Minerva Pillai
CS284 Final Project Report

Mesh Simplification for Improved Adjustability

Introduction

With the advent of 3D scanners, it is possible to create very large triangle dense 3D models of objects. If a feature of this dense model is to be modified, it could require manipulating hundreds of points. It would thus be useful to have some mechanism to change a feature on such models in a non-tedious manner i.e. manipulate 3 or 4 points rather than 100 points. I propose that this can be done, by first simplifying the mesh into a coarse mesh , then modifying the feature of interest and then subdividing the modified coarse mesh to obtain mesh similar to the original dense mesh.

Prior Work:

Mesh Simplification

Many mesh simplification algorithms have been developed for a variety of purposes such as Level of Detail approximation, ease of storage etc. Below I list of few of these

Progressive Meshes13[1]: This algorithm performs edge collapses to reduce a mesh. The collapsed edges are calculated by using an optimization algorithm. The collapsed edges are found based on minimizing energy. This energy is a function of the distance between points, a "spring force" keeping the edges close to their original position and energy associated with some scalar attributes.

Decimation of Triangle Meshes[4]: This algorithm first scans the entire mesh to characterize the local vertex geometries and topology. The vertices are grouped into simple, complex, boundary, interior or corner vertex. The decimation is performed on a vertex whose distance to an average plane is within a threshold. For boundaries the distance to an edge is measures. The vertex chosen and all the triangles associated with it are removed. The hole in the mesh is filled using a local triangulation.

Quadric Error Metric[2][3]: The quadric error metric is a quick and cheap algorithm to perform mesh simplification by performing an edge contraction. As this is the method that I implemented for my algorithm, I shall explain it in more detail.

The algorithm first picks all the vertex pairs than form an edge. The decision to contract a pair is based on minimizing the error caused by the removal of the vertex. The error of contraction is related to the sum of the squared distances to the planes connected to the points in question (all the triangles incident to a point). This is given by equation (1)

$$\Delta(v) = \Delta([v_x \quad v_y \quad v_z \quad 1]^T) = \sum_{p \in planes(v)} (p^T v)^2 = v^T (pp^T) v \Rightarrow v^T \left(\sum_{p \in planes(v)} K_p \right) v \qquad \textbf{(1)}$$

Where p=[a b c d]^T represents the plane defined by ax+by+cz+d=1;

And $K_p = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$

For each face connected to the vertex in question, a fundamental error quadric K_p is used to compute the distance between any point in space to a plane p. The sum of all the K_p is represented by a single matrix Q. The contraction pair with the lowest value of Δv is contracted. The new point that it replaces is calculated using the matrix in equation (2)

$$\bar{v} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{14} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (2)$$

The actual minimum that is calculated for each potential contraction is given equation (1) where v is replaced by $\bar{v}$ in equation (2). Once the minimum pair is found, v_j is removed and all the edges associated with v_j are moved to v_i. The errors are recalculated and the process is repeated.

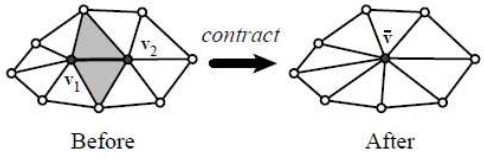

Figure 1: **Edge contraction.** The highlighted edge is contracted into a single point. The shaded triangles become degenerate and are removed during the contraction.

Each edge contraction should cause 2 triangles to disappear. In [3], Garland adds more detail to the algorithm described in [2]. To ensure proper contraction for a plane with triangles of different sizes, he suggested implementing an area weighted error quadric. In addition he suggests certain consistency checks and constraint additions to prevent mesh inversion, preserve boundaries and prevent the formation of sliver triangles.

Subdivision

Subdivision is a well explored subject. There are many algorithms that exist that perform different types of subdivision (interpolating vs approximating subdivision schemes) on different types of mesh structures. Most of these were discussed during the course. Below I discuss some subdivision methods that can be used on triangle meshes.

Loop Subdivision[6]: Loop subdivision is an approximating scheme to subdivide triangle meshes. Loop applies a weighted mask onto the triangle mesh and creates new coordinates for the existing vertices as well as places new vertices on the edges of the triangles.

Zorin Subdivision[5]: Zorin subdivision is an interpolating scheme to subdivide a triangle mesh. It uses a modified butterfly mask to create new vertices on the edges of the triangles.

Project Implementation

Initial Proposal

- Implementation language Matlab
- Use Progressive meshes edge collapse to simplify/minimize the mesh.
- Once simplified the object can be modified
- An optimization algorithm is used to modify the coarse mesh further such that a subdivision will cause the new mesh to reobtain features from the original dense mesh.
- Zorin subdivision is used for the subdivision

Updates in Implementation

- Quadric Error Metric [1] was used from simplification

As I picked matlab for my implementation, I was very concerned with the run time of the code. The Quadric Error Metric is designed to be a fast mesh simplification algorithm. The quadric error metric simplification also had a simpler data structure to implement.

I broke the project into 2 different steps

I. Implementation of Zorin Subdivision:

Zorin subdivision uses a modified butterfly mask to find new vertex points on a mesh. Most of the frame work for the Zorin subdivision was already implemented during the work on the subdivision assignment. I used a half edge and vertex data structure to simplify the subdivision.

Vertex list

- Coordinates
- HalfEdge Pointer
- Valence

Half Edge Structure list

- Vertex Pointer
- Half Edge Pair Pointer
- Half Edge Next Pointer
- Flag (flag of weather it has been subdivided)
- Flag Value (new vertex pointer value)

Some results from this subdivision algorithm are shown in Figure 1 and Figure 2.

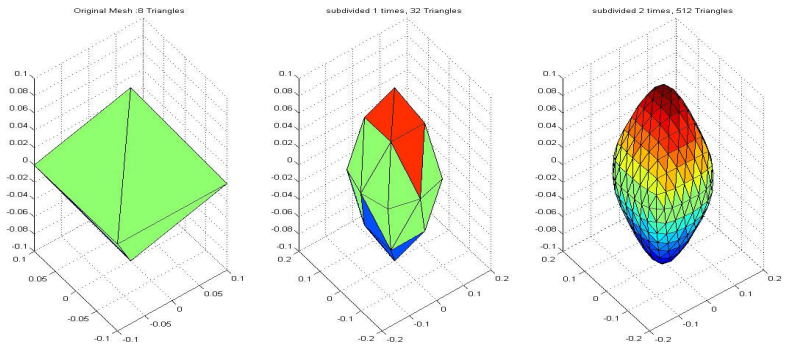

Figure 1: Zorin Subdivision on an octahedron

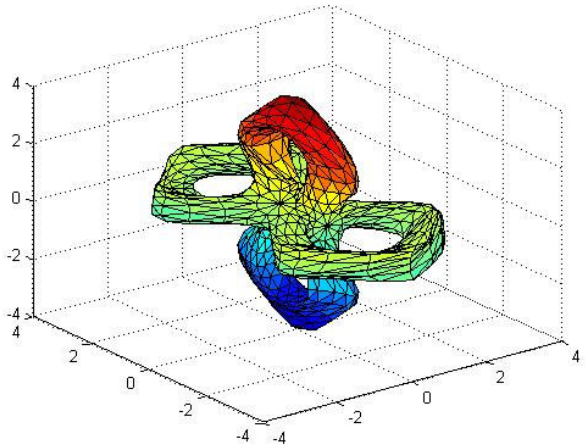

Figure 2: Two Zorin Subdivisions on a genus 4 object

II. Mesh Simplification

Originally I implemented a half edge structure to perform the mesh simplification, however the simplification was unsuccessful due to multiple referencing of the same half edges when reconstructing the triangles. After input from the professor, I used a vertex only data structure which proved to be better. Each vertex stores a list of other vertices it is connected via edges. The vertices were stored in order, i.e. if vertex v1 with a vertex list i, i+1, i+2, i+3…i+n, then v1, i and i+1 are the vertices of 1 triangle and v1, i+1 and i+2 are another triangle and so on with the last triangle looping around(v1,i+n,i)

Data Structure

Vertex list

- Coordinates
- Vertex list

The structure of the final algorithm I implemented is a follows

1) Read in triangle and vertex coordinates
2) Setup data structure
3) Calculate triangle areas and K_p matrix
4) Select candidate pairs v_i and v_j such that v_i and v_j are an edge pair.
5) Allocate an area weighted quadric Q to v_i and v_j
6) Use quadric to calculate new vertex point
7) Use new vertex and error quadric to calculate minimum error for every edge contraction
    a. Check for mesh inversion if mesh inversion occurs add error value to minimum error
8) Put errors in a heaped queue with minimum at top. Remove v_j corresponding to minimum error
9) "Connect" all point associated with v_j to $\bar{v}$
10) Replace v_j with $\bar{v}$
11) Quadric Error for $\bar{v}$: Q is the sum of the v_i and v_j quadric errors
12) Repeat simplification process until number of faces is attained

Results and Discussion

During the course of my implementation I obtained several different results by implementing some of the constraints that Garland suggested versus just the error metric itself. These results are discussed below

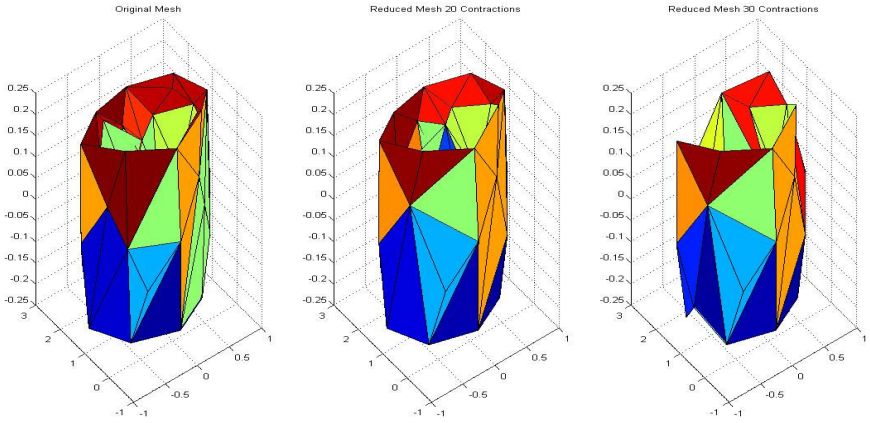

Figure 3: Genus 1 object simplified with 30 contractions. Only the basic error metric was implemented

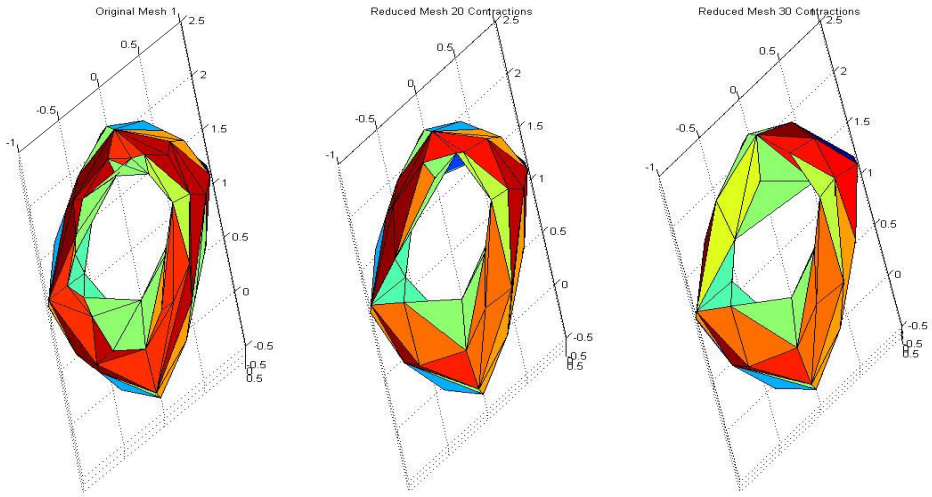

Figure 4:Genus 1 object simplified with 30 contractions. Only the basic error metric was implemented (Top View)

As can be seen after 30 contractions, the model begins to degrade. In addition certain overlapping behavior was observed. These results were not as good as desired so a mesh inversion constraint was included in the code. The constraint stated that for the new vertex must lie within the same region of the plane created by every edge opposite the vertex except for those in comment with the contraction pair. If the new vertex is falls outside this realm, a high error was added to the contraction. This is graphically represented in Figure 5. It is important to note that instead of eliminating the contraction completely, we only increase the weight, by doing so we ensure that the contraction process wont just stop.

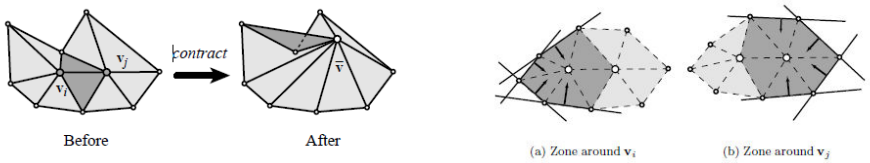

Figure 5: From [3] represents the idea used in qslim to avoid mesh inversions

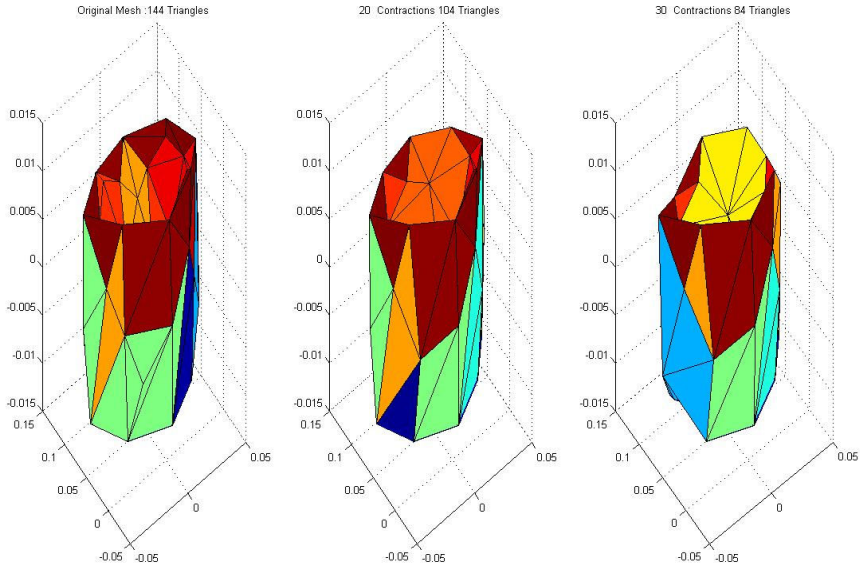

Figure 6: Genus 1 contraction with mesh inversion constraint

In Figure 6, the mesh inversion constraint was applied to the initial quadric and is not recalculated after the edge contractions begin. As can be seen the results are much better than those in Figure 4. Figure 7 depicts the comparison between only an initial inversion constraint (inversion constraint is added before any contractions are performed) and a complete constraint (also reevaluates the mesh inversion constraint after every mesh contraction).

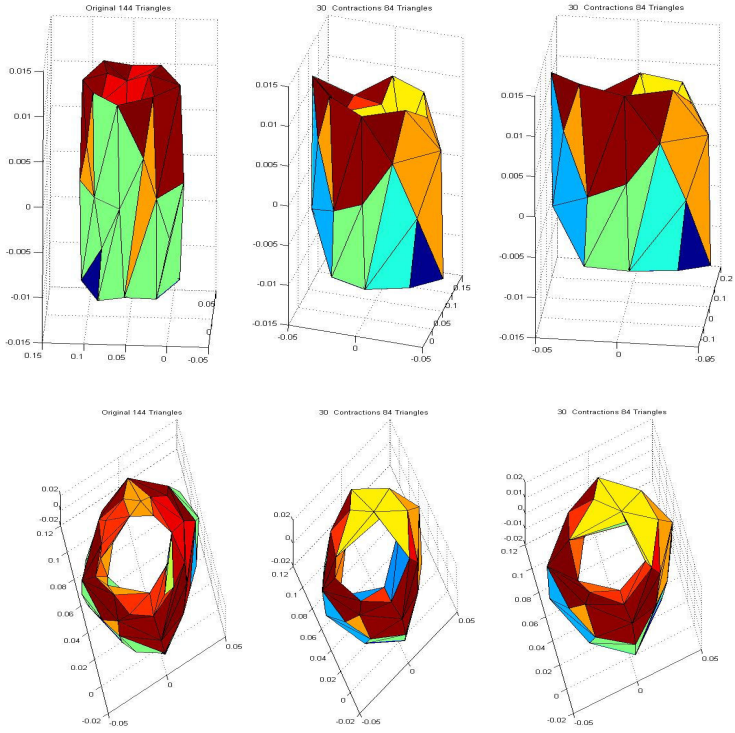

Figure 7: Genus 1 original object is on the left, only initial inversion constraint result(middle) ; Completely inversion constraint (right)

As can be seen the result for both cases are similar, however as seen in Figure 8, as the number of contractions increase, the complete inversion constraint code appears to give a cleaner result.

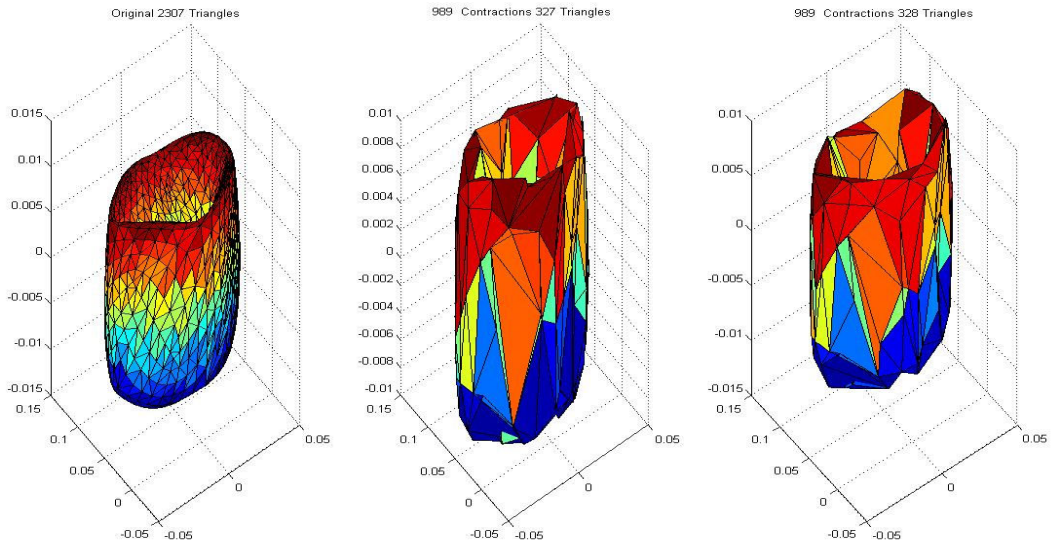

Figure 8: Genus 1 mesh simplification implemented with initial inversion constraint(middle) and complete inversion constraint(right)

This is expected behavior as we would expect the likely hood of new mesh inversions to increase as the simplification process continues.

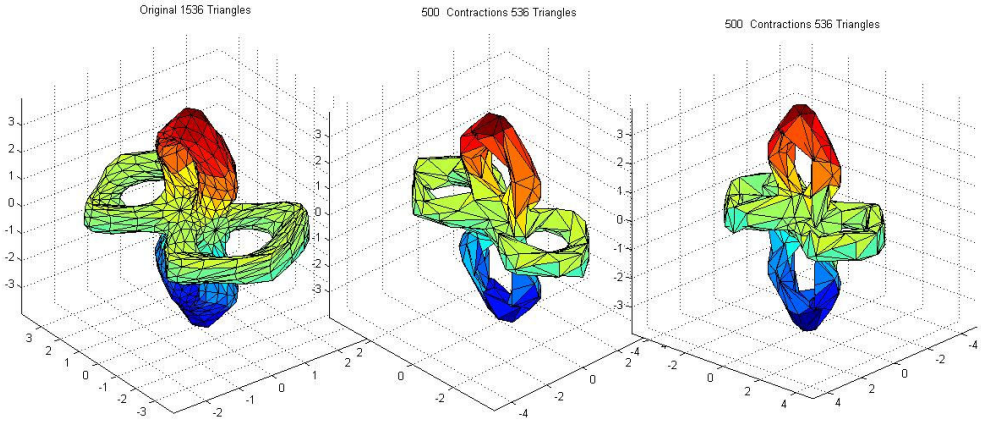

Figure 9: Reduced Genus 4 Object Original(right), Initial mesh inversion only(middle) ; Complete mesh inversion constraint applied(right)

Qslim versus Implemented Algorithm

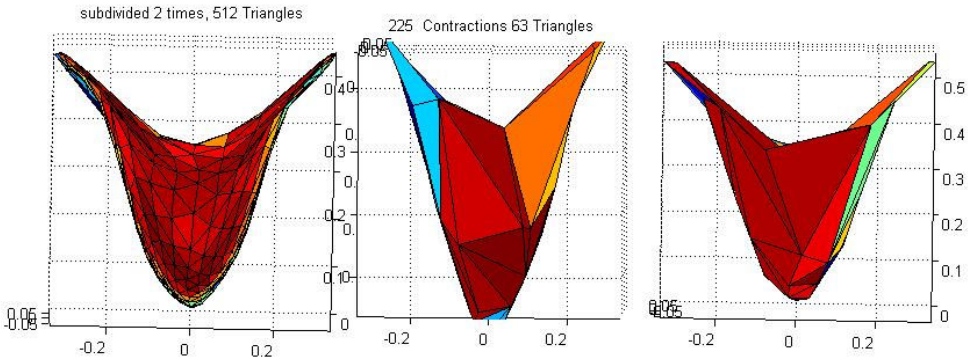

Figure 10: "Face" Model simplified; Original(left), Implemented(middle), Qslim(right). The Qslim and implemented case have the same number of faces

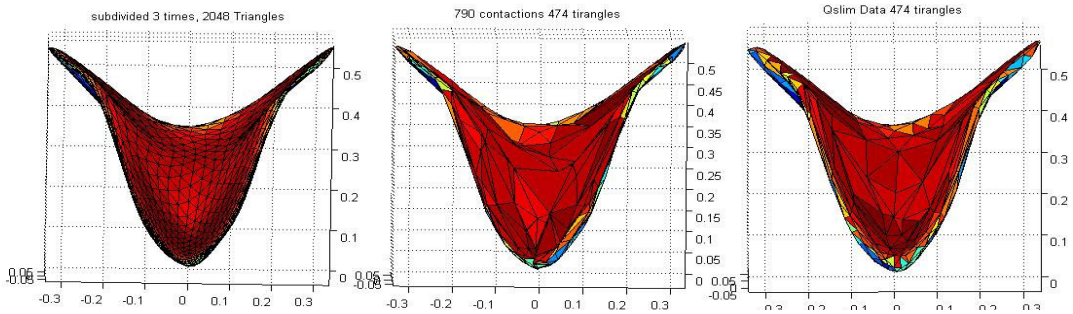

Figure 11:"Face" Model simplified; Original(left), Implemented(middle), Qslim(right). The Qslim and implemented case have the same number of faces

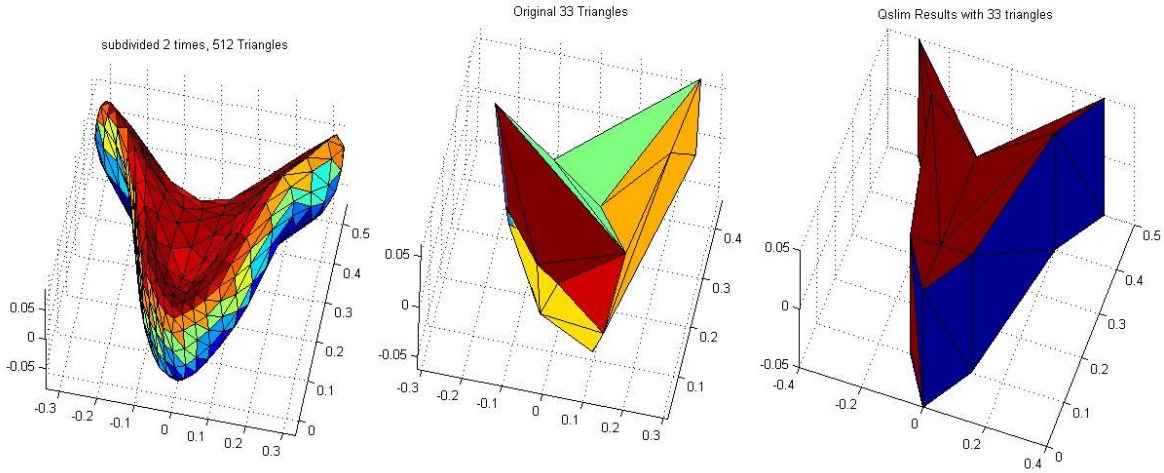

Figure 12:"Face" Model simplified; Original(left), Implemented(middle), Qslim(right). The model was reduced to the size of the original object that was subdivided to form the image on the left. Qslim very clearly does a great job in recreating the object

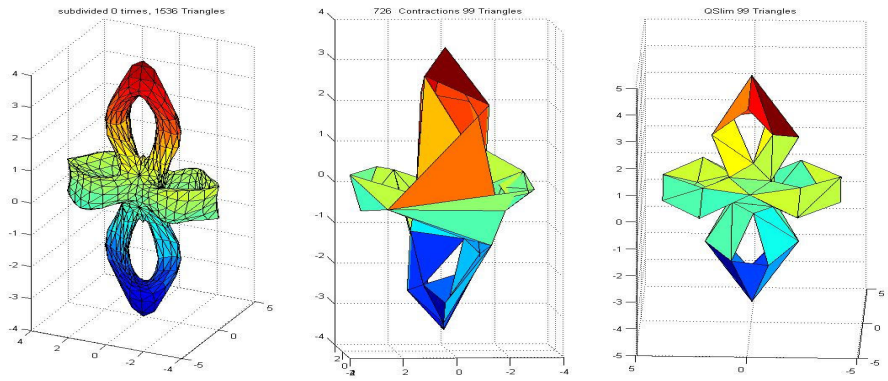

Figure 13:"Genus 4" Model simplified; Original(left), Implemented(middle), Qslim(right). The Qslim and implemented case have the same number of faces

Figure 10, Figure 11 and Figure 13 show the results from Qslim and my implementation are comparable to a certain point. When the mesh is broken down into its coarsest form, my implementation loses a lot of the detail, which Qslim retains. There are several reasons for this discrepancy. The sliver triangles and boundary constraints are not implemented in my algorithm. While most of the framework for the sliver triangle error implementation has been created, I didn't have time to completely implement it. I misinterpreted that in [3] the boundary constraint implementation is a manual process. The thesis more specifically states that one can create boundary constraint points that can be manually picked, however after examining the source code for the qslim program and further reviewing the implementation, I realized that a version of boundary constraints was implemented in the algorithm.

Even thought the mesh inversion error is applied in the algorithm, we can still observe instances of mesh inversion. In addition, the code sometimes becomes degenerate and errors out. Delving the errors show that the vertex list, which is used to keep track of triangles is wrong. These errors compound as the mesh simplification steps increase, which is why the errors rarely appear while simplifying a small mesh. These errors could be on account of 2 possible reasons,

1) There is an error in the vertex contraction mechanism. More specifically, the mechanism to reallocate vertices after a vertex is removed is flawed. To check this I created 8 test cases where I varied the position of contracted vertex and the altered vertex inside the vertex list. The placements of reallocated vertices are highly depended on these positions. For all the test cases the reallocation mechanism successfully placed the vertices. This can thus only be the cause of the error if an unknown case not tested occurs.
2) While in the general case an edge contraction must always result in the removal of 2 triangles, this is not always the true. For both my implementation as well as Qslim, there appear to be cases where an odd number off triangles get contracted. This could cause some errors in my vertex reallocation mechanism. The errors appear soon after this case is observed. Thus some correlation is hypothesized.

In addition there are certain unfavorable aspects to using the quadric error metric simplification for my application. As I intend to subdivide my reduced mesh, I would require my reduced mesh to be a manifold object. The quadric error metric simplification algorithm is designed to work on non- manifold models as well. Due to this the algorithm has no problems with creating singular triangles to represent features. A clear example of this is in the 100 face cow image Figure 3.7 in [3], where the ears and legs of the cow are represented by single triangles Figure 14. This would not create good results when a subdivision is implemented. The Qslim algorithm also tends to reduce the size of the initial mesh on occasion. This was observed for the Genus 1 torus example. In order to counter the errors and the limitations of Qslim, I proposed some additional implementations in the future work section of this paper.

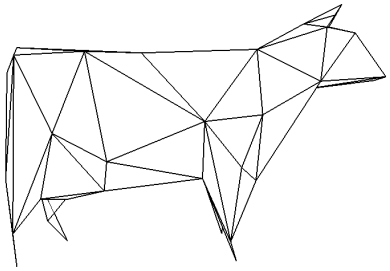

Figure 14: Qslim simplified cow at 100 faces

Future Work

It is important that the algorithm work in all cases, thus it is important that the errors in vertex reallocation be fixed. If this is caused by the inability to decimate 2 triangles with 1 edge contraction, a penalty must be applied to edge contractions which do not decimate 2 triangles.

In order to be able to use this quadric error metrics for the application that I have suggested additional rules must contribute to the error quadric in the algorithm. A constraint to eliminate singular triangles must be implemented, this can be done by forcing every vertex to have a minimum of 3 faces. Sometimes a triangle overlap was observed in Qslim, i.e. 2 triangles are created using the same 3 vertices. This can be avoided by checking if an edge contraction creates 2 triangles with the same vertices and then putting a high weight on its contraction error. To preserve the shapes of sharp corners, a higher error must be associated with contractions that reduce the dihedral angle between 2 planes below a certain threshold. This would force retention of key features in a model.

In addition stopping criterion based on genus information of the object, or based on maximum obtainable dihedral angle can be implemented.

Once a successful mesh simplification is performed, an optimization scheme to overcome Qslim reduction of object size and re-obtaining the original mesh can be implemented.

Using Algorithm

Subdivision algorithm runs using a function called subdivide. The function takes in a triangle list and a vertex list and ouputs the new subdivided list as well as a plot of the subdivided model. Go to matlab command line

[Tri, V]= subdivide(T_new,Vtemp,subtimes)

T_new → original triangle list array,
Vtemp → original vertex list array
Subtimes → number of subdivisions
Tri→ Output triangle list
V→ Output vertex list

The mesh simplification algorithm runs using a function called meshSimp_contra1. The function takes in the name of a .mat file, which contains a triangle array and a vertex array name tri and V respectively.

[Vtemp, T_new]=meshSimp_contra1(name,contraction,zeroIdx);

name → .mat file name
contraction→ number of contractions wanted
zeroIdx→ is a boolean to inform the code if the triangle input is zero indexed
T_new→ Output triangle list
Vtemp→ Output vertex list

A function name MeshOptPretty can be used to use both files. It performs a specified number of contrations and subdivides the simplified mesh.

MeshOptPretty(name, contr, subtimes)
name → .mat file name
contr→ number of contractions wanted
subtimes→ Number of subdivisions

In addition:
Smf2mat converts a file in .smf format with vertex list v < > and face list f < > to matlab format
Smf2mat(name)
name→ smf file name with extension by default .mat file is stored as matname

Example files and sample.mat files are explained in README

References:

[1] Hoppe, H., Progressive meshes. In *ACM Computer Graphics Proc.*. Annual Conference Series (Siggraph '96), 1996, pp. 99–108.

[2] Garland, M. and Heckbert, P. S., Surface simplification using quadric error metrics. In *Comp. Graph. Proc.*. Annual Conf. Series (Siggraph '97). ACM Press (to appear), 1997.

[3] *Quadric-Based Polygonal Surface Simplification*
Michael Garland, Ph.D. Thesis, Tech. Rept. CMU-CS-99-105.

[4] Schroeder, W. J., Zarge, J. A. and Lorensen, W. E., Decimation of triangle meshes. In *ACM Computer Graphics (SIGGRAPH'92 Proceedings)*, 26, ed. E. E. Catmull, July, 1992, pp. 65–70.

[5] Interpolating subdivision for meshes with arbitrary topology – Zorin, Schröder, et al. – 1996

[6] Smooth subdivision surfaces based on triangles – Loop