Simplification of Polygonal Models for Virtual Reality Applications

Marcio Luis Teixeira
Senior Design Project for EE402
Spring 2000

Department of Electrical Engineering
Colorado State University
Fort Collins, Colorado 80523

Report Approved _____
Project Advisor

Abstract

In computer graphics systems, the need frequently arises to render three dimensional polygonal models with limitations on the amount of available computational power. This problem is particularly significant in virtual reality systems, where the requirement of having an acceptable frame rate places an upper limit on the amount of time that can be dedicated to the processing and rendering of each frame.

This paper describes an implementation of a polygonal simplification algorithm that could be used to enhance the performance of such systems. The work was motivated primarily by the needs of the SHARC system currently under development at Colorado State University and aims to provide fast, incremental, simplification of polygonal models.

Table of Contents

List of Figures

I. Introduction

The author's own research was motivated by ongoing research at Colorado State University in the area of haptic (tactile) rendering. Chen et. al. have demonstrated the viability and usefulness of a computer system that interacts with the user through a combination of visual and haptic sensory information[5]. The SHARC system they developed combines the technologies for autosteroscopic display, acoustic sensing and rendering, and haptic rendering into a single machine.

Some of the most exciting applications suggested for the SHARC system—such as virtual laboratories, teleoperation and endoscopic surgery[5]—will require that the system provide, at least to some extent, the capabilities for rendering arbitrary polygonal models. It has been noted that haptic rendering and collision detection within arbitrary models would be prohibitively expensive unless the means were found to simplify the complexity of these models.

Previous research into the simplification of polygonal models has been motivated primarily by the need to simplify the large, static, polygonal models that one obtains from medical or industrial scanners. Because of this, many algorithms presently available fail to address the more dynamic needs of a virtual reality system. In this paper, the author demonstrates an implementation of a polygonal simplification algorithm that borrows many ideas from existing techniques and presents an algorithm that is specifically tailored for virtual reality application and can be used to accelerate the haptic rendering of a scene.

In a previous report, the author surveyed the past research in this area and evaluated it in terms of the specific requirements of haptic rendering and real-time collision detection. It was found that an algorithm based on the progressive mesh encoding technique developed by Hoppe et. al.[1] could prove useful in haptic rendering.

In this paper, the author describes the development of one such algorithm. Section II explores previous research in the area and describes the various components that were brought together in the development of the present algorithm. Section III reviews the specific characteristics that the author believes the simplification algorithm should have. Section IV explores the current implementation from a structural viewpoint. Section IV presents an more in depth analysis of the progressive meshes encoding technique developed by Hoppe et. al. as well as describing the aspects of the simplification algorithm presented Schroeder, from which the author's own implementation is derived. Section VI describes what has been accomplished, discusses planned improvements to the algorithm and proposes a direction for further research.

II. Previous Work

A. Polygonal Simplification

In my previous report, I identified several key requirements for a polygonal simplification algorithm and surveyed several different research papers in the area. I had explored in detail the work done by Schroeder, Hoppe and Melax and had suggested that a polygonal simplification algorithm that would meet the requirements of the SHARC virtual reality system would likely be derived from one of these algorithms.

The current implementation of the polygonal simplification algorithm is a subset of the general algorithm presented by Schroeder in [2]. For the sake of performance, the current implementation does not attempt to modify topology and considers only simple, manifold vertices as candidates for removal. As in Schroeder [2], the implementation makes use of the progressive meshes technique that was introduced by Hoppe [1] but does not include his mathematically rigorous similarity metrics. As it is currently implemented, the polygonal simplification algorithm uses Schroeder's distance-to-plane metric as its sole similarity metric.

B. Supporting Class Library

One of the core requirements for a polygonal simplification algorithm is the availability of an underlying library for the storage and manipulation of polygonal models. This library should at minimum consist of data structures for storing a vertex list and a list of triangles that form a model. In addition, it should also provide the capability for reading models files

from disk. Of particular importance to a simplification algorithm is that this library should keep information about local topology, such as triangle adjacency, a feature that is not often present in display-oriented model classes.

The unavailability of such a class was one of the major setbacks for the development of the polygonal simplification algorithm. Some components of this library were available, but only in a very fragmented form. Previous work by Jamie Grier included a set of C++ classes that were designed for the display of Wavefront OBJ files. The GLM library, written by Nate Robbins, was a more complete library for the same purpose, though it had been written in C and could not be directly incorporated into this work.

. The class library as it is presently implemented combines ideas from both Grier and Robbins, though it should be emphasized that it is a entirely new implementation tuned for the particular needs of a polygonal simplification algorithm.

At the time I began my project, Jeremy Slade had proposed the development of a haptics library. It is probable that this library would have contained much of the necessary supporting code, but as this library was in its infancy at the time I had begun my work, it was not considered as a starting point for my own class library. The class library that has been developed for the polygonal simplification algorithm is functionally very similar to the proposed HL library and it is likely that the two libraries overlap in many ways.

C. Other Influences

In previous work, the author has found that basing class library interfaces on established well-known standards increase code readability and is preferable to coming up with an entirely new interfaces for similar functions. Programmers familiar with the Java programming language will recognize that some of the classes developed for this project borrow ideas and interfaces from the Java programming language and the Java class library.

III. Requirements

In my previous report, I had included a list of requirements for the proposed polygonal simplification algorithm. In particular, it was noted that the end product of the algorithm should be the creation of a multiresolution model (MRM) with the following characteristics:

- Allows the extraction of many different LODs, so that there is only minor differences between one LOD and the next.
- Has the ability to smoothly blend one LOD into the next.
- Allows the selective refinement or coarsening of select areas of the model.
- Incurs minimal overhead when retrieving a particular LOD from the MRM. The MRM must contain all information required to extract a LOD.
- Provides a natural and efficient way to store the model in memory or on disk.

It was noted that the progressive meshes algorithm developed by Hoppe would provide all these characteristics and would therefore be suitable for the SHARC system. The preceeding list of functional requirements has remained unchanged since the beginning of the project.

In addition to these functional requirements, there was one important non-functional requirement that was not formally stated in my original report. That requirement was the provision for adequate code documentation. Since much of the work in this project was necessarily derived from previous work by others, it was helpful not only to include code

comments, but also a historical of research and references to the relevant papers from which these ideas were gathered. The general approach which I followed can be summarized into the following non-functional requirements:

- All source code that implements an algorithm or idea that was previously presented or described in a paper or conference proceeding should include a reference to that paper. Author and year are generally sufficient, but may also include section numbers or page numbers where appropriate.

- All conference papers that were used directly or are of particular relevance should be photocopied. These photocopies may be written on, highlighted or annotated. Even in lieu of any other code documentation, papers that have been marked and highlighted will allow future programmers to get a sense of which ideas were likely incorporated into the implementation.

- All source code which was used as reference should also be kept, for the same reasons.

- These materials are to be kept together, in a project folder, along with a source code and documentation related to the project.

IV. Implementation Details

A. Class Structure Overview

The polygonal simplification algorithm as it now exists is a part of a larger class library that was developed for the project. This library consists of the following components:

- Data storage classes: Array, ArrayIteratior, List, ListIterator, and PtrListIterator, PriorityQueue
- Geometric entity classes: Vector, Vertex and Triangle.
- Model class hierarchy: Model, ViewIndependentModel and ViewDependentModel
- Utility classes: Metric, Exception and ObjTokenizer

B. Data Storage Classes

The data storage classes are template based and implement a generic set of classes for ordered arrays and linked lists. Of particular importance to this project is the availability of Java-like iterator classes that have greatly simplified the design of the polygonal simplification algorithm and allows the code to written in a more concise, intuitive manner.

C. Geometric Entity Classes

The geometric entity classes define classes for a mathematical vector and for vertices and triangles in the model. The vector class provides an implementation of a three

component mathematical vector and includes methods to implement all the standard mathematical operators on vectors.

The vertex class is derived from the vector class and provides information that is relevant to a vertex in a model. Of particular importance is that each vertex holds information related to the topology of the model at that vertex, including a categorization of the vertex as simple, boundary or complex. Each vertex also caries an ordered list of triangles that use that vertex. For simple vertices or boundary vertices, this list is kept ordered so that the triangles in that list form a complete or semi-complete cycle of triangles around that vertex. The maintenance of these triangle lists is crucial to the correct operation of the decimation algorithm and a considerable amount of care has been taken to ensure that these lists remain sorted.

The vertex class also provides methods for computing and keeping track of the errors that are introduced by the removal of vertices from the model. These error measures are based on the distance-to-plane metric that was suggested by Schroeder[8] and summarized in section V of this document.

The triangle class provides information about the triangles in the model. This class keeps track of the vertices belonging to a triangle, the triangle normal and other properties such as areas and centers. Several methods are provided to facilitate the manipulation of triangles and the relationships among their vertices.

C. Model Hierarchy Classes

The model hierarchy is the most significant component in the library. In addition to providing the basic functions of file I/O, display and viewpoint orientation, the Model class implements two methods that are at the heart of the polygon simplification algorithm. These methods are “edgeColapse” and “vertexSplit.” These methods implement the edge collapse and vertex split operations that are described by Hoppe [1] and are described more fully in

section V of this document. These functions, in particular, have the responsibility of preserving the integrity of the data structure and of preserving the topology of the model.

The Model class itself does not perform polygonal simplification. Instead it only provides the facilities through which vertices may be added or removed from the model. The actual implementation of the simplification algorithm is relegated to subclasses of Model, which may implement any number of variations on the basic simplification algorithms, tailored to specific uses. Two such subclasses have currently been implemented.

The ViewIndependentModel subclass directly implements the polygonal simplification algorithm described by Schroeder [2]. This subclass keeps a priority queue of all vertices in the model. In order to achieve a particular level of reduction, the ViewIndependentModel subclass removes vertices from the queue in order of increasing error and performs an edge collapse operations to remove that vertex. This process continues until the desired level of simplification is achieved. To make this process reversible, each vertex that is removed from the model the ViewIndependentModel class causes an EdgeColapseRecord to be kept. These records allow the model to be reconstructed via vertex splits. In this manner, ViewIndependentModel is able to accommodate a change in the desired level of detail simply by incrementally deleting or adding vertices to a model. The support for incremental refinement is what makes a polygonal simplification algorithm suitable for real-time applications.

The class ViewDependentModel is a variation of ViewIndependentModel that allows the specification of a viewpoint in addition to a desired level of reduction. When removing vertices from the model, the ViewDependentModel gives precedence to vertices that are furthest away from the selected viewpoint making is so areas that are closest to the viewpoint will have more details than those areas which are further away. Though currently this class does not provide support for incremental changes of the viewpoint, support for this is planned.

V. Algorithm Details

A. Definition of Mesh Simplification

The triangular mesh is the most widely used representation for three dimensional models. The mesh approximates the possibly continuous surface of a model by a discrete set of connected triangular faces. Formally, a mesh M consists of a pair (K,V), where K is a simplical complex specifying the connectivity of the vertices, edges and faces that make up the model[7]. The set V specifies the geometric positions of all the vertices in the model and is formally defined as, $V=\{v_1, v_2, v_3, \ldots, v_{n-1}, v_n\}$ where $v_i \in \mathbf{R^3}$. The simplical complex K is the set of vertices indexes{1,...,m} together with a set of non-empty subsets of these vertices. These subsets are called the simplices of K. The single-element simplices {i}∈K correspond to the vertices in the model, the double-element simplices {i,j}∈K correspond to edges in the model and the triple-element simplices {i,j,k}∈K correspond to triangular faces in the model[7].

Polygonal simplification is the process whereby a model M is simplified to create a simpler model M_o having fewer vertices, edges, and faces than the original model. At the same time, the algorithm tries to maintain a visual similarity between the simplified and the original model. The various simplification algorithms are differentiated by the different methods they use for judging the similarity between models M and M_o, by the types of operations they perform the model, and by the heuristics they use to decide which operations to apply to the model in order to meet the simplification goal.

B. Progressive Meshes Encoding

In general, a polygonal mesh simplification algorithm works by interactively making local modifications to the polygonal mesh. Each of these modification affect only a vertex and its adjoining triangles. Many early algorithms operated by removing a vertex from the mesh and retriangulating the resulting hole, as demonstrated in figure 1:

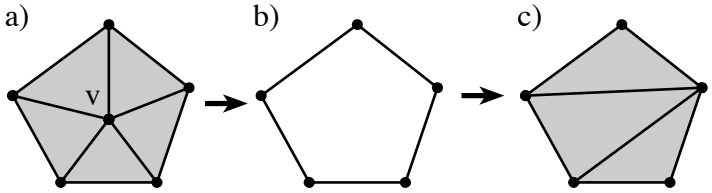

Figure 1: Retriangulating a hole after a vertex *v* has been removed.

Once the algorithm determines that vertex *v* is a candidate for removal (a), the algorithm removes that vertex and the adjoining triangles (b). Then, it fills the hole by retriangulating the hole (c). As the figure demonstrates, each vertex removal eliminates two triangles from the mesh. Many algorithms employing this and other local transformations were developed.

A breakthrough in polygonal mesh simplification occurred when Hoppe introduced progressive meshes in 1996[1]. Hoppe noticed that the retriangulation depicted in figure 1 is equivalent to an edge collapse. Figure 2 shows an edge collapse from vertex *v* to vertex *u*. The operations in figures 1 and 2 result in the same modification to the polygon mesh.

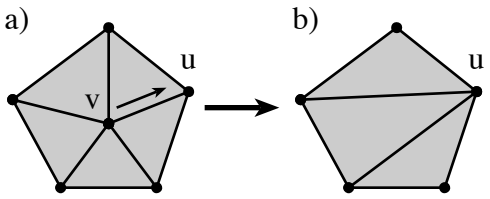

Figure 2: An edge collapse from vertex *v* to *u*

Hoppe determined that edge collapse operations are sufficient to adequately simplify a polygonal model. Hoppe also showed that the edge collapse operations are reversible, so that it is possible to reconstruct the original model from a simplified model simply by applying the edge collapse operations in reverse. To allow this, Hoppe encodes each edge

collapse operation as a record that can be saved to memory or disk. The record is the set of vertex indexes u, v_l and v_r and the coordinates for the removed vertex v:

$$edge\ \ colapse \equiv \left\{u, v_l, v_r, \left(x_v, y_v, z_v\right)\right\}$$

Figure 3a) illustrates the encoding of a edge collapse. Figure 3b) illustrates how this operation may be reversed. Hoppe calls the inverse of an edge collapse a vertex split.

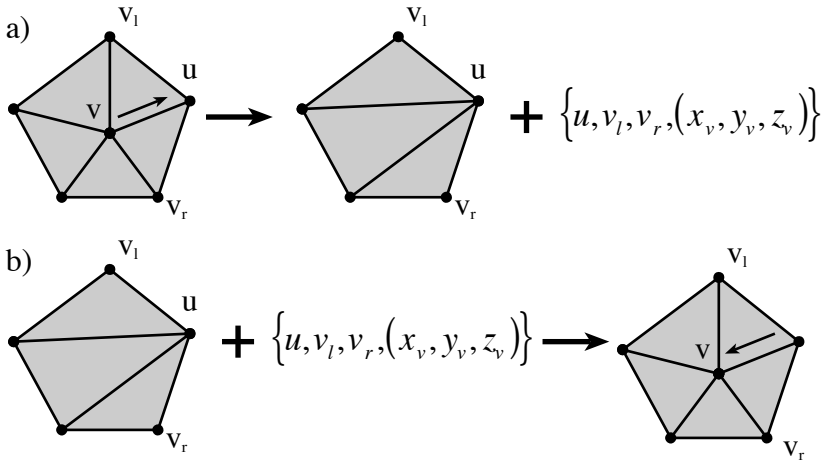

Figure 3: An edge collapse (a) and the associated vertex split (b)

The progressive meshes encoding technique provides a convenient way to create a MRM. Beginning with an unsimplified model M, defined by (K,V), the algorithm uses some criterion to select an appropriate edge to collapse. The algorithm collapses that edge and stores a record corresponding to the edge collapse operation. The procedure is repeated until a model M_o of the desired minimal LOD is obtained. The model M_o is called the base model and is defined by the pair (K_o, V_o) such that $K_o \subset K$ and $V_o \subset V$. That is, the model M_o contains only a subset of the vertices, edges and triangles in M. The MRM generated by the simplification algorithm consists of M_o together with an ordered list $E=\{e_1,e_2,...,e_{n-1},e_n\}$ of edge collapse records corresponding to the edge collapse operations that were performed on M. The list E is reverse-ordered, so that e_1 corresponds to the last edge collapse operation that was performed and e_n corresponds to the first.

By successively reversing the edge collapse operations encoded in E, the decoder may use M_o to create a series of intermediate LODs, each differing from the next by a single

edge collapse. A particular level of detail M_j may be defined as the result of undoing the edge collapse operation e_1 through e_j. When j=n, the resulting model M_n is a perfect reconstruction of the original model M.

C. Edge Selection Criteria

A polygon simplification algorithm is successful only if it is able to reduce the complexity of a model while at the same time ensuring that the resulting model differs as little as possible from the original. For a progressive mesh algorithm, the question becomes one of choosing the ordering of the set of edge collapse operations E, such that for every specific level of detail M_j there exists no other E' such that the corresponding M_j' becomes a better approximation to the original model. An absolute best ordering may not even exist, since the ultimate evaluation of what is "best" is done by humans.

In [2], Schroder proposes an edge selection criterion based on the distance-to-plane and distance-to-line criteria he employed in his earlier polygonal decimation algorithm[8]. Schroder's algorithm operates in three stages. During the first stage, the algorithm classifies each vertex into one of five categories, as shown in figure 4:

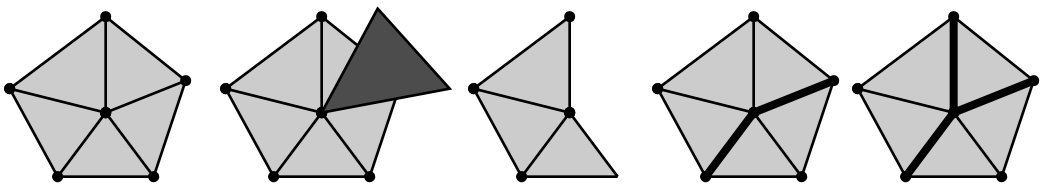

Figure 4: Classifying a vertex in Schroeder's algorithm.

The classification is based on a vertex's topological characteristics and geometric features. Key topological features include whether a vertex is manifold or not, and whether that vertex is a boundary vertex. Non-manifold vertices are classified as *complex*, while manifold vertices are categorized as *boundary*, *simple*, *interior edge*, or *corner vertices*. Interior edge and corner vertices are different from simple vertices in that they posses one or more *feature edges*, that is, they possess one or more edges that separate triangles whose normal vectors differ by some angle greater than a specified threshold. During the second

stage, the algorithm computes for each vertex an error value representing the amount of error that would be introduced if that vertex were removed. Complex and corner vertices are not considered for removal and are assigned infinite errors. The error associated with the removal of a simple vertex is calculated by determining the distance of that vertex to the average plane defined by weighted average of the normal vectors and centers of the triangles adjacent to that vertex. The error associated with boundary and interior-edge vertices is determined by the distance-to-edge metric[8]. The distance-to-plane and distance-to-edge metrics are illustrated in figure 5:

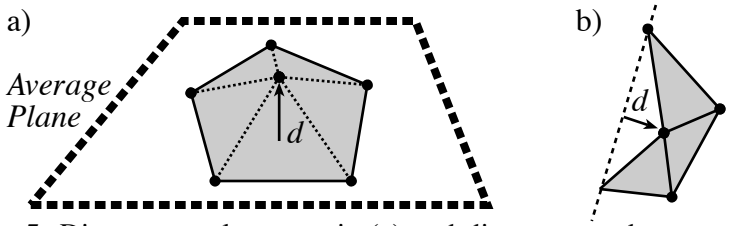

Figure 5: Distance-to-plane metric (a) and distance-to-edge metric (b)

During the last stage, the algorithm puts all the vertices in a priority queue that is sorted by error values. Vertices at the top of the queue incur the smallest error when removed and are removed first. The algorithm collapses the shortest edge connecting that vertex to its neighbors. Every time a vertex is removed the error value associated with that vertex is distributed to the adjacent vertices and the priority queue is updated. The algorithm continues to pick edges to collapse until the desired reduction factor is achieved.

VI. Conclusion & Future Work

The polygonal simplification algorithm, as it has currently been implemented, meets all the requirements that were set forth in section III. The author has demonstrated both a incremental non-viewpoint dependent algorithm, that is suitable for real-time applications, and a non-incremental version of a viewpoint dependent algorithm that is not. For the SHARC system, it is desirable that a incremental, viewpoint dependent algorithm be implemented, as this would allow the simplification of models to be performed real-time.

At the present time, the author sees no impediments to the development of an incremental (and hence real-time) version of the viewpoint dependent simplification algorithm. Such version was under development at the time this paper was written and it will combine the techniques that were used in the two existing versions of the polygonal simplification algorithms. This will involve modifications to the priority queue so that it supports the ability to reprioritize vertices each time the viewpoint is changed. Additional work may also be performed in the ”edgeColapse” and ”vertexSplit” methods in order to improve overall performance. It is in the author's opinion that when these changes have been made, we will have an algorithm that may be directly incorporated into the SHARC system.

Glossary

Boundary vertex - In Schroeder's nomenclature, a vertex surrounded by triangles that form a *manifold* semicircle[2,8]. See figure 4.

Complex vertex - In Schroeder's nomenclature, a vertex which is *non-manifold*[2,8]. See figure 4.

Corner vertex - In Schroeder's nomenclature, a type of simple vertex having two *feature edges*[2,8]. See figure 4.

Distance-to-line metric - In Schroeder's nomenclature, a measurement of the error incurred by removing a boundary vertex from the model[2,8]. See figure 5.

Distance-to-plane metric - In Schroeder's nomenclature, a measurement of the error incurred by removing a simple interior vertex from the model[2,8]. See figure 5.

Edge collapse - Operation defined by Hoppe whereby an edge of a model is collapsed, removing two triangles from the model[1]. See figure 2 and 3.

Feature edge - In Schroeder's nomenclature, an edge that separates two triangles having normal vectors differing by more than a user specified feature angle threshold[2,8].

Geomorphing - An extension, proposed by Hoppe, to the progressive meshes algorithm that allows the generation of smooth interpolations between any two LODs[1].

Haptic rendering - To provide tactile feedback to a user regarding the shape of a three dimensional model. Generally accomplished by means of a mechanical feedback device.

Incremental simplification algorithm - An incremental simplification algorithm is able to accommodate small changes in the simplification parameters by selectively adding or removing vertices from a model. This is a requirement of a real-time simplification algorithm.

Interior edge vertex - In Schroeder's nomenclature, a type of simple vertex having two feature edges[2,8].

Level of Detail (LOD) - One of the many different models, of differing resolution and accuracy, that can be recalled from a multiresolution model[4].

Manifold vertex - In Schroeder's nomenclature, a vertex where each edge using that vertex is shared by no more than two triangles. See figure 4.

Multiresolution Model (MRM) - A data structure that contains multiple different LODs for a particular model and from which a desired LOD may be quickly retrieved[6].

Non-viewpoint dependent simplification - A form of simplification where all parts of the model are simplified uniformly. Contrast this to *viewpoint dependent simplification.*

Polygonal mesh/model - A three-dimensional model composed of polygons. Usually comprised of a vertex list V and a simplical complex K defining the connectivity among those vertices[7].

Polygonal simplification - The process whereby a simplified version of an polygonal mesh may be created.

Progressive meshes - A polygonal simplification algorithm developed by Hoppe whereby a mesh is simplified by successively collapsing edges in the model. The original model may be progressively reconstructed from the simplified model by reversing the edge collapse operations[1].

Retriangulation - After a vertex and its associated triangles are removed, the hole may be filled in by retriangulation. See figure 1.

Selective refinement - The ability to refine or coarsen only a specific portion of a model.

Simple vertex - A vertex whose triangles form a complete cycle around the vertex and whereby each edge connected to the vertex is shared by only two triangles.

Simplical complex - A set of vertices indexes along with a set of non-empty subsets of these vertices, which define the connectivity within a polygonal mesh[1].

Simplification parameter - A parameter that is used to control the manner or extent in which a model is simplified. For a viewpoint independent simplification algorithm, this parameter is typically the desired level of detail. For a viewpoint dependent simplification algorithm, this also includes a vector indicating the viewpoint.

Surface curvature - A measure of the curvature of the surface at a particular vertex. Determined by evaluating the cross products of the normal vectors of the triangles in the vicinity of that vertex [3,9].

Triangle mesh/model - A special type of polygonal mesh containing only triangles.

Vertex split - The inverse of an edge collapse, as defined by Hoppe[1]. See figure 3.

Viewpoint dependent simplification - A form of simplification where a particular viewpoint is used to control the simplification. Typically portions of the model closest to the viewpoint are simplified less than portions that are farthest from the viewpoint.

Acknowledgments

I would like to thank John Cooley, of the Computer Visualization Laboratory, for allowing me to dig into his trove of SIGGRAPH conference proceedings; Jamie Grier, of Colorado State University, for providing me with a fruitful lead in my research; and Stan Melax, of the University of Alberta, for allowing me to read a early copy of his as yet unpublished paper.

References & Bibliography

[1] H. Hoppe: “Progressive Meshes,” Computer Graphics (SIGGRAPH ‘96 Proceedings), 1996, pp. 99-108

[2] W. Schroeder: “A topology modifying progressive decimation algorithm,” In. Proc. of Visualization ‘97, 205-212

[3] S. Melax, B. Watson: A Simple Edge Collapse Cost Measure, Department of Computer Science, University of Alberta

[4] C. Erikson: Polygonal Simplification: An Overview, Department of Computer Science, University of North Carolina at Chapel Hill, 1996

[5] T. Chen, P. Young, D. Anderson, J. Yu, S. Nagata, “Development of a stereoscopic hapatic acoustic real-time computer (sharc),” *Proc. SPIE* **3295**, 1997.

[6] R. Klein, J. Krämer: “Building multiresolution models for fast interactive visualization,” (SCCG, 1997)

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle: “Mesh Optimization,” Computer Graphics (SIGGRAPH ‘93 Proceedings), 1993, 19-26

[8] W. Schroeder, J. Zarge, and W. Lorensen: “Decimation of Triangle Meshes,” Computer Graphics (SIGGRAPH ‘92 Proceedings), Volume 26, Number 2, July 1992, pp. 65-70

[9] S. Melax: Fast, Simple and Effective Polygon Reduction, a demonstration of the authors algorithm: http://www.cs.ualberta.ca/~melax/9/

[10] M. Teixeira: "Simplification of Polygonal Models for Haptic Rendering," Department of Electrical Engineering, Colorado State University.

[11] R. Klein, D. Cohen-Or: "Incremental View-dependent Multiresolution Triangulation of Terrain"

[…] R. Klein, G. Liebich, W. Straßer: “Mesh reduction with error control” In Proc. of Visualization 96, pp. 311-318, October 1996.

[…] R. Klein: "Multiresolution representations for surface meshes", Wilhelm-Schickard-Institut, GRIS, Universität Tübingen, Germany, June 1997

[…] Lorensen, William E. and Harvey E.Cline: “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” Computer Graphics (SIGGRAPH ‘87 Proceedings), July 1987, pp. 163-169.

Appendix A:

Source Code