To appear in IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 2, April–June 1999.

Evaluation of Memoryless Simplification

Peter Lindstrom
lindstro@cc.gatech.edu

Greg Turk
turk@cc.gatech.edu

Graphics, Visualization, and Usability Center
Georgia Institute of Technology

Abstract

This paper investigates the effectiveness of the Memoryless Simplification approach described by Lindstrom and Turk [14]. Like many polygon simplification methods, this approach reduces the number of triangles in a model by performing a sequence of edge collapses. It differs from most recent methods, however, in that it does not retain a history of the geometry of the original model during simplification. We present numerical comparisons showing that the memoryless method results in smaller mean distance measures than many published techniques that retain geometric history. We compare a number of different vertex placement schemes for an edge collapse in order to identify the aspects of the Memoryless Simplification that are responsible for its high level of fidelity. We also evaluate simplification of models with boundaries, and we show how the memoryless method may be tuned to trade between manifold and boundary fidelity. We found that the memoryless approach yields consistently low mean errors when measured by the Metro mesh comparison tool. In addition to using complex models for the evaluations, we also perform comparisons using a sphere and portions of a sphere. These simple surfaces turn out to match the simplification behaviors for the more complex models that we used.

Keywords: Model Simplification, Surface Approximation, Level of Detail, Geometric Error, Optimization

1 INTRODUCTION

Automatic simplification methods are an important technique for accelerating the display of large models. Large polygon models come from a number of sources including computer-aided design, range scanners, terrain mapping and isosurface extraction from volume data. Model sizes continue to grow, thus finding better simplification methods will remain an important problem in computer graphics.

Most simplification methods repeatedly execute simple local changes to geometry that are selected based on some measure of geometric fidelity. Often these measures of fidelity are based on a distance measure from the original geometric model. Usually some form of geometric history is carried along with the partly simplified model to help with distance calculations. In this paper, we evaluate a particular simplification method that keeps no such geometric history and yet that performs as well as many methods that use geometric history [14]. Color Plates 2a through 2d demonstrate the results of using this Memoryless Simplification on a detailed buddha model that is composed of more than one million triangles. There are several reasons for exploring memoryless techniques. First, the memory requirements for such an algorithm are smaller than those that require storing a geometric history. Second, memoryless algorithms are typically faster than those that must query a geometric history in order to calculate a distance measure. Finally, what we learn from the memoryless approach might be used to improve some aspects of those algorithms that retain a geometric history.

As can be done with any other edge collapse approach to simplification, our memoryless technique stores a sequence of edge collapses as a progressive mesh [13]. This means that models that have been simplified using the memoryless approach can benefit from all of the advantages that accrue from using a progressive mesh description, such as progressive transmission and fine-grain selection of polygon count. Our use of the term *memoryless* means that during the off-line simplification process, no geometric comparisons are made between the partly simplified model and the original. It does not mean that the history of the edge collapse operations are forgotten.

The remainder of this paper is organized as follows. In Section 2 we survey some of the simplification methods that have appeared in the literature. Section 3 outlines our overall framework for performing simplification. Section 4 describes the treatment of manifold edges by Memoryless Simplification and compares its results with several other published algorithms. In Section 5 we evaluate several memoryless methods for choosing a new vertex after an edge collapse operation. Section 6 investigates the behavior of several simplification methods on models that have a significant number of boundary edges. We briefly examine the timing results of the various algorithms in Section 7. The final section discusses possible directions for future research.

2 PREVIOUS WORK

In this section we review some of the published work in polygon model simplification. Because the literature on this topic is extensive, we center our attention on those methods that iteratively make local changes to the geometry. Although issues of color and texture are addressed in some of the work that we discuss, we focus our review on geometric history and fidelity. Even with this focus it is not possible to entirely cover this topic, so the reader should not consider this review to be exhaustive. At the end of this section we also describe a geometric measurement tool called Metro for evaluating simplification techniques.

2.1 Simplification Methods

Several methods repeatedly perform vertex removal in order to simplify a model. Such methods must create new polygons in order to fill in the hole that is created by deleting a vertex from a model. An early example of this approach is the work by Schroeder and co-workers, in which they determine a plane that approximates the region around the vertex to be removed and measures the distance from the vertex to this plane [17]. This distance is then used to decide the order in which vertices are removed. Renze and Oliver also make use of the distance measure of [17] in their work, but concentrate their attention on more robust methods of triangulation [15].

More recently, Schroeder has extended his approach in order to perform simplifications that may modify the topology of the model [18]. Topology modification allows greater freedom in simplification, and can yield models with fewer triangles than methods that preserve topology. In addition to allowing topology changes, Schroeder's new method maintains a scalar value at each vertex that gives the maximum error up until that point in the neighborhood of the vertex. This scalar error is an example of retaining some form of geometric history. Another method that uses vertex removal is the work of Hamann [11]. His approach is to estimate the curvature of the surface and remove vertices in locally flat regions. No geometric history is retained in this method.

Several of the more recent vertex removal techniques have made use of geometric history to bound the simplification error. Bajaj and Schikore maintain positive and negative error bounds during repeated removal of vertices [1]. They project into a plane the triangles surrounding the vertex to be removed, and examine intersections between the edges of the original and the proposed new triangles to determine how much error would be incurred by removing the vertex. This error measure is used to prioritize the vertex removals. Ciampalini and co-workers also use vertex removal [2]. For geometric history, they store with each triangle a list of those vertices that have already been removed and that fell within the region now represented by the particular triangle. They use these vertices to make error estimates for each triangle. Cohen et al. use as history an inside and an outside "envelope" (approximations to offset surfaces) in order to constrain the possible choices for vertex removal [4]. In their method, vertices are tested for removal in random order, but only those vertices that would create triangles within the envelopes are actually removed.

A number of researchers use the edge collapse operation as the basis of performing simplification. The edge collapse operator merges two vertices of an edge into a single new vertex, thus removing two triangles from the model. Two decisions must be made: 1) which edge to remove next, and 2) where to place the new vertex. Hoppe uses the edge collapse operator to construct a progressive mesh [13]. He uses a measure of distance from the proposed new triangles to a set of sample points on the original mesh as a quality measure for deciding which edge to collapse. These point samples are a form of geometric history. The distance to the sample points is also used to select a new vertex position that minimizes this measure. This approach is a refinement of earlier work by Hoppe and co-workers in which edge swap and edge split operations are also allowed in order to perform simplification [12].

Cohen and co-workers use edge collapse to help produce a mapping between the original mesh and the simplified model [5, 6]. They use an error box for each triangle to keep a history of the greatest measured error between the meshes, and this error guides the selection of edges to collapse. The new vertex is selected along a line passing through the kernel of a planar projection of the polygons surrounding the edge. The position along this line is chosen in order to minimize the distance to the original mesh. Guéziec also uses edge collapse for simplification, and associates a sphere with each triangle of the mesh as a history of the distance to the original model [10]. He selects edges based on shortest edge length, and then chooses a new vertex position that maintains the enclosed volume of the surface.

Ronfard and Rossignac also use edge collapse as the simplification operator, and they associate a set of planes with each vertex for geometric history [16]. Each plane contains a polygon from the original mesh, and the new vertex from an edge collapse inherits the list of planes from both vertices of the edge. New vertices are placed such that they are at a minimum distance from the planes from both vertex lists. The distance between a proposed new vertex and its list of planes is used to select the next edge to collapse. Garland and Heckbert used this work as a starting point for their own simplification approach [7]. Instead of retaining a list of planes, they store a measure of the squared distance to a collection of planes. This measure is stored as a symmetric 4×4 matrix, one matrix per vertex. This distance measure is used both to place the new vertex and to order the list of edge collapses. More recently they have generalized this method to accurately maintain color and texture values [8].

It should be noted that different users of simplification have different goals in mind. Some applications may require a strict polygon budget, but may not require an exact bound on error. In other situations it may be vital to have an exact bound on the maximum deviation from the original model. Most of the above methods place a greater emphasis on one or the other of these goals (polygon budget or global error).

2.2 Metro

To analyze the effectiveness of various simplification algorithms, we use a program called Metro that compares the distance between two models [3].[1] The fundamental operation of Metro is to find the closest position on the surface of one model to a given position on a second model. This search is accelerated using uniform spatial subdivision. The distance measure is calculated for a large number of point samples on a model. These points may be placed on the surface by scan conversion of the polygons (the method that we used) or they may be distributed randomly. Metro returns both the mean and maximum distances between the two models. Metro optionally computes symmetric surface errors, i.e. by performing two scans, switching the roles of the original and simplified surfaces as the mesh to which the deviation is measured. We use such symmetric errors in our evaluation.

There are several reasons why we chose to use Metro for our numerical comparisons. First, we did not author this tool, and it is our hope that this eliminates one potential source of bias on our part. Second, it is publically available so that others may perform evaluations that can be compared to those presented here. Finally, we have some degree of confidence in Metro because the values that it returns for various models are a good match to our own visual impressions of the qualities of different models.

3 OVERVIEW OF ALGORITHM

3.1 Notation

Before describing our simplification method, we will briefly introduce some terminology and notation. The topological entity called a 0-simplex, or a *vertex*, is denoted by v, with its geometric counterpart written as the 3-vector $\mathbf{x}$. A non-oriented edge $\bar{e}$ is a set $\{v_0^e, v_1^e\} = \{\lfloor e \rfloor_0, \lfloor e \rfloor_1\}$, where $\lfloor s \rfloor$ denotes the $n-1$-*faces* of an n-simplex s; in this case the vertices of the 1-simplex e. Oriented edges are written as ordered pairs $\vec{e} = (\lfloor e \rfloor_0, \lfloor e \rfloor_1)$. All higher order simplices are assumed to be oriented unless otherwise specified, and we use the distinction $\bar{s}$ and $\vec{s}$ only to resolve ambiguities. A *triangle*, or 2-simplex, is a set of oriented edges, e.g. $t = \{\vec{e}_0^t, \vec{e}_1^t, \vec{e}_2^t\} = \{(v_0^t, v_1^t), (v_1^t, v_2^t), (v_2^t, v_0^t)\}$. For convenience, we sometimes write $t = (v_0^t, v_1^t, v_2^t)$ to mean $\{(v_0^t, v_1^t), (v_1^t, v_2^t), (v_2^t, v_0^t)\}$.

The operator $\lceil s \rceil$ gives the $n+1$-simplices of which an n-simplex s is a subset of, e.g. $\lceil v \rceil$ denotes the edges that are incident upon the vertex v. This notation trivially extends to sets, for example $\lfloor S \rfloor = \bigcup_{s \in S} \lfloor s \rfloor$. Thus, the operator $\lfloor S \rfloor$ reduces the dimension of S by one, while $\lceil S \rceil$ adds a dimension. We write the boundary of a set S as the set of oriented edges $\partial S = \{\vec{e} \in S : |\lceil e \rceil| = 1\}$. Fig. 1 illustrates the simplex operators.

[1] We used Metro v2.5 with the options `-s -t` for symmetric errors and textual output, respectively.

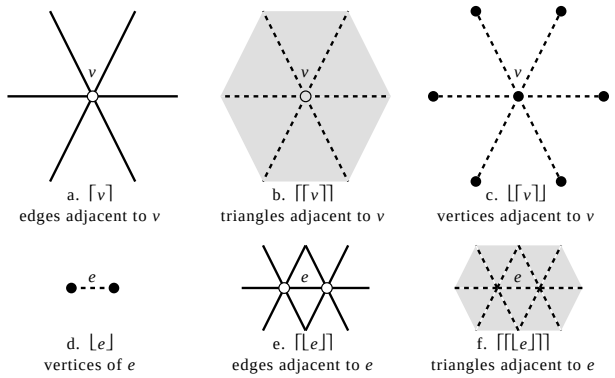


Fig. 1: The simplex operators $\lfloor s \rfloor$ and $\lceil s \rceil$.

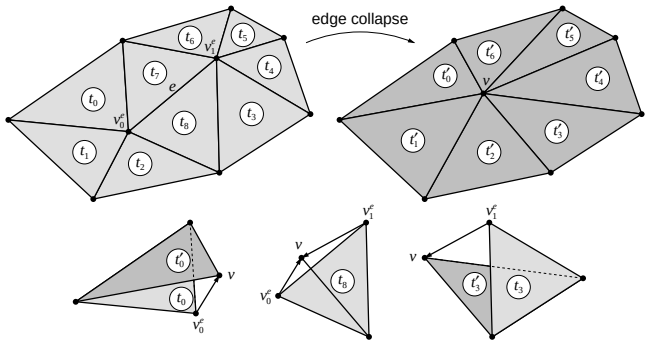


Fig. 2: The edge collapse operation. The manifold edge e is collapsed and replaced with a vertex v. Triangles t_7 and t_8 are removed in the process. Example tetrahedral volumes associated with triangles t_0, t_3, and t_8 are shown.

We use L, A, and V to denote length, area, and signed volume, respectively. All vectors are assumed to be column vectors, and are written as small, bold-face letters; matrices are written as capital letters, e.g. $\mathbf{I}$ is the identity matrix. We make no distinction between points and vectors; we assume that the geometric description of a vertex is a 3-vector relative to some fixed origin. Transposition is used to denote row vectors, e.g. $\mathbf{x}^{\mathrm{T}}$. We will make frequent use of homogeneous coordinates; given a 3-vector $\mathbf{x}^{\mathrm{T}}$, $\overline{\mathbf{x}}^{\mathrm{T}} = \begin{pmatrix}\mathbf{x}^{\mathrm{T}} & 1\end{pmatrix}$ are its corresponding homogeneous coordinates. As is common in computer graphics, we will use homogeneous coordinates as a compact notation for writing affine transformations. In conjunction with block matrix notation, we can express many vector operators, such as addition, inner product, and cross product, in matrix form in a unified manner. As an example,

$$\begin{pmatrix}\mathbf{A} & \mathbf{b}\end{pmatrix}\overline{\mathbf{x}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

where we have used block matrix notation to construct a 3×4 matrix by concatenating a 3×3 matrix $\mathbf{A}$ with a 3-vector $\mathbf{b}$. When applied to matrices, the bar notation is simply used to indicate that the matrix has four columns, e.g. $\mathbf{A} = \begin{pmatrix}\mathbf{A} & \mathbf{b}\end{pmatrix}$. The expression $\mathbf{a} \times \mathbf{b}$ denotes the cross product of two 3-vectors. Since the cross product is a linear operator, it can also be written in matrix form. For a vector $\mathbf{x} = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^{\mathrm{T}}$, $[\mathbf{x}\times]$ denotes the matrix

$$\begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}$$

and we have $\mathbf{a} \times \mathbf{b} = [\mathbf{a}\times]\mathbf{b}$. The scalar triple product is written as $[\mathbf{a}, \mathbf{b}, \mathbf{c}] = \det\begin{pmatrix}\mathbf{a} & \mathbf{b} & \mathbf{c}\end{pmatrix}$.

3.2 Iterative Edge Collapse

Our simplification method consists of repeatedly selecting the edge with a minimum cost, collapsing this edge, and then re-evaluating the cost of edges affected by this edge collapse. Specifically, an edge collapse operation takes an edge $e = \{v_0, v_1\}$ and substitutes its two vertices with a new vertex v. In this process, the triangles $\lceil e \rceil$ are collapsed to edges, and are discarded. The remaining edges and triangles incident upon v_0 and v_1, i.e. $\lceil \lfloor e \rfloor \rceil - \{e\}$ and $\lceil \lceil \lfloor e \rfloor \rceil \rceil - \lceil e \rceil$, respectively, are modified such that all occurrences of v_0 and v_1 are substituted with v. Fig. 2 illustrates the edge collapse operation.

The first step in the simplification process is to assign costs to all edges in the mesh, which are maintained in a priority queue. For each iteration, the edge with the lowest cost is selected and tested for candidacy. Most edge collapse algorithms avoid collapses that result in badly shaped meshes, such as degenerate or non-manifold topology, or geometric artifacts such as folds in the mesh. We have chosen to use the topological constraints described by Hoppe et al. to preserve the genus and to avoid introducing non-manifold simplices [12]. If the edge is not a valid candidate, it is removed from the queue. Given a valid edge, the edge collapse is performed, followed by a re-evaluation of edge costs for all nearby edges affected by the collapse. If any of these edges were previously invalid candidates, they are re-inserted into the queue. As will be described below, our edge cost depends on the triangles $\lceil \lceil \lfloor e \rfloor \rceil \rceil$ and their lower order simplices. Since all these triangles are potentially modified in an edge collapse, all other edges $\{e_j\}$ for which $\lceil \lceil \lfloor e_j \rfloor \rceil \rceil \cap \lceil \lceil \lfloor e \rfloor \rceil \rceil \neq \varnothing$ must be updated, i.e. $\{e_j\} = \lceil \lfloor \lceil v \rceil \rfloor \rceil$, with $|\{e_j\}| = 38$ assuming an average vertex valence of 6. Once the costs for $\{e_j\}$ have been updated, the next iteration begins, and the process is repeated until a desired number of simplices remain.

The general edge collapse method involves two major steps: choosing a measure that specifies the cost of collapsing an edge e, and choosing the position $\mathbf{x}$ of the vertex v that replaces the edge. Many approaches to vertex placement have been proposed, such as picking one of the vertices of e, using the midpoint of e, or choosing a position that minimizes the distance between the mesh before and after the edge collapse. This problem can be viewed as an optimization problem, that is, given an objective function $f_C(\mathbf{x})$ that specifies the cost of replacing e with v, $\mathbf{x}$ is chosen such that f_C is minimized. We have chosen a cost function f_C that encapsulates volume and area information about a model. The following sections describe these geometric issues in greater detail.

3.3 Merging Constraints for Vertex Placement

In choosing the vertex position $\mathbf{x}$ from an edge collapse, we attempt to minimize the change of several geometric properties such as volume and area. In [14], we derived the equations that mathematically describe these properties and provided arguments for why they may be useful in surface simplification. In this paper, we will not provide such an in-depth discussion. Rather, we present only the mathematics required to implement the algorithm.

Our basic approach to finding $\mathbf{x}$ is to combine three linear equality constraints $\hat{\mathbf{a}}_i^{\mathrm{T}}\mathbf{x} = \hat{b}_i$ for $i = 1, 2, 3$, i.e. $\mathbf{x}$ is the intersection of three non-parallel planes in $\mathbb{R}^3$. We have decided to incorporate more than three such constraints in the event that two or more of them are linearly dependent, and we compute and add these to the list of constraints in a pre-determined order of importance. When

three independent constraints

$$\begin{pmatrix}\hat{\mathbf{a}}_1 & \hat{\mathbf{a}}_2 & \hat{\mathbf{a}}_3\end{pmatrix}^{\mathrm{T}}\mathbf{x} = \hat{\mathbf{A}}\mathbf{x} = \hat{\mathbf{b}} = \begin{pmatrix}\hat{b}_1 & \hat{b}_2 & \hat{b}_3\end{pmatrix}^{\mathrm{T}} \tag{1}$$

have been found, the new vertex position $\mathbf{x}$ is computed as

$$\mathbf{x} = \hat{\mathbf{A}}^{-1}\hat{\mathbf{b}} \tag{2}$$

To avoid the case where $\hat{\mathbf{A}}$ is singular or otherwise ill-conditioned, we impose certain rules for adding a new constraint that ensures that $\hat{\mathbf{A}}$ is sufficiently well-conditioned. Given n previous constraints ($n < 3$), we accept $(\hat{\mathbf{a}}_{n+1}, \hat{b}_{n+1})$ according to the following rules:

$$\begin{aligned}
&n = 0: \ \hat{\mathbf{a}}_1 \neq \mathbf{0} \\
&n = 1: \ (\hat{\mathbf{a}}_1^{\mathrm{T}}\hat{\mathbf{a}}_2)^2 < \hat{\mathbf{a}}_1^{\mathrm{T}}\hat{\mathbf{a}}_1\hat{\mathbf{a}}_2^{\mathrm{T}}\hat{\mathbf{a}}_2\cos^2(\alpha) \\
&n = 2: \ [\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3]^2 > (\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_2)^{\mathrm{T}}(\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_2)\hat{\mathbf{a}}_3^{\mathrm{T}}\hat{\mathbf{a}}_3\sin^2(\alpha)
\end{aligned}$$

where α is the minimum permissible angle between the constraint planes. If $(\hat{\mathbf{a}}_{n+1}, \hat{b}_{n+1})$ meets these conditions, we say that it is α-compatible with the list of prior constraints. We have found that the choice of α does not greatly influence the model quality, and that any reasonably small positive value can be used. For all results presented in this paper, α has been set to 1°. We further improve the numerical accuracy by using double precision arithmetic throughout our calculations and employ standard numerical techniques for matrix computations to eliminate large errors.

3.4 Quadratic Optimization

Several of the vertex placement constraints that we use are obtained by minimizing some quadratic objective function subject to a set of linear constraints. In fact, we can cast all subproblems related to choosing the new vertex as quadratic optimization problems, and we have chosen to do so to make our presentation more concise. The quadratic objective functions can be written in the following form:

$$\begin{aligned}
f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{x} + \frac{1}{2}c \\
&= \frac{1}{2}\begin{pmatrix}\mathbf{x}^{\mathrm{T}} & 1\end{pmatrix}\begin{pmatrix}\mathbf{A} & -\mathbf{b} \\ -\mathbf{b}^{\mathrm{T}} & c\end{pmatrix}\begin{pmatrix}\mathbf{x} \\ 1\end{pmatrix} \\
&= \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}\bar{\mathbf{x}}
\end{aligned} \tag{3}$$

with $\mathbf{A}$ being the symmetric positive definite Hessian of f, and $\bar{\mathbf{A}}$ a symmetric positive semidefinite 4×4 matrix. We seek to minimize f subject to the set of prior linear constraints $\hat{\mathbf{A}}\mathbf{x} = \hat{\mathbf{b}}$. This is a linear problem that can be solved analytically. Given n constraints $(\hat{\mathbf{a}}_i, \hat{b}_i)$, let $\mathbf{Q}$ be a $3-n$ by 3 matrix with rows orthogonal to each other and to the vectors $\hat{\mathbf{a}}_i$. Then the additional $3-n$ constraints are

$$\mathbf{Q}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{0} \tag{4}$$

where $\mathbf{A}\mathbf{x} - \mathbf{b} = \nabla f$ is the gradient of f. That is, the constrained minimum of f is found where the projection of its gradient onto the space of free search directions spanned by $\mathbf{Q}$ vanishes. The additional $3-n$ linear constraints given by Equation 4 are added provided they satisfy the compatibility rules.

Throughout the remainder of this paper, we will assume that e is the edge to be collapsed, and v is the replacement vertex, positioned at $\mathbf{x}$. $\{t_i\} = \lceil\lceil\lfloor e\rfloor\rceil\rceil$ are the triangles surrounding an edge, $\{\vec{e}_i\} = \partial\lceil\lfloor e\rfloor\rceil$ are the boundary edges of the changing region, and $\{v_i\} = \lfloor\lceil v\rceil\rfloor - \{v\}$ are the vertices adjacent to v.

4 SIMPLIFICATION OF CLOSED SURFACES

The edges of a polygon model may be classified as *boundary*, *manifold*, and *non-manifold*. A boundary edge has exactly one incident triangle, a manifold edge has two, while non-manifold edges have three or more incident triangles. As is true of many simplification techniques, our Memoryless Simplification algorithm treats manifold and boundary edges differently. In this section we begin by describing the memoryless treatment of manifold edges. We will then use two entirely manifold models, one complex and one quite simple, to compare the memoryless method to a number of other published simplification techniques.

4.1 Vertex Placement

The following subsections describe how to compute the position $\mathbf{x}$ of the new vertex v after a manifold edge e is collapsed. In our discussion below, we will need to compute the signed volume V of a tetrahedron formed by the vertex $\mathbf{x}$ and the three vertices of a triangle t_i. We here derive a simple matrix form for this quantity:

$$\begin{aligned}
&V(\mathbf{x}, \mathbf{x}_0^{t_i}, \mathbf{x}_1^{t_i}, \mathbf{x}_2^{t_i}) = \\
&= \frac{1}{6}\det\begin{pmatrix}\bar{\mathbf{x}} & \bar{\mathbf{x}}_0^{t_i} & \bar{\mathbf{x}}_1^{t_i} & \bar{\mathbf{x}}_2^{t_i}\end{pmatrix} \\
&= \frac{1}{6}((\mathbf{x}_0^{t_i} \times \mathbf{x}_1^{t_i} + \mathbf{x}_1^{t_i} \times \mathbf{x}_2^{t_i} + \mathbf{x}_2^{t_i} \times \mathbf{x}_0^{t_i})^{\mathrm{T}}\mathbf{x} - [\mathbf{x}_0^{t_i}, \mathbf{x}_1^{t_i}, \mathbf{x}_2^{t_i}]) \\
&= \frac{1}{6}\begin{pmatrix}(\mathbf{x}_0^{t_i} \times \mathbf{x}_1^{t_i} + \mathbf{x}_1^{t_i} \times \mathbf{x}_2^{t_i} + \mathbf{x}_2^{t_i} \times \mathbf{x}_0^{t_i})^{\mathrm{T}} & -[\mathbf{x}_0^{t_i}, \mathbf{x}_1^{t_i}, \mathbf{x}_2^{t_i}]\end{pmatrix}\bar{\mathbf{x}} \\
&= \frac{1}{6}\bar{\mathbf{G}}_{V_i}\bar{\mathbf{x}}
\end{aligned} \tag{5}$$

where we have used block matrix notation to define the 1 by 4 matrix $\bar{\mathbf{G}}_{V_i}$ associated with the triangle t_i.

4.1.1 Volume Preservation

As demonstrated in [14] and in sections below, the shape of a model can be better retained if the edge collapse scheme is volume-preserving. In non-planar regions of the surface, tetrahedral volumes are swept out by the triangles involved in an edge collapse (Fig. 2). We associate a sign with each volume according to whether the tetrahedron yields an increase or a decrease in volume. Thus, to preserve the volume enclosed by the surface, we choose $\mathbf{x}$ such that the sum of the signed tetrahedral volumes equals zero, which implies

$$\frac{1}{6}\sum_i \bar{\mathbf{G}}_{V_i}\bar{\mathbf{x}} = 0 \tag{6}$$

where the sum is over the triangles $\{t_i\} = \lceil\lceil\lfloor e\rfloor\rceil\rceil$. Clearly, this equation constrains the solution $\mathbf{x}$ to a plane, and we could proceed by adding this as the first linear constraint on $\mathbf{x}$, provided it is non-degenerate (see Section 3.3), and then use additional criteria to fully specify $\mathbf{x}$. We can equivalently write this as an underdetermined quadratic optimization problem, and let the objective function f_{V_p} be the squared distance of $\mathbf{x}$ to the volume-preserving plane, noting that $f_{V_p}(\mathbf{x}) = 0$ is a guaranteed minimum:

$$f_{V_p}(\mathbf{x}) = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}_{V_p}\bar{\mathbf{x}} = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(\frac{1}{18}\sum_i \bar{\mathbf{G}}_{V_i}^{\mathrm{T}}\sum_i \bar{\mathbf{G}}_{V_i}\right)\bar{\mathbf{x}} \tag{7}$$

The quadratic minimization procedure described in Section 3.4 is then used to recover the single linear constraint described by Equation 6. This may seem like a great deal of unnecessary extra work,

but it lets us use a single procedure for producing linear constraints and allows us to express the vertex placement in terms of a series of quadratic minimizations. It also allows extensions to our algorithm where, instead of minimizing several objective functions in a pre-determined order, one may decide to combine a number of them and minimize their weighted sum, which would effectively de-emphasize the importance of exact volume preservation. This is quite common, for example, in least-squares minimization problems for which the system is generally overdetermined, i.e. no single solution exists that minimizes all objective functions. We will see later how such weighting of objective functions is used to express the edge cost f_C.

4.1.2 Volume Optimization

In addition to setting the sum of *signed* tetrahedral volumes associated with an edge collapse to zero, we would like to choose $\mathbf{x}$ such that the sum of *unsigned* tetrahedral volumes is minimized. Collectively, these unsigned volumes measure the deviation between the surface before and after an edge collapse, integrated over the affected region. As is commonly done, we use squared quantities in place of their absolute values, and minimize the sum of squared volumes swept out:

$$f_{V_o}(\mathbf{x}) = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}_{V_o}\bar{\mathbf{x}} = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(\frac{1}{18}\sum_i \bar{\mathbf{G}}_{V_i}^{\mathrm{T}}\bar{\mathbf{G}}_{V_i}\right)\bar{\mathbf{x}} \tag{8}$$

As described in the previous section, the linear constraints induced by minimization of the quadratic form f_{V_o} are found by applying Equation 4.

We make an interesting observation that the objective function f_{V_o} bears a great deal of similarity to the quadratic form described by Garland and Heckbert [7]. Each of our volume squared terms can be decomposed into the squared distance between $\mathbf{x}$ and the plane of the triangle, weighted by the *square* of the triangle area. In [7], triangles are weighted equally or, in an extension of their algorithm, by the *absolute value* of the triangle area. The two quadratic forms additionally differ in that Garland and Heckbert explicitly store the quadratic form at the vertices during simplification, whereas Memoryless Simplification calculates the quadratic form on-the-fly.

4.1.3 Triangle Shape Optimization

Where the surface is locally planar, the objective function f_{V_o} is zero wherever $f_{V_p} = 0$, and $\mathbf{x}$ can be chosen freely in the volume-preserving plane without introducing any geometric error. However, certain choices of $\mathbf{x}$ may be preferred over others. In such planar regions, we have decided to choose $\mathbf{x}$ such that the resulting triangulation is as uniform as possible to prevent triangles that are long and skinny. By minimizing the sum of squared lengths of the edges $\lceil v \rceil$, we maximize the area to perimeter ratio of the resulting triangles $\lceil \lceil v \rceil \rceil$. Again, we will make use of an auxiliary matrix:

$$\mathbf{x} - \mathbf{x}_i = \begin{pmatrix}\mathbf{I} & -\mathbf{x}_i\end{pmatrix}\bar{\mathbf{x}} = \bar{\mathbf{G}}_{S_i}\bar{\mathbf{x}} \tag{9}$$

where $\bar{\mathbf{G}}_{S_i}$ is associated with the edge formed by v and one of its adjacent vertices $\lfloor \lceil v \rceil \rfloor - \{v\} = \{v_i\}$. The objective function for shape optimization is then

$$f_S(\mathbf{x}) = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}_S\bar{\mathbf{x}} = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(2\sum_i \bar{\mathbf{G}}_{S_i}^{\mathrm{T}}\bar{\mathbf{G}}_{S_i}\right)\bar{\mathbf{x}} \tag{10}$$

This results in additional constraints that yield a unique solution for $\mathbf{x}$.

4.2 Edge Priorities

Many of the edge collapse methods discussed in Section 2 order the edge collapses to minimize the deviation between the simplified model and the *original* surface. In our memoryless algorithm, no such distance measure is available. Rather, we attempt to minimize the deviation of the surfaces between two *successive* edge collapse iterations, and always collapse the edge that yields the smallest amount of change. As mentioned previously, the objective function f_{V_o} associated with volume optimization is a measure of the integrated distance between successive meshes, and we use that as our edge cost for closed surfaces. Formally, we write the edge cost function f_C as

$$\begin{aligned} f_C(\mathbf{x}) &= \lambda f_{V_o}(\mathbf{x}) + (1-\lambda)L(e)^2 f_{B_o}(\mathbf{x}) \\ &= \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(\lambda\bar{\mathbf{A}}_{V_o} + (1-\lambda)L(e)^2\bar{\mathbf{A}}_{B_o}\right)\bar{\mathbf{x}} \end{aligned} \tag{11}$$

The objective function f_{B_o} is used to measure changes to the boundaries of a surface. We will describe f_{B_o} and the user selectable parameter λ below. For closed surfaces, f_{B_o} is zero, and the edge cost reduces to $f_C(\mathbf{x}) = f_{V_o}(\mathbf{x})$.

With the edge cost and vertex placement methods of Memoryless Simplification in hand, we now turn to an evaluation of the method.

4.3 Comparison with Other Methods

We compared the results of six published simplification methods using the Metro tool. Only one of the six methods uses no form of geometric history, namely the Memoryless Simplification method. The complete list of methods is:

1. Mesh Optimization [12].
2. Progressive Meshes [13].
3. Simplification Envelopes v1.1 [4].
4. JADE v2.1 [2].
5. QSlim v2.0β [7].
6. Memoryless Simplification [14].

We chose to use public-domain implementations of each algorithm rather than trying to re-implement each method ourselves. We believe that the public-domain code is probably better tuned than if we had attempted to write the code ourselves. Unfortunately there is still a bias in using only public-domain code because there may be excellent methods that are proprietary. It would be impossible to compare all published methods, but we nevertheless feel that some attempt should be made to perform such comparisons.

The numerical comparisons were performed using two different models, one simple and one complex. The simple model is that of a sphere. This model was created starting from an icosahedron and repeatedly subdividing each triangle into four smaller triangles and projecting the new vertices onto the surface of the unit sphere. This "simple" sphere model is composed of 20,480 triangles, shown in Color Plate 3a. The complex model is that of a horse that was created by merging a number of laser range scans. The horse model contains 96,966 triangles (Plate 1a).

Each simplification method was used to produce eight distinct levels-of-detail for the horse model. The measure of model complexity that we use is the number of edges. For models with few holes, the edge count is nearly the same as the sum of the vertices and faces. Plates 1b through 1g show the results for each simplification method for one level-of-detail. Fig. 3 and 5 graph the numerical comparisons between methods. The horizontal axis is the number of edges of the model, with the more drastically simplified versions appearing towards the left. The vertical axes give the mean and maximum errors in Fig. 3 and 5, respectively. Fig. 3 shows very regular behavior for each of the methods with respect to the mean

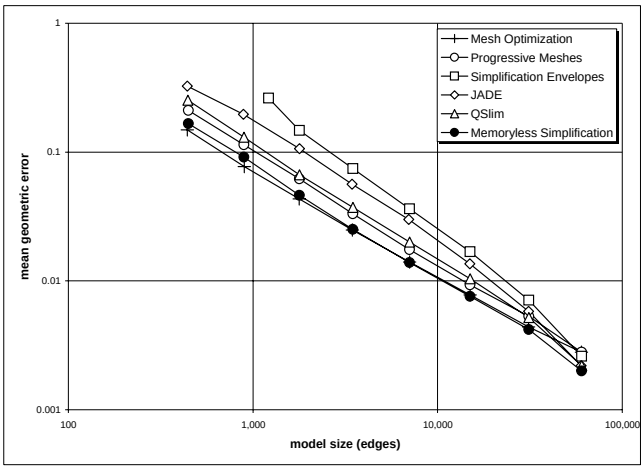


Fig. 3: Mean geometric error for the horse model measured as the percentage of the bounding box diagonal.

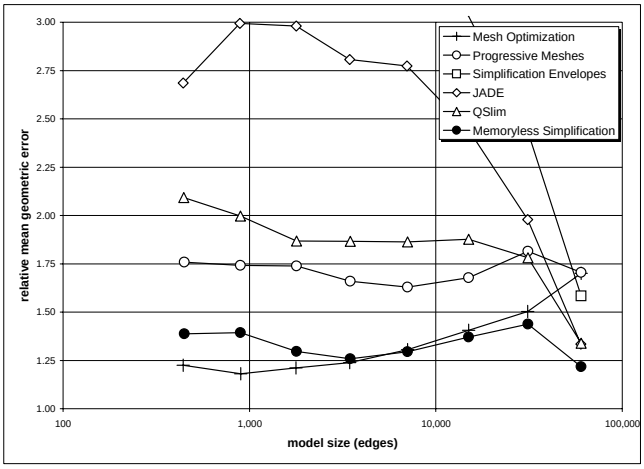


Fig. 4: Normalized mean geometric error for the horse model. The errors have been multiplied by $\frac{1}{25}E^{\frac{7}{8}}$, where E is the number of edges.

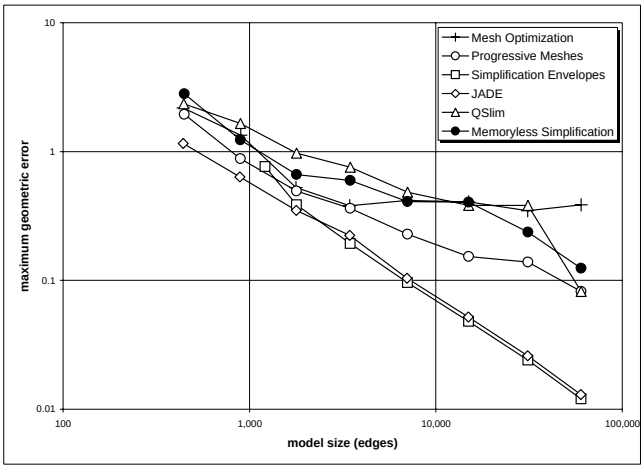


Fig. 5: Maximum geometric error for the horse model.

error. A less cluttered version of this figure can be seen in Fig. 4, in which the data from Fig. 3 has been normalized according to a curve that nearly matches the data. The best mean error is given by Mesh Optimization, followed by Memoryless Simplification.

The maximum errors (Fig. 5) show quite a different assessment of the methods. JADE and Simplification Envelopes out-perform the other methods in terms of maximum error. Note that in [14] the maximum errors seemed to follow no regular pattern. We have used a more recent version of the Metro tool in order to measure mean and maximum error, and we believe that this newer program is more faithful at computing maximum errors.

Unlike complex models such as the horse, we actually know the "best" simplified model of a sphere for certain numbers of faces. In particular, with a budget of 20 faces, the best possible simplification of a sphere is a regular icosahedron. We use this fact to evaluate each of the six simplification methods. Starting with the 20,480 triangle sphere model, we produced just one simplified model from each of the simplification methods, and we forced each method to produce a model with exactly 20 triangles. Fig. 6 contains the numerical analysis of these results, and Plates 3b through 3g show the models graphically. The original sphere is shown transparently in each of these figures, and each model's faces are also translucent so that the back edges can be seen. The "ideal" result, an icosahedron, is shown in Plate 3h. The radius of this regular polyhedron was chosen empirically to be the value for which Metro reported a minimum mean error.

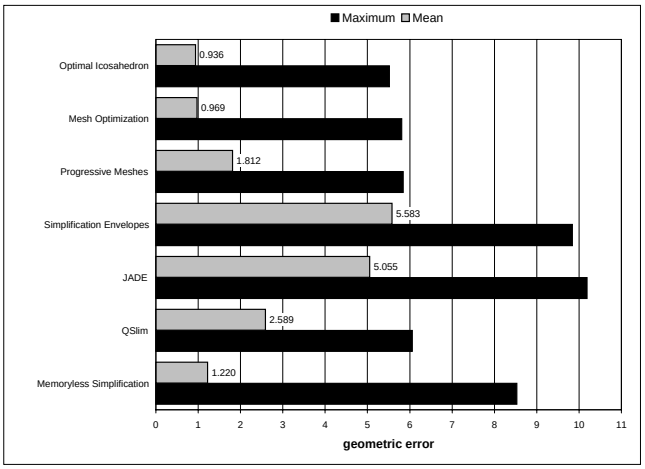


Fig. 6: Mean and maximum geometric errors for the sphere model.

For this sphere model, once again Mesh Optimization outperformed all other methods, followed by Memoryless Simplification.[2] In fact, the relative order of the different methods according to mean error follows exactly the performances for the more complex horse model (Fig. 3) and the Stanford Bunny (numerical results in [14]). This result is tantalizing. Could it be that there are a handful of simple models that are predictors of how a simplification method behaves over a wide class of models? If so, then one might perform initial evaluations of a new simplification algorithm by first testing its performance on a small set of predictor objects. We do not suggest that a sphere alone could serve as such a predictor since we have only compared the rankings it yields to just two models, the horse and the bunny.

Notice that the two vertex removal methods (Simplification Envelopes and JADE) produce simplified spheres that are interior to

[2]Indeed, as evidenced by Fig. 6, Mesh Optimization nearly matched the optimal icosahedron. For such simple models as the sphere, this simplification method is likely to converge to the global optimum.

the original model. This will be the case for any algorithm that is based on vertex removal.

5 VERTEX PLACEMENT

Given the high quality of the results from Memoryless Simplification, it is natural to ask why the method is so effective. Is it the volume preservation, the volume optimization, or the combination of the two? In this section we examine eight different vertex placement schemes in order to better understand the success of Memoryless Simplification. None of these methods rely on any kind of geometric history, but instead they all rely solely on the local geometry of the partially simplified mesh. As noted above, vertex placement is only half of an edge collapse algorithm—we also need an evaluation of the cost of a potential edge collapse in order to prioritize the list of edges. To simplify matters, we have chosen to use the same edge cost with each of the eight vertex placement methods. In particular we use the volume optimization cost f_C, described in Section 4, as the edge cost for all of the methods.

5.1 Various Placement Methods

A number of vertex placement algorithms have been proposed in the simplification literature. Our eight methods attempt to cover a range of possible history-free vertex placement approaches. Selecting one of the original vertices of an edge is one possibility. Which vertex should be selected? We evaluate two possibilities: 1) randomly select one of the vertices, and 2) select the vertex that minimizes the volume optimization edge cost (the "best" vertex). Note that using an original edge vertex forces an edge collapse operation to be a special kind of vertex removal operation.

Another logical position for a new vertex is the midpoint of the edge to be collapsed. This is the third vertex placement method that we examine. For the fourth vertex placement method, we note that the volume optimization criteria for a particular edge actually has a single location that minimizes this cost, and we use this vertex position in the fourth method.

As we described in Section 4, the Memoryless Simplification method uses not just volume optimization, but also exactly preserves the volume of the model. Volume preservation may be used as an additional constraint for any of the four vertex placement methods that are described above. Volume preservation constrains the vertex to lie on a particular plane. We can modify any one of the four methods to select the position on that constraint plane that is closest to the location given by the particular method. This yields four new placement strategies, one of which is the combination of volume optimization and volume preservation that is described in Section 4.1 and that was published in [14]. Note that the two schemes that use volume optimization additionally use triangle shape optimization on occasion to compute the vertex position. However, this optimization is rarely invoked on the models used in this paper. Neither does it have an effect on the geometric error. Here is the complete list of vertex placement methods:

1. Random vertex.
2. Best vertex.
3. Edge midpoint.
4. Volume optimization.
5. Volume preservation, random vertex.
6. Volume preservation, best vertex.
7. Volume preservation, edge midpoint.
8. Volume preservation, volume optimization.

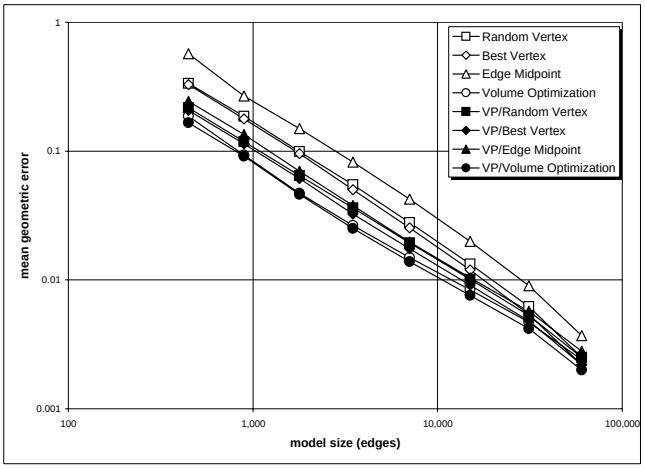


Fig. 7: Mean geometric error for the horse model.

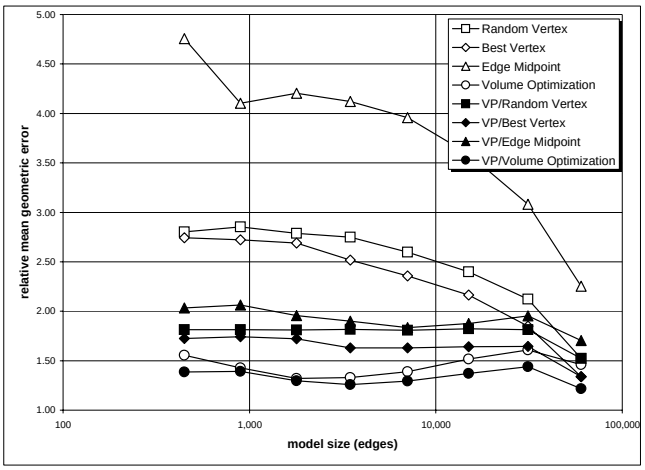


Fig. 8: Normalized mean geometric error for the horse model. See Fig. 4 for details.

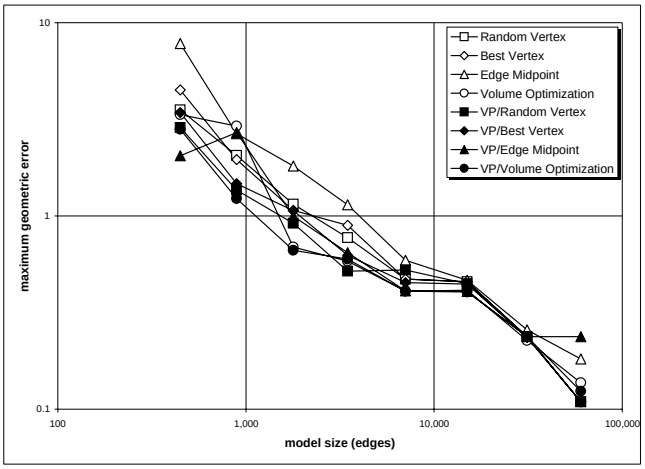


Fig. 9: Maximum geometric error for the horse model.

5.2 Comparison

Fig. 7 and 8 show the mean geometric error of each of these vertex placement methods for the horse model. As before, the horse model has been simplified over a range of levels-of-detail. Several results are worth noting. First, the two edge midpoint selection methods (3 and 7) are out-performed by the methods that use one of the original vertices of the edge. Implementors of simplification methods should not be taken in by the elegant symmetry of using edge midpoints. Second, the best vertex methods (2 and 6) do slightly better than their respective random vertex methods (1 and 5). Third, volume optimization performs better than either using an original vertex or the midpoint. Finally, adding the constraint for volume preservation (5, 6, 7, 8) improves all of the methods (1, 2, 3, 4). Method 8, the vertex placement method described in Section 4.1, yields the smallest mean error over all of the different levels-of-detail. Notice that for the horse model, the mean errors for these last four methods are as low or even lower than the ones produced by most of the simplification methods discussed in Section 4.3, which suggests that volume preservation alone is an important and useful property for model simplification.

Fig. 9 shows the corresponding maximum errors for the same model and set of vertex placement methods. These graphs exhibit less consistency, and we draw no conclusions from the maximum error results other than to point out that the methods that use volume preservation generally produce smaller maximum errors than the ones that don't.

6 BOUNDARY SIMPLIFICATION

It is often desirable to preserve the shape of boundary loops in surfaces that are not closed. In Section 4.1.1, we described a method for preserving the volume bounded by a surface in $\mathbb{R}^3$. This method can easily be reformulated to solve the two-dimensional case; preservation of the area bounded by a curve in $\mathbb{R}^2$. Boundary curves in $\mathbb{R}^3$ are generally not planar, however. In [14], we presented a generalization of the otherwise analogous notion of signed changes in area to non-planar boundaries. This generalization handles the special case of planar boundaries correctly. We here briefly review how to preserve the shape of boundary curves.

6.1 Vertex Placement

When a boundary edge is moved as a result of an edge collapse, it sweeps out a triangular region (Fig. 10). Rather than using a binary sign to encode the change in area enclosed by the boundary, we use the vector $\mathbf{n}$ to signify the change. The direction of $\mathbf{n}$ is perpendicular to the swept out area A and depends on the direction of the boundary edge; the magnitude of $\mathbf{n}$ is equal to A. In Section 4.1, we introduced a convenient matrix notation for computing signed volumes. When dealing with boundary edges, we will use a similar notation for computing the area vectors associated with these edges. We define a 3×4 matrix $\bar{\mathbf{G}}_{B_i}$ used in computing the area vector $\mathbf{n}$ of the triangle given by the vertex $\mathbf{x}$ and a directed boundary edge e_i:

$$\begin{aligned}\mathbf{n}(\mathbf{x},\mathbf{x}_0^{e_i},\mathbf{x}_1^{e_i}) &= \frac{1}{2}(\mathbf{x}\times\mathbf{x}_0^{e_i}+\mathbf{x}_0^{e_i}\times\mathbf{x}_1^{e_i}+\mathbf{x}_1^{e_i}\times\mathbf{x})\\ &= \frac{1}{2}((\mathbf{x}_1^{e_i}-\mathbf{x}_0^{e_i})\times\mathbf{x}-\mathbf{x}_1^{e_i}\times\mathbf{x}_0^{e_i})\\ &= \frac{1}{2}\left([(\mathbf{x}_1^{e_i}-\mathbf{x}_0^{e_i})\times] \quad -\mathbf{x}_1^{e_i}\times\mathbf{x}_0^{e_i}\right)\bar{\mathbf{x}}\\ &= \frac{1}{2}\bar{\mathbf{G}}_{B_i}\bar{\mathbf{x}} \end{aligned} \tag{12}$$

6.1.1 Boundary Preservation

When boundary changes take place in a single plane, only two opposite directions are possible for the area normals, which can then be arbitrarily associated with "positive" and "negative" changes in area. The residual of the sum of these area vectors measures the cumulative change in area due to the edge collapse. In the planar case, we could simply set this residual to zero and solve for $\mathbf{x}$, and the boundary area would be exactly preserved. In the non-planar case, there is generally no $\mathbf{x}$ for which the residual vanishes, so the best we can do is to minimize its magnitude. Again, we can formulate this as a quadratic optimization problem, with the squared magnitude of the residual as the objective function:

$$f_{B_p}(\mathbf{x}) = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}_{B_p}\bar{\mathbf{x}} = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(\frac{1}{2}\sum_i \bar{\mathbf{G}}_{B_i}^{\mathrm{T}}\sum_i \bar{\mathbf{G}}_{B_i}\right)\bar{\mathbf{x}} \tag{13}$$

with $\bar{\mathbf{G}}_{B_i}$ defined in Equation 12. Note that the left submatrix $\sum_i[(\mathbf{x}_1^{e_i}-\mathbf{x}_0^{e_i})\times]$ of $\sum_i \bar{\mathbf{G}}_{B_i}$ is a skew symmetric rank 2 matrix. Consequently, this optimization problem is underdetermined and yields at most two linear constraints. Rather than finding those constraints explicitly, we rely on the α-compatibility tests (Section 3.3) to sort out which constraints are linearly independent.

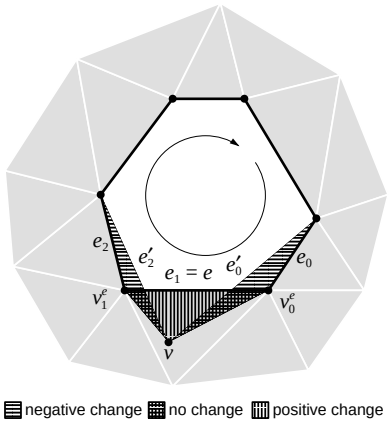


Fig. 10: Collapsing a boundary edge e. The sum of signed areas of the hatched triangles is zero. The circular arc indicates the orientation of the boundary edges $\{\vec{e}_i\}$.

6.1.2 Boundary Optimization

To further improve the shape of simplified boundaries, we minimize the sum of squared changes in area. Here we are only concerned with the magnitude of each change in area (and not its orientation), making this minimization a 2D analogy to the 3D volume optimization. The associated objective function can be written as

$$f_{B_o}(\mathbf{x}) = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\bar{\mathbf{A}}_{B_o}\bar{\mathbf{x}} = \frac{1}{2}\bar{\mathbf{x}}^{\mathrm{T}}\left(\frac{1}{2}\sum_i \bar{\mathbf{G}}_{B_i}^{\mathrm{T}}\bar{\mathbf{G}}_{B_i}\right)\bar{\mathbf{x}} \tag{14}$$

As before, Equation 4 is used to further constrain $\mathbf{x}$.

6.2 Edge Priorities

In order to account for boundary changes in the simplification and to penalize edge collapses that significantly alter the boundary shape, we include the boundary optimization term as part of the edge cost for open surfaces. We have decided to weight the boundary term by the square of the length of the edge e to ensure scale invariance. The resulting edge cost is a convex combination of the

volume and boundary terms, given by Equation 11 in Section 4.2. We will later see how the the parameter λ can be varied to trade off the quality between the interior of the surface and its boundaries. In general, a setting of $\lambda = \frac{1}{2}$ tends to yield good results.

Rather than computing constraints for the volume and boundary optimization independently, we use f_C as the objective function to be minimized. To summarize the vertex placement scheme, we add constraints in the following order: 1) volume and boundary preservation, 2) edge cost minimization, and 3) triangle shape optimization. Since volume and boundary preservation together yield at most three constraints, the order in which they are performed does not matter.

6.3 Comparison with Other Methods

Using the same array of published simplification methods as described in Section 4.3, we simplified two models that contain boundary edges. One of these objects is the Stanford Bunny, which has several holes in the base of the model, and is composed of 69,451 triangles (Color Plate 4a). The other model is a spherical segment that is tessellated with 16,384 triangles (Plate 5a). The captions in the color plates for this section include edge counts for the boundaries of the models, E_∂.

There was considerable variability between algorithms in the qualitative results of simplifying the Stanford Bunny's boundaries. Two of the methods, JADE and QSlim, devoted a large number of polygons towards preserving the boundaries (Plates 4e and 4f). Progressive Meshes and Simplification Envelopes used considerably fewer polygons around the boundaries (Plates 4c and 4d). Mesh Optimization produced the poorest visual results (Plate 4b), but there are a number of parameters in this algorithm and it is likely that its results could be improved with some adjustments to these values.

We believe that different applications require different amounts of fidelity for boundaries, thus it is useful to have explicit control over the boundary detail. The λ value of the Memoryless Simplification edge cost (Equation 11) provides such control. For Plates 4g, 4h and 4i, the λ value for Memoryless Simplification was varied to produce three different models to show this control over boundary fidelity. When $\lambda = \frac{1}{32}$, many polygons are devoted to preserving the boundary (Plate 4g). At the other extreme in Plate 4i, when $\lambda = \frac{31}{32}$, the boundary is formed by many fewer polygons. The model of Plate 4h shows the middle ground between the two extremes, using $\lambda = \frac{1}{2}$.

As with the sphere model used in Section 4.3, we created a geometrically simple model to further exercise simplification of boundaries. This model is a portion of a sphere with two boundaries, as shown in Plate 5a. This 16,384 triangle model was created by connecting vertices that are at 33 latitude and 256 longitude positions. The range of latitudes is from 30 to 60 degrees. For each algorithm, we created a simplified model with 87 or 88 edges. The results shown in Plates 5b through 5i emphasize that boundary preservation is a tug-of-war between the number vertices on the boundary of a model and the number of vertices that are entirely surrounded by triangles (the manifold vertices). There was considerable variation in the number of vertices that each method placed on the boundaries. Mesh Optimization used the fewest boundary vertices, whereas QSlim chose the opposite extreme and placed the most vertices on the boundary. Changing the parameter λ of Memoryless Simplification spans this variation. Plate 5g shows that $\lambda = \frac{1}{32}$ produces a model much like that of QSlim, with few vertices that are not on the boundary. Plate 5i shows that $\lambda = \frac{31}{32}$ results in a model that is much like that of Mesh Optimization, with relatively fewer vertices on the boundary. Using $\lambda = \frac{1}{2}$ produces a model that is between these two extremes (Plate 5h).

Fig. 11 is a numerical comparison of the different simplification methods for the spherical segment. It should be noted that Metro does not give a separate error measure for boundaries, so these results reflect the geometric fidelity across the entire model. Fig. 11 graphs the mean and maximum error for a range of λ values for Memoryless Simplification in order to highlight the tradeoff between manifold and boundary fidelity. The figure shows that Progressive Meshes and Mesh Optimization give the least mean geometric error, and that for large values of λ these results are also matched by Memoryless Simplification. There is a clear relationship between high values of λ (greater emphasis on volume optimization) and smaller mean geometric error. With lower λ values, the edge cost emphasizes the boundary cost f_{B_o} more heavily, resulting in better boundary detail but more error over the surface of the model.

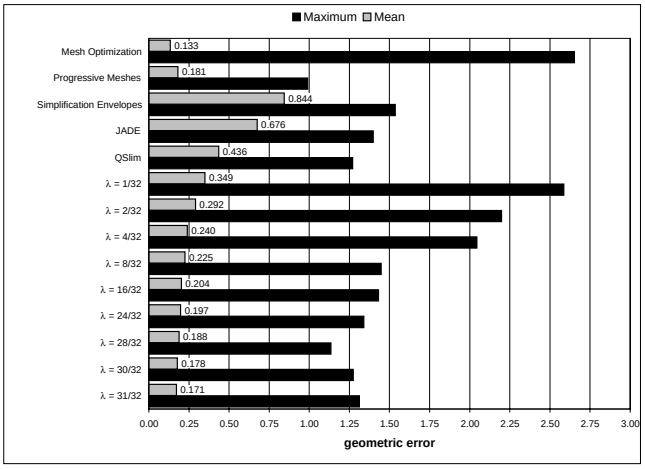


Fig. 11: Mean and maximum geometric errors for the spherical segment model.

7 COMPUTE TIMES

We collected timing statistics for all six algorithms in the process of creating the simplified versions of the horse model. All models were simplified on a four-processor, 195 MHz R10000 Silicon Graphics Onyx^2 machine with 1 GB of main memory. These times are shown in Fig. 12. As can be seen in this figure, QSlim was the fastest of the algorithms that we tried, followed in order by Memoryless Simplification, JADE, Simplification Envelopes and Mesh Optimization. We only have one data point for Progressive Meshes, and this indicates that the speed of Progressive Meshes is close to that of Simplification Envelopes.

One striking feature of Fig. 12 is that Simplification Envelopes and Mesh Optimization have nearly constant running times regardless of the degree of simplification. This suggests that initialization costs dominate the running times for these algorithms. QSlim actually appears to run faster when producing more coarse models. We believe that QSlim is so fast that it becomes I/O bound when writing out the files for the higher levels-of-detail.

We recognize that our comparison of the running times of different methods is highly dependent on the particular *implementations* of these algorithms. It is entirely possible that each of these methods may have faster and more efficient implementations. However, we believe that the results presented in this section will provide both a reasonable estimate of how the algorithms compare amongst each other, as well as what their individual characteristic running times are with respect to model size. Memory usage is another interesting and important aspect of algorithmic efficiency, and is also

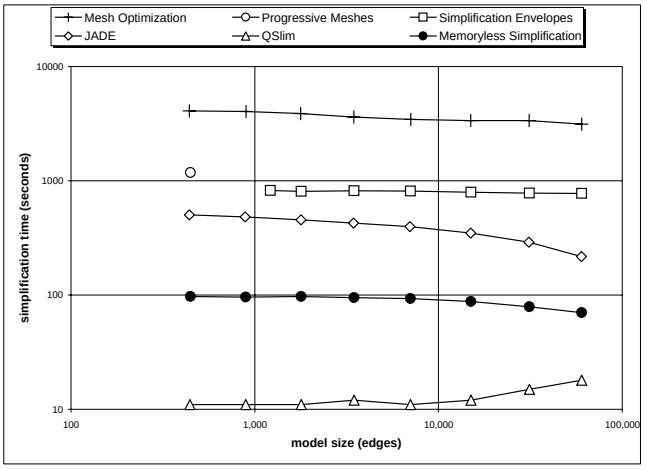


Fig. 12: Simplification time for the horse model.

one of the strengths of our simplification algorithm. We decided against comparing memory efficiency since such results tend to be even more implementation dependent. Many of the public-domain codes support features that are not necessary for such an evaluation, e.g. progressive storage, algorithms for preservation and storage of other surface attributes such as colors and texture coordinates, etc., which may unfairly skew their memory usage. However, based both on the nature of our algorithm and empirical data, we conclude that implementations of our algorithm have the potential to be more memory efficient than previous edge collapse methods.

8 CONCLUSION AND FUTURE WORK

This paper presents an evaluation of the Memoryless Simplification method that was introduced in [14]. Such an analysis is important because it is counter-intuitive that a method that retains no geometric history can be an effective simplification method. New contributions of this paper include:

- Evaluation of eight different vertex placement schemes to better understand what aspects of vertex placement are important.
- Demonstration of the tradeoff between preservation of the boundary and the manifold geometry of a model.
- Finding that simplification times for some methods are largely independent of the final level-of-detail to be created.
- Introduction of two "simple" models that can be used to help evaluate surface and boundary simplification.

There are several potential avenues for future research. One possibility is to use the memoryless edge collapse approach to improve the quality of other simplification methods. For instance, a hybrid of Simplification Envelopes and the memoryless edge collapse may yield an algorithm with a global bound on error that also has better mean distance behavior. Another future direction is to see if indeed there are simple models that can act as reliable predictors of complex models. We might look for objects that can act as predictors for models with sharp corners or negative curvature. Another potential direction is to create tools that can numerically evaluate the simplification fidelity near the boundaries of a model. The edge cost and vertex placement methods used by Memoryless Simplification work well in many cases, but there might be still better placement methods that do not require geometric history. Finally, are there memoryless simplification methods that are effective at preserving surface attributes such as color and texture?

Acknowledgements

This work was funded by an NSF CAREER award (CCR–9703265). We would like to thank Cyberware, Inc., for the horse model, the Stanford Computer Graphics Laboratory for the bunny and buddha models, Hugues Hoppe for providing progressive mesh models and an implementation of "Mesh Optimization", and the people at CMU, CNUCE, and UNC for making their simplification tools available.

References

[1] C. Bajaj and D.R. Schikore, "Error-Bounded Reduction of Triangle Meshes with Multivariate Data," *Proc. Visual Data Exploration and Analysis III*, SPIE, Vol. 2656, Jan. 1996, pp. 34–45.

[2] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno, "Multiresolution Decimation Based on Global Error," *The Visual Computer*, Springer International, Vol. 13, No. 5, 1997, pp. 228–246.

[3] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces," *Computer Graphics Forum*, Vol. 17, No. 2, June 1998, pp. 167–174.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification Envelopes," *Proc. SIGGRAPH 96*, ACM SIGGRAPH, Aug. 1996, pp. 119–128.

[5] J. Cohen, D. Manocha, and M. Olano, "Simplifying Polygonal Models Using Successive Mappings," *Proc. Visualization '97*, IEEE Computer Soc. Press, Oct. 1997, pp. 395–402.

[6] J. Cohen, D. Manocha, and M. Olano, "Appearance-Preserving Simplification," *Proc. SIGGRAPH 98*, ACM SIGGRAPH, July 1998, pp. 115-122.

[7] M. Garland and P.S. Heckbert, "Surface Simplification using Quadric Error Metrics," *Proc. SIGGRAPH 97*, ACM SIGGRAPH, Aug. 1997, pp. 209–216.

[8] M. Garland and P.S. Heckbert, "Simplifying Surfaces with Color and Texture using Quadric Error Metrics," *Proc. Visualization '98*, IEEE Computer Soc. Press, Oct. 1998, pp. 263–269.

[9] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts, "Smooth Hierarchical Surface Triangulations," *Proc. Visualization '97*, IEEE Computer Soc. Press, Oct. 1997, pp. 379–386.

[10] A. Guéziec, "Surface Simplification with Variable Tolerance," *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, Nov. 1995, pp. 132–139.

[11] B. Hamann, "A Data Reduction Scheme for Triangulated Surfaces," *Computer Aided Geometric Design*, Vol. 11, No. 2, Apr. 1994, pp. 197–214.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Proc. SIGGRAPH 93*, ACM SIGGRAPH, Aug. 1993, pp. 19–26.

[13] H. Hoppe, "Progressive Meshes," *Proc. SIGGRAPH 96*, ACM SIGGRAPH, Aug. 1996, pp. 99-108.

[14] P. Lindstrom and G. Turk, "Fast and Memory Efficient Polygonal Simplification," *Proc. Visualization '98*, IEEE Computer Soc. Press, Oct. 1998, pp. 279–286.

[15] K.J. Renze and J.H. Oliver, "Generalized Surface and Volume Decimation for Unstructured Tessellated Domains," *Proc. VRAIS '96*, IEEE Computer Soc. Press, Mar. 1996, pp. 111–121.

[16] R. Ronfard and J. Rossignac, "Full-Range Approximation of Triangulated Polyhedra," *Proc. Eurographics '96*, Computer Graphics Forum, Vol. 15, No. 3, Aug. 1996, pp. 67–76.

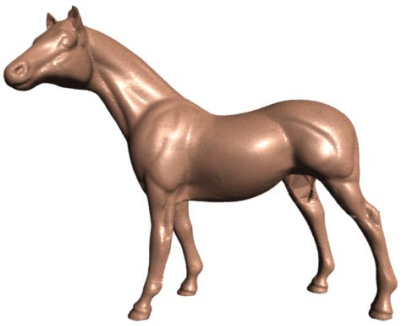
[17] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *Proc. SIGGRAPH 92*, ACM SIGGRAPH, July 1992, pp. 65–70.

[18] W.J. Schroeder, "A Topology Modifying Progressive Decimation Algorithm," *Proc. Visualization '97*, IEEE Computer Soc. Press, Oct. 1997, pp. 205–212.

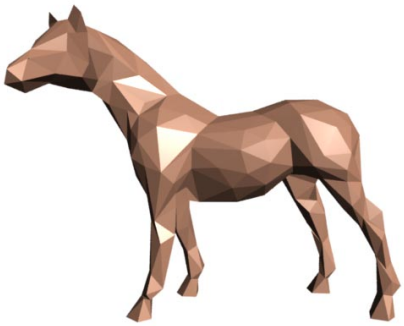
Biographies

Peter Lindstrom is a doctoral student at the Graphics, Visualization, and Usability Center at the Georgia Institute of Technology. He received a BS degree in computer science, mathematics, and physics from Elon College, North Carolina in 1994. His research interests include model simplification and multiresolution methods, real-time rendering, and geometry compression.

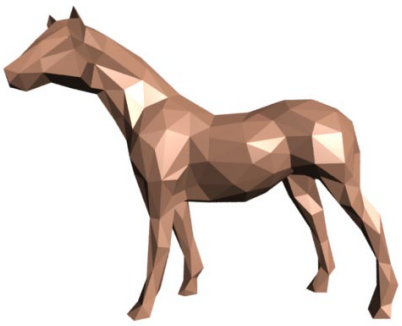
Greg Turk received a PhD in computer science in 1992 from the University of North Carolina at Chapel Hill. He was a postdoctoral researcher at Stanford from 1992 to 1994, and then returned to UNC for two years as a research scientist. He is currently an assistant professor at the Georgia Institute of Technology, where he is in the College of Computing and is also a member of the Graphics, Visualization and Usability Center. His research interests include 3D modeling, rendering, image processing and scientific visualization.



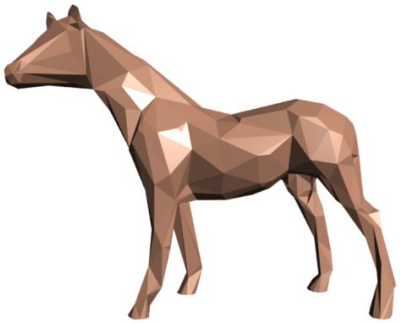
1a.
Original model
T = 96,966



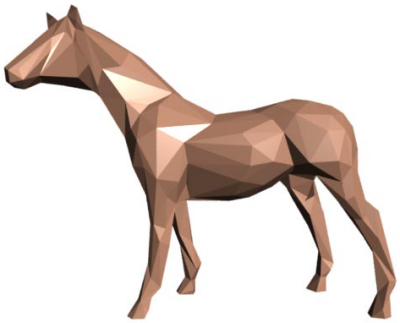
1b.
Mesh Optimization
T = 598
time = 1:07:27



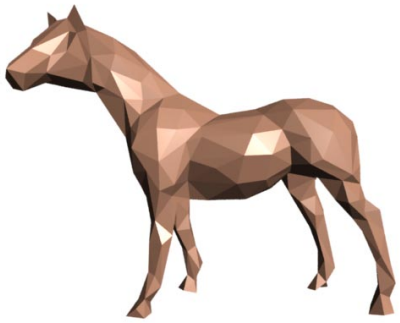
1c.
Progressive Meshes
T = 596
time = 19:41



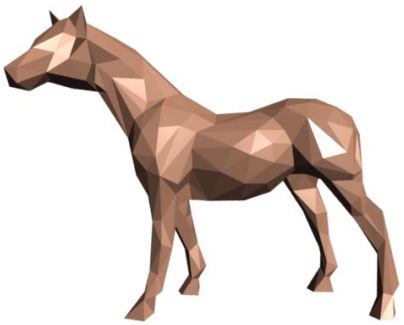
1d.
Simplification Envelopes
T = 810
time = 13:28



1e.
JADE
T = 592
time = 08:04



1f.
QSlim
T = 596
time = 00:11



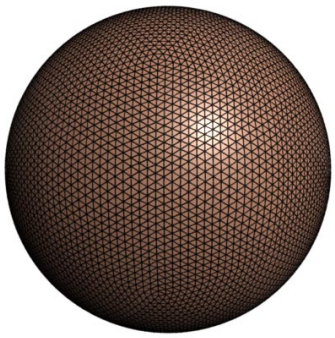
1g.
Memoryless Simplification
T = 596
time = 01:36

2a.
Original model
T = 1,087,716

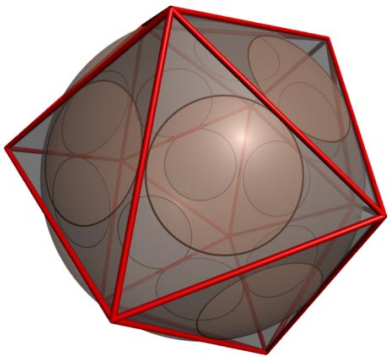
2b.
Memoryless Simplification
T = 19,999
time = 19:32

2c.
Memoryless Simplification
T = 4,999
time = 19:33

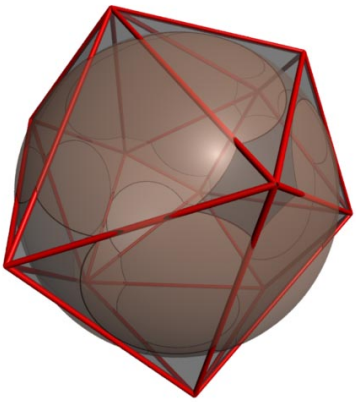
2d.
Memoryless Simplification
T = 1,000
time = 19:09



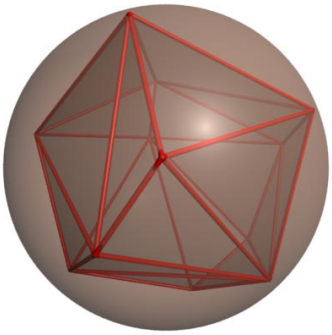
3a.
Original model
T = 20,480



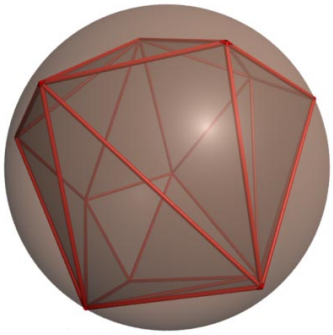
3b.
Mesh Optimization
T = 20
time = 13:43



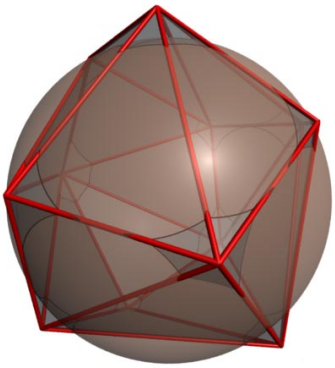
3c.
Progressive Meshes
T = 20
time = 02:50



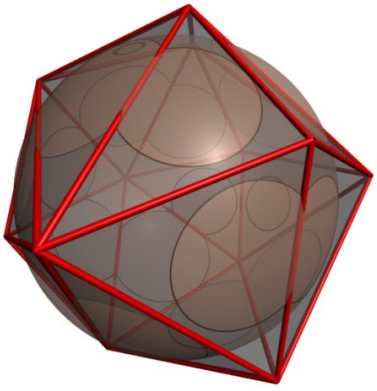
3d.
Simplification Envelopes
T = 20
time = 15:22



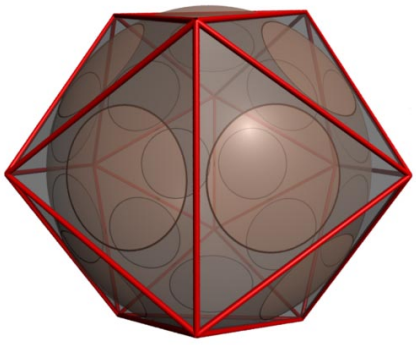
3e.
JADE
T = 20
time = 02:06



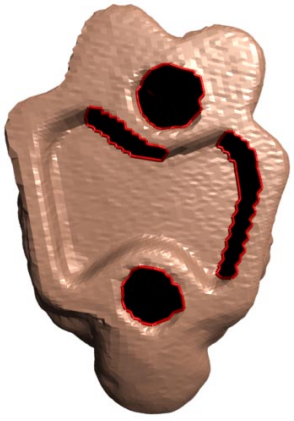
3f.
QSlim
T = 20
time = 00:03



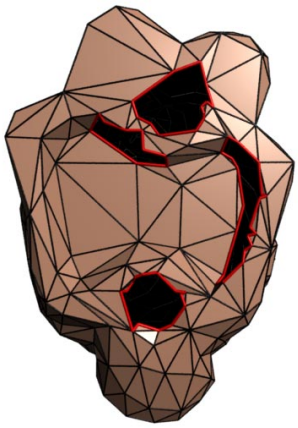
3g.
Memoryless Simplification
T = 20
time = 00:18



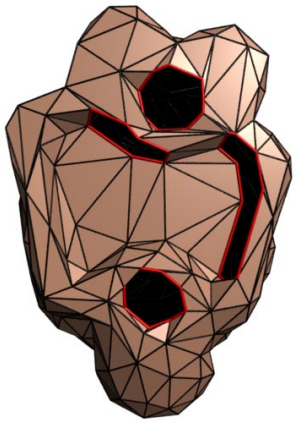
3h.
Optimal Icosahedron
T = 20



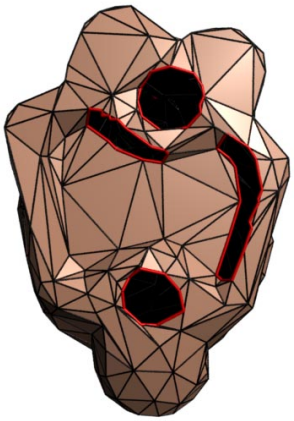
4a.
Original model
V = 34,834; T = 69,451; E = 104,288
E_∂ = 223



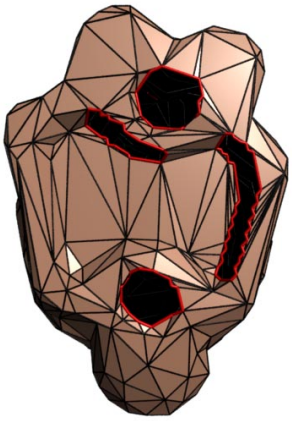
4b.
Mesh Optimization
V = 701; T = 1,342; E = 2,046
E_∂ = 66
time = 42:04



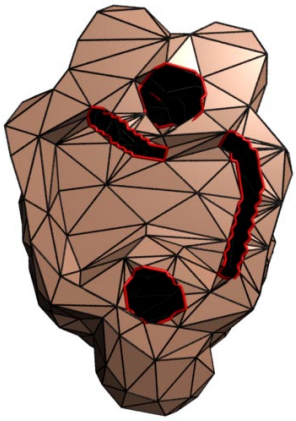
4c.
Progressive Meshes
V = 686; T = 1,338; E = 2,027
E_∂ = 40
time = 08:20



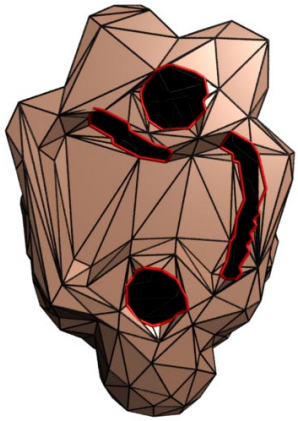
4d.
Simplification Envelopes
V = 686; T = 1,314; E = 2,003
E_∂ = 64
time = 09:33



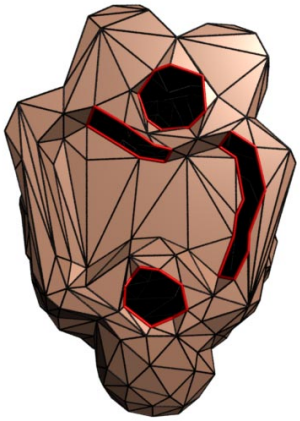
4e.
JADE
V = 691; T = 1,289; E = 1,983
E_∂ = 99
time = 05:25



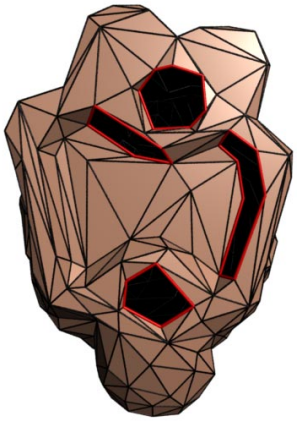
4f.
QSlim
V = 711; T = 1,313; E = 2,027
E_∂ = 123
time = 00:03



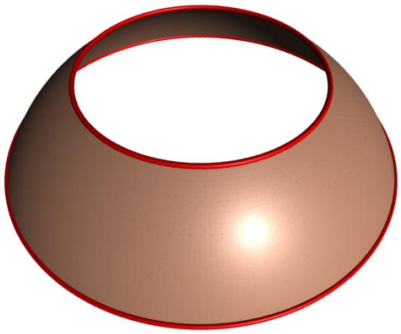
4g.
Memoryless Simplification (λ = 1/32)
V = 698; T = 1,326; E = 2,027
E_∂ = 76
time = 01:34



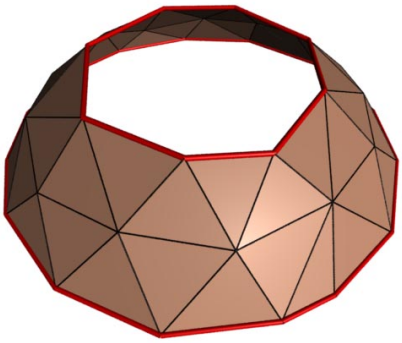
4h.
Memoryless Simplification (λ = 1/2)
V = 686; T = 1,336; E = 2,025
E_∂ = 42
time = 01:20



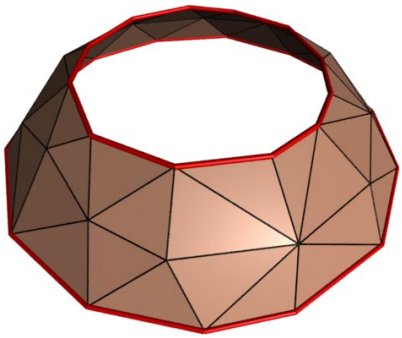
4i.
Memoryless Simplification (λ = 31/32)
V = 682; T = 1,342; E = 2,027
E_∂ = 28
time = 01:21



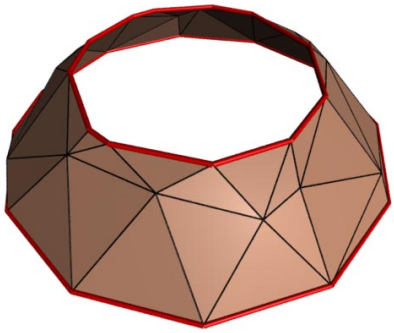
5a.
Original model
V = 8,448; T = 16,384; E = 24,832
E_∂ = 256 + 256 = 512



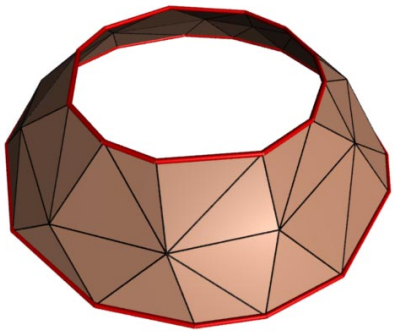
5b.
Mesh Optimization
V = 37; T = 51; E = 88
E_∂ = 15 + 8 = 23
time = 08:47



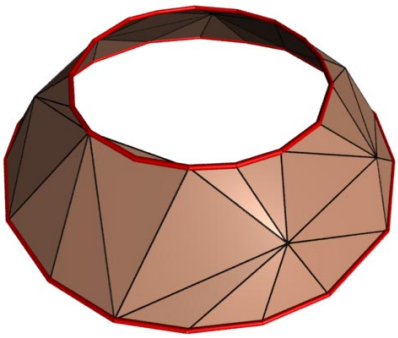
5c.
Progressive Meshes
V = 38; T = 50; E = 88
E_∂ = 14 + 12 = 26
time = 02:39



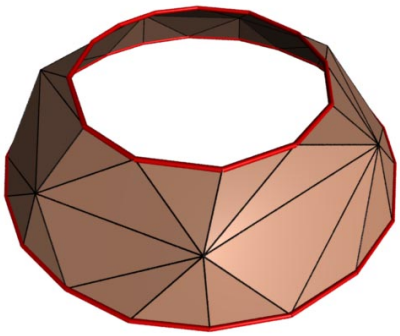
5d.
Simplification Envelopes
V = 37; T = 50; E = 87
E_∂ = 12 + 12 = 24
time = 02:58



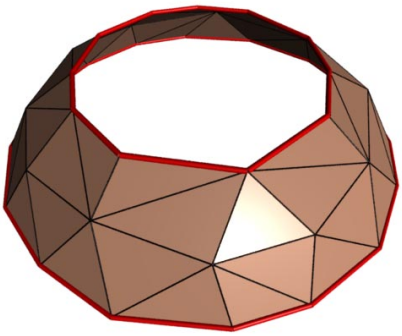
5e.
JADE
V = 38; T = 49; E = 87
E_∂ = 15 + 12 = 27
time = 01:23



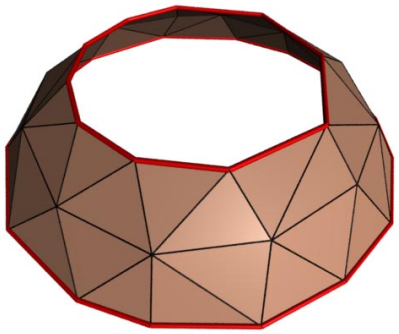
5f.
QSlim
V = 41; T = 47; E = 88
E_∂ = 18 + 17 = 35
time = 00:02



5g.
Memoryless Simplification (λ = 1/32)
V = 40; T = 47; E = 87
E_∂ = 20 + 13 = 33
time = 00:19



5h.
Memoryless Simplification (λ = 1/2)
V = 38; T = 49; E = 87
E_∂ = 16 + 11 = 27
time = 00:17



5i.
Memoryless Simplification (λ = 31/32)
V = 37; T = 49; E = 86
E_∂ = 15 + 10 = 25
time = 00:16