figure 3.2: Extreme simplification with Stan Melax's Error Metric.

CHAPTER 4

Code Description

4.1 Modules

geom This is a rewrite of the 3D python geometry library I used last year for the second COSC363 assignment. Just simple 3D geometry stuff to make working with points and vectors easier.

view This module mainly takes care of the display and reshape functions and setting up what display type is used for the models along with all the OpenGL initialisation.

player This module simply represents the cameras viewpoint and handles moving this around when the movement keys or mouse are used.

controller This module takes care of calling all the other initialise functions then takes the input and converts it to the correct function calls.

model This module is the most important part of the assignment. It does all the actual model and edge collapse processing. Some parts should probably be split out into the actual half-edge data structures but at the moment they're all just dumb data-stores.

4.2 Controls

All keys listed below must be lowercase when pushed.

Looking Hold down the left mouse button and use the mouse to look around.

Navigation Use *A*, *W*, *S*, *D*, *Z* and *the spacebar* to move the viewpoint left, forwards, back, right, up and down respectively.

Other

B Toggle back-culling of the polygons. Mostly useful when displaying in wireframe mode.

X Choose a single edge and collapse it.

Q Collapse edges every 50ms until only 50 remain.

E Toggle showing the collapsed model in wireframe mode.

R Toggle showing the original model in wireframe mode.

F Toggle which error metric to use, starts at simple and changes to Stan Melax's.

T Toggle whether to show the original model.

P Reset the collapsed model to the original.

K Calculate the current smooth normals of all vertices.

L Toggle whether to display the model with the pre-calculated smooth normals.

BIBLIOGRAPHY

- [1] Melax, S. (1998). A simple, fast, and effective polygon reduction algorithm. Game Developer Magazine, (November):4449. <http://www.melax.com/gqmag.pdf>
- [2] Inria (2004). A horse model. This is a typical example model for LSCM parameterization algorithm. Aim@Shape Project - Shape Repository. <http://shapes.aim-at-shape.net/view.php?id=31>