Surface Simplification Using Quadric Error Metrics

Michael Garland* Paul S. Heckbert†

Carnegie Mellon University

Abstract

Many applications in computer graphics require complex, highly detailed models. However, the level of detail actually necessary may vary considerably. To control processing time, it is often desirable to use approximations in place of excessively detailed models.

We have developed a surface simplification algorithm which can rapidly produce high quality approximations of polygonal models. The algorithm uses iterative contractions of vertex pairs to simplify models and maintains surface error approximations using quadric matrices. By contracting arbitrary vertex pairs (not just edges), our algorithm is able to join unconnected regions of models. This can facilitate much better approximations, both visually and with respect to geometric error. In order to allow topological joining, our system also supports non-manifold surface models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surface and object representations

Keywords: surface simplification, multiresolution modeling, pair contraction, level of detail, non-manifold

1 Introduction

Many computer graphics applications require complex, highly detailed models to maintain a convincing level of realism. Consequently, models are often created or acquired at a very high resolution to accommodate this need for detail. However, the full complexity of such models is not always required, and since the computational cost of using a model is directly related to its complexity, it is useful to have simpler versions of complex models. Naturally, we would like to automatically produce these simplified models. Recent work on surface simplification algorithms has focused on this goal.

As with most other work in this area, we will focus on the simplification of polygonal models. We will assume that the model consists of *triangles only*. This implies no loss of generality, since every polygon in the original model can be triangulated as part of a preprocessing phase. To achieve more reliable results, when corners of two faces intersect at a point, the faces should be defined as sharing a single vertex rather than using two separate vertices which happen to be coincident in space.

We have developed an algorithm which produces simplified versions of such polygonal models. Our algorithm is based on the iterative contraction of vertex pairs (a generalization of edge contraction). As the algorithm proceeds, a geometric error approximation is maintained at each vertex of the current model. This error approximation is represented using quadric matrices. The primary advantages of our algorithm are:

- *Efficiency:* The algorithm is able to simplify complex models quite rapidly. For example, our implementation can create a 100 face approximation of a 70,000 face model in 15 seconds. The error approximation is also very compact, requiring only 10 floating point numbers per vertex.
- *Quality:* The approximations produced by our algorithm maintain high fidelity to the original model. The primary features of the model are preserved even after significant simplification.
- *Generality:* Unlike most other surface simplification algorithms, ours is able to join unconnected regions of the model together, a process which we term *aggregation*. Provided that maintaining object topology is not an important concern, this can facilitate better approximations of models with many disconnected components. This also requires our algorithm to support non-manifold[1] models.

2 Background and Related Work

The goal of polygonal surface simplification is to take a polygonal model as input and generate a simplified model (i.e., an approximation of the original) as output. We assume that the input model (M_n) has been triangulated. The target approximation (M_g) will satisfy some given target criterion which is typically either a desired face count or a maximum tolerable error. We are interested in surface simplification algorithms that can be used in rendering systems for *multiresolution modeling* — the generation of models with appropriate levels of detail for the current context.

We do not assume that the topology of the model must be maintained. In certain application areas, medical imaging for example, maintaining the object topology can be essential. However, in application areas such as rendering, topology is less important than overall appearance. Our algorithm is capable of both closing topological holes as well as joining unconnected regions.

Many prior simplification algorithms have either implicitly or explicitly assumed that their input surfaces were, and ought to remain, manifold surfaces. Let us stress that we do not make this assumption. In fact, the process of aggregation will regularly create non-manifold regions.

2.1 Surface Simplification

In recent years, the problem of surface simplification, and the more general problem of multiresolution modeling, has received increas-

*garland@cs.cmu.edu; http://www.cs.cmu.edu/∼garland/

†ph@cs.cmu.edu; http://www.cs.cmu.edu/∼ph/

[1] A *manifold* is a surface for which the infinitesimal neighborhood of every point is topologically equivalent to a disk (or half-disk for a *manifold with boundary*).

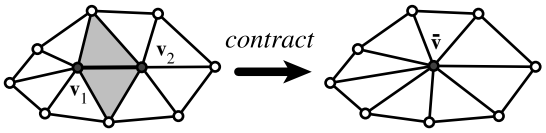

Figure 1: **Edge contraction.** The highlighted edge is contracted into a single point. The shaded triangles become degenerate and are removed during the contraction.

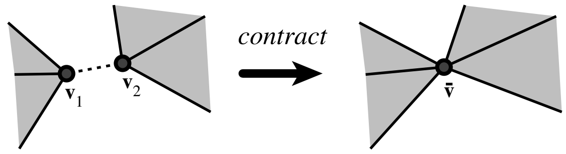

Figure 2: **Non-edge contraction.** When non-edge pairs are contracted, unconnected sections of the model are joined. The dashed line indicates the two vertices being contracted together.

ing attention. Several different algorithms have been formulated for simplifying surfaces. Those algorithms which are most relevant to our work can be broadly categorized into 3 classes:

Vertex Decimation. Schroeder *et al.* [9] describe an algorithm which we would term *vertex decimation*. Their method iteratively selects a vertex for removal, removes all adjacent faces, and retriangulates the resulting hole. Soucy and Laurendeau [10] described a more sophisticated, but essentially similar algorithm. While they provide reasonable efficiency and quality, these methods are not really suited for our purpose. Both methods use vertex classification and retriangulation schemes which are inherently limited to manifold surfaces, and they carefully maintain the topology of the model. While these are important features in some domains, they are restrictions for multiresolution rendering systems.

Vertex Clustering. The algorithm described by Rossignac and Borrel [8] is one of the few capable of processing arbitrary polygonal input. A bounding box is placed around the original model and divided into a grid. Within each cell, the cell's vertices are clustered together into a single vertex, and the model faces are updated accordingly. This process can be very fast, and can make drastic topological alterations to the model. However, while the size of the grid cells does provide a geometric error bound, the quality of the output is often quite low. In addition, it is difficult to construct an approximation with a specific face count, since the number of faces is only indirectly determined by the specified grid dimensions. The exact approximation produced is also dependent on the exact position and orientation of the original model with respect to the surrounding grid. This uniform method can easily be generalized to use an adaptive grid structure, such as an octree [6]. This can improve the simplification results, but it still does not support the quality and control that we desire.

Iterative Edge Contraction. Several algorithms have been published that simplify models by iteratively contracting edges (see Figure 1). The essential difference between these algorithms lies in how they choose an edge to contract. Some notable examples of such algorithms are those of Hoppe [4, 3], Ronfard and Rossignac [7], and Guéziec [2]. These algorithms all seem to have been designed for use on manifold surfaces, although edge contractions can be utilized on non-manifold surfaces. By performing successive edge contractions, they can close holes in the object but they cannot join unconnected regions.

If it is critical that the approximate model lie within some distance of the original model and that its topology remain unchanged, the simplification envelopes technique of Cohen *et al.* [1] can be used in conjunction with one of the above simplification algorithms. As long as any modification made to the model is restricted to lie within the envelopes, a global error guarantee can be maintained. However, while this provides strong error limits, the method is inherently limited to orientable manifold surfaces and carefully preserves model topology. Again, these are often limitations for the purposes of simplification for rendering.

None of these previously developed algorithms provide the combination of efficiency, quality, and generality that we desire. Vertex decimation algorithms are unsuitable for our needs; they are careful to maintain model topology and usually assume manifold geometry. Vertex clustering algorithms are very general and can be very fast. However, they provide poor control over their results and these results can be of rather low quality. Edge contraction algorithms can not support aggregation.

We have developed an algorithm which supports both aggregation and high quality approximations. It possesses much of the generality of vertex clustering as well as the quality and control of iterative contraction algorithms. It also allows faster simplification than some higher quality methods [3].

3 Decimation via Pair Contraction

Our simplification algorithm is based on the iterative contraction of vertex pairs; a generalization of the iterative edge contraction technique used in previous work. A *pair contraction*, which we will write $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$, moves the vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ to the new position $\bar{\mathbf{v}}$, connects all their incident edges to $\mathbf{v}_1$, and deletes the vertex $\mathbf{v}_2$. Subsequently, any edges or faces which have become degenerate are removed. The effect of a contraction is small and highly localized. If $(\mathbf{v}_1, \mathbf{v}_2)$ is an edge, then 1 or more faces will be removed (see Figure 1). Otherwise, two previously separate sections of the model will be joined at $\bar{\mathbf{v}}$ (see Figure 2).

This notion of contraction is in fact quite general; we can contract a set of vertices into a single vertex: $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k) \rightarrow \bar{\mathbf{v}}$. This form of generalized contraction can express both pair contractions as well as more general operations such as vertex clustering. However, we use pair contraction as the atomic operation of our algorithm because it is the most fine-grained contraction operation.

Starting with the initial model M_n, a sequence of pair contractions is applied until the simplification goals are satisfied and a final approximation M_g is produced. Because each contraction corresponds to a local incremental modification of the current model, the algorithm actually generates a sequence of models $M_n, M_{n-1}, \ldots, M_g$. Thus, a single run can produce a large number of approximate models or a multiresolution representation such as a progressive mesh [3].

3.1 Aggregation

The primary benefit which we gain by utilizing general vertex pair contractions is the ability of the algorithm to join previously unconnected regions of the model together. A potential side benefit is that it makes the algorithm less sensitive to the mesh connectivity of the original model. If in fact two faces meet at a vertex which is duplicated, the contraction of that pair of vertices will repair this shortcoming of the initial mesh.

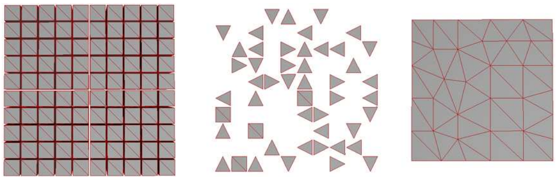

Figure 3: On the left is a regular grid of 100 closely spaced cubes. In the middle, an approximation built using only edge contractions demonstrates unacceptable fragmentation. On the right, the result of using more general pair contractions to achieve aggregation is an approximation much closer to the original.

In some applications, such as rendering, topology may be less important than overall shape. Consider a shape such as that shown in Figure 3 which is made up of 100 closely spaced cubes in a regular grid. Suppose we wanted to construct an approximation of the model on the left for rendering at a distance. Algorithms based on edge contraction can close holes in objects, but they can never join disconnected components. In an algorithm using only edge contraction, the individual components are individually simplified into nothing, as in the model in the middle. Using pair contraction, the individual components can be merged into a single object, as in the model on the right. The result is a much more faithful approximation.

Allowing aggregation also requires us to support non-manifold surfaces. At the instant when two separate regions are joined, a non-manifold region is quite likely to be created. It would require a great deal of care and effort to ensure that a contraction never created a non-manifold region without severely limiting the kinds of contractions that we could perform.

3.2 Pair Selection

We have chosen to select the set of valid pairs at initialization time, and to consider only these pairs during the course of the algorithm. Our decision is based on the assumption that, in a good approximation, points do not move far from their original positions.

We will say that a pair $(\mathbf{v}_1, \mathbf{v}_2)$ is a *valid* pair for contraction if either:

1. $(\mathbf{v}_1, \mathbf{v}_2)$ is an edge, *or*

2. $\|\mathbf{v}_1 - \mathbf{v}_2\| < t$, where t is a threshold parameter

Using a threshold of $t = 0$ gives a simple edge contraction algorithm. Higher thresholds allow non-connected vertices to be paired. Naturally, this threshold must be chosen with some care; if it is too high, widely separated portions of the model could be connected, which is presumably undesirable, and it could create $O(n^2)$ pairs.

We must track the set of valid pairs during the course of iterative contraction. With each vertex, we associate the set of pairs of which it is a member. When we perform the contraction $(\mathbf{v}_1, \mathbf{v}_2) \to \bar{\mathbf{v}}$, not only does $\mathbf{v}_1$ acquire all the edges that were linked to $\mathbf{v}_2$, it also merges the set of pairs from $\mathbf{v}_2$ into its own set. Every occurrence of $\mathbf{v}_2$ in a valid pair is replaced by $\mathbf{v}_1$, and duplicate pairs are removed.

4 Approximating Error With Quadrics

In order to select a contraction to perform during a given iteration, we need some notion of the *cost* of a contraction. To define this cost, we attempt to characterize the error at each vertex. To do this, we associate a symmetric 4×4 matrix $\mathbf{Q}$ with each vertex, and we define the error at vertex $\mathbf{v} = [v_x\ v_y\ v_z\ 1]^\mathsf{T}$ to be the quadratic form $\Delta(\mathbf{v}) = \mathbf{v}^\mathsf{T}\mathbf{Q}\mathbf{v}$. In Section 5, we will describe how the initial matrices are constructed. Note that the level surface $\Delta(\mathbf{v}) = \epsilon$, which is the set of all points whose error with respect to $\mathbf{Q}$ is ϵ, is a quadric surface.

For a given contraction $(\mathbf{v}_1, \mathbf{v}_2) \to \bar{\mathbf{v}}$, we must derive a new matrix $\bar{\mathbf{Q}}$ which approximates the error at $\bar{\mathbf{v}}$. We have chosen to use the simple additive rule $\bar{\mathbf{Q}} = \mathbf{Q}_1 + \mathbf{Q}_2$.

In order to perform the contraction $(\mathbf{v}_1, \mathbf{v}_2) \to \bar{\mathbf{v}}$, we must also choose a position for $\bar{\mathbf{v}}$. A simple scheme would be to select either $\mathbf{v}_1$, $\mathbf{v}_2$, or $(\mathbf{v}_1 + \mathbf{v}_2)/2$ depending on which one of these produces the lowest value of $\Delta(\bar{\mathbf{v}})$. However, it would be nice to find a position for $\bar{\mathbf{v}}$ which minimizes $\Delta(\bar{\mathbf{v}})$. Since the error function Δ is quadratic, finding its minimum is a linear problem. Thus, we find $\bar{\mathbf{v}}$ by solving $\partial\Delta/\partial x = \partial\Delta/\partial y = \partial\Delta/\partial z = 0$. This is equivalent[2] to solving:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{v}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

for $\bar{\mathbf{v}}$. The bottom row of the matrix is empty because $\bar{\mathbf{v}}$ is an homogeneous vector — its w component is always 1. Assuming that this matrix is invertible, we get that

$$\bar{\mathbf{v}} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

If this matrix is not invertible, we attempt to find the optimal vertex along the segment $\mathbf{v}_1\mathbf{v}_2$. If this also fails, we fall back on choosing $\bar{\mathbf{v}}$ from amongst the endpoints and the midpoint.

4.1 Algorithm Summary

Our simplification algorithm is built around pair contractions and error quadrics. The current implementation represents models using an adjacency graph structure: vertices, edges, and faces are all explicitly represented and linked together. To track the set of valid pairs, each vertex maintains a list of the pairs of which it is a member. The algorithm itself can be quickly summarized as follows:

1. Compute the $\mathbf{Q}$ matrices for all the initial vertices.

2. Select all valid pairs.

3. Compute the optimal contraction target $\bar{\mathbf{v}}$ for each valid pair $(\mathbf{v}_1, \mathbf{v}_2)$. The error $\bar{\mathbf{v}}^\mathsf{T}(\mathbf{Q}_1 + \mathbf{Q}_2)\bar{\mathbf{v}}$ of this target vertex becomes the *cost* of contracting that pair.

4. Place all the pairs in a heap keyed on cost with the minimum cost pair at the top.

5. Iteratively remove the pair $(\mathbf{v}_1, \mathbf{v}_2)$ of least cost from the heap, contract this pair, and update the costs of all valid pairs involving $\mathbf{v}_1$.

The only remaining issue is how to compute the initial $\mathbf{Q}$ matrices from which the error metric Δ is constructed.

[2] You can verify this for yourself by taking partial derivatives of

$$\begin{aligned} \mathbf{v}^\mathsf{T}\mathbf{Q}\mathbf{v} &= q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 \\ &+ 2q_{23}yz + 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44} \end{aligned}$$

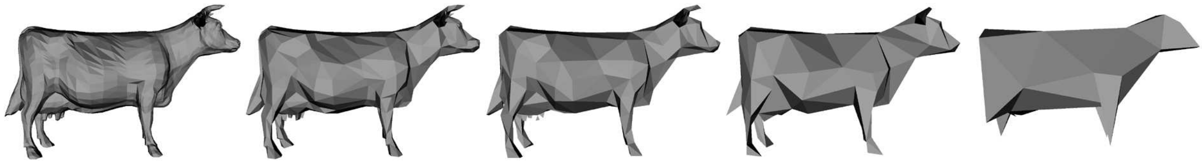

Figure 4: A sequence of approximations generated using our algorithm. The original model on the left has 5,804 faces. The approximations to the right have 994, 532, 248, and 64 faces respectively. Note that features such as horns and hooves continue to exist through many simplifications. Only at extremely low levels of detail do they begin to disappear.

5 Deriving Error Quadrics

To construct our error quadrics, we must choose a heuristic to characterize the geometric error. We have selected a heuristic which is quite similar to the one given by Ronfard and Rossignac [7]. Following [7], we can observe that in the original model, each vertex is the solution of the intersection of a set of planes — namely, the planes of the triangles that meet at that vertex. We can associate a set of planes with each vertex, and we can define the error of the vertex with respect to this set as the sum of squared distances to its planes:

$$\Delta(\mathbf{v}) = \Delta([v_x\ v_y\ v_z\ 1]^\mathsf{T}) = \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} (\mathbf{p}^\mathsf{T}\mathbf{v})^2 \tag{2}$$

where $\mathbf{p} = [a\ b\ c\ d]^\mathsf{T}$ represents the plane defined by the equation $ax + by + cz + d = 0$ where $a^2 + b^2 + c^2 = 1$. This approximate error metric is similar to [7], although we have used a summation rather than a maximum over the set of planes. The set of planes at a vertex is initialized to be the planes of the triangles that meet at that vertex. Note that if we were to track these plane sets explicitly, as [7] did, we would propagate planes after a contraction $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$ using the rule: $\text{planes}(\bar{\mathbf{v}}) = \text{planes}(\mathbf{v}_1) \cup \text{planes}(\mathbf{v}_2)$. This can require a sizeable amount of storage that does not diminish as simplification progresses.

The error metric given in (2) can be rewritten as a quadratic form:

$$\begin{aligned}
\Delta(\mathbf{v}) &= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} (\mathbf{v}^\mathsf{T}\mathbf{p})(\mathbf{p}^\mathsf{T}\mathbf{v}) \\
&= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{v}^\mathsf{T}(\mathbf{p}\mathbf{p}^\mathsf{T})\mathbf{v} \\
&= \mathbf{v}^\mathsf{T}\left(\sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{K}_\mathbf{p}\right)\mathbf{v}
\end{aligned}$$

where $\mathbf{K}_\mathbf{p}$ is the matrix:

$$\mathbf{K}_\mathbf{p} = \mathbf{p}\mathbf{p}^\mathsf{T} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

This *fundamental error quadric* $\mathbf{K}_\mathbf{p}$ can be used to find the squared distance of any point in space to the plane $\mathbf{p}$. We can sum these fundamental quadrics together and represent an entire set of planes by a single matrix $\mathbf{Q}$.

We implicitly track sets of planes using a single matrix; instead of computing a set union $(\text{planes}(\mathbf{v}_1) \cup \text{planes}(\mathbf{v}_2))$ we simply add two quadrics $(\mathbf{Q}_1 + \mathbf{Q}_2)$. If the sets represented by $\mathbf{Q}_1$ and $\mathbf{Q}_2$ in the original metric are disjoint, the quadric addition is equivalent to the set union. If there is some overlap, then a single plane may be counted multiple times. However, any single plane can be counted at most 3 times since each plane is initially distributed only to the vertices of its defining triangle. This may introduce some imprecision into the error measurement, but it has major benefits: the space required to track a plane set is only that required for a 4×4 symmetric matrix (10 floating point numbers), and the cost of updating the approximation is only that for adding two such matrices. If we are willing to sacrifice some additional storage, it would even be possible to eliminate this multiple counting using an inclusion-exclusion formula.

Thus, to compute the initial $\mathbf{Q}$ matrices required for our pair contraction algorithm, each vertex must accumulate the planes for the triangles which meet at that vertex. For each vertex, this set of planes defines several fundamental error quadrics $\mathbf{K}_\mathbf{p}$. The error quadric $\mathbf{Q}$ for this vertex is the sum of the fundamental quadrics. Note that the initial error estimate for each vertex is 0, since each vertex lies in the planes of all its incident triangles.

5.1 Geometric Interpretation

As we will see, our plane-based error quadrics produce fairly high quality approximations. In addition, they also possess a useful geometric meaning[3].

The level surfaces of these quadrics are almost always ellipsoids. In some circumstances, the level surfaces may be degenerate. For instance, parallel planes (e.g., around a planar surface region) will produce level surfaces which are two parallel planes, and planes which are all parallel to a line (e.g., around a linear surface crease) will produce cylindrical level surfaces. The matrix used for finding optimal vertex positions (Eq. 1) will be invertible as long as the level surfaces are non-degenerate ellipsoids. In this case, $\bar{\mathbf{v}}$ will be at the center of the ellipsoid.

6 Additional Details

The general algorithm outlined so far performs well on most models. However, there are a few important enhancements which improve its performance on certain types of models, particularly planar models with open boundaries.

Preserving Boundaries. The error quadrics derived earlier do not make any allowance for open boundaries. For models such as terrain height fields, it is necessary to preserve boundary curves while simplifying their shape. We might also wish to preserve discrete color discontinuities. In such cases, we initially label each edge as either normal or as a "discontinuity". For each face surrounding a particular discontinuity edge, we generate a perpendicular plane running through the edge. These constraint planes are then converted into quadrics, weighted by a large penalty factor, and

[3] Kalvin and Taylor [5] describe a somewhat similar use of quadrics to represent plane sets. They were tracking sets of planes which fit a set of points within some tolerance. They used ellipsoids in plane-space to represent the set of valid approximating planes.

Model	Faces	t	Init (s)	Simplify (s)
Bunny	69,451	0	3.3	12.0
Crater Lake	199,114	0	10.6	36.0
Cow	5,804	0	0.22	0.69
Cube Grid	1,200	0.12	0.25	0.17
Foot	4,204	0	0.16	0.41
Foot	4,204	0.318	0.43	0.76

Figure 5: Sample running times. All data reflects the time needed to make a 10 face approximation of the given model. Initialization time includes selecting valid pairs, computing initial error matrices, and choosing contraction targets. Simplification time includes the iterative contraction of pairs.

Faces (i)	Fixed (E_i)	Optimal (E_i)	Reduction
10	0.0062	0.0054	13.4%
100	0.00032	0.00025	21.7%
500	2.4e-05	1.3e-05	47.6%
1000	5.7e-06	3.4e-06	40.3%
2000	1.2e-06	7.9e-07	32.4%
3000	3.6e-07	2.6e-07	28.2%

Figure 6: Effect of optimal vertex placement. Choosing an optimal position, rather than a fixed choice amongst the endpoints and midpoint, can significantly reduce approximation error. Approximations are of cow model using $t = 0$.

added into the initial quadrics for the endpoints of the edge. We have found that this works quite well.

Preventing Mesh Inversion. Pair contractions do not necessarily preserve the orientation of the faces in the area of the contraction. For instance, it is possible to contract an edge and cause some neighboring faces to fold over on each other. It is usually best to try to avoid this type of mesh inversion. We use essentially the same scheme as others have before ([7] for example). When considering a possible contraction, we compare the normal of each neighboring face before and after the contraction. If the normal flips, that contraction can be either heavily penalized or disallowed.

6.1 Evaluating Approximations

In order to evaluate the quality of approximations produced by our algorithm, we need an error measurement which is a bit more rigorous than the heuristic error measurement employed by the algorithm itself. We have chosen a metric which measures the average squared distance between the approximation and the original model. This is very similar to the E_{dist} energy term used by Hoppe *et al.* [4]. We define the approximation error $E_i = E(M_n, M_i)$ of the simplified model M_i as:

$$E_i = \frac{1}{|X_n| + |X_i|}\left(\sum_{v \in X_n} d^2(v, M_i) + \sum_{v \in X_i} d^2(v, M_n)\right)$$

where X_n and X_i are sets of points sampled on the models M_n and M_i respectively. The distance $d(v, M) = \min_{p \in M} \|v - p\|$ is the minimum distance from v to the closest face of M ($\|\cdot\|$ is the usual Euclidean vector length operator). We use this metric for evaluation purposes only; it plays no part in the actual algorithm.

7 Results

Our algorithm can produce high fidelity approximations in fairly short amounts of time. Figure 5 summarizes the running time[4] of our current implementation using the models shown in this paper. Figure 4 shows a sample sequence of approximations generated by our algorithm. This entire sequence of cows was constructed in about a second. Notice that features such as horns and hooves remain recognizable through many simplifications. Only at extremely low levels of detail do they begin to disappear.

As described earlier, our algorithm attempts to optimize the placement of vertices after contractions. Figure 6 summarizes the effect of this policy on approximations for a representative model (the cow model of Fig. 4). At extremely low levels of detail, the effect is modest. However, at more reasonable levels of detail the effect is substantial; optimal vertex placement can reduce the overall error by as much as 50%. In our experiments, we have also found that using optimal vertex placement tends to produce more well-shaped meshes.

Figures 8–13 demonstrate the performance of our algorithm on much larger models. Figure 9 represents a significant simplification (1.4%) of the original, but notice that all the major details of the original remain. In particular, notice that the contours on the interior of the ear and the large contours around the rear leg have been preserved. Figure 10 presents an even more drastic approximation: 100 faces or 0.14% of the original size. While most of the detail of the model has disappeared, the basic structure of the object is still intact; the major features such as the head, legs, tail, and ears are all apparent, albeit highly simplified. Figure 12 shows a very densely tessellated terrain model, and Figure 13 shows a highly simplified version. While the small scale texture of the terrain has largely disappeared, all the basic features of the terrain remain. Also note that the open boundary has been properly preserved. Without the boundary constraints described earlier, the boundary would have been significantly eroded.

Figure 11 illustrates the nature of the error quadrics accumulated during simplification. The level surface $\mathbf{v}^{\mathsf{T}}\mathbf{Q}\mathbf{v} = \epsilon$ is shown for each vertex. Each level surface is an ellipsoid centered around the corresponding vertex. The interpretation of these ellipsoids is that a vertex can be moved anywhere within its ellipsoid and have an error less than ϵ. The significant feature of the ellipsoids is that they conform to the shape of the model surface very nicely. They are large and flat on mostly planar areas such as the middle of the hind leg, and they are elongated along discontinuity lines such as the contours through the ear and along the bottom of the leg. In some sense, these ellipsoids can be thought of as accumulating information about the shape of the local surface around their vertex.

Figures 14–17 demonstrate the real benefits which aggregation via pair contractions can provide. The original model consists of the bones of a human's left foot. There are many separate bone segments, some of which even interpenetrate (obviously an error in model reconstruction). Three approximations are compared: one built by uniform vertex clustering, one built with edge contractions alone, and one built with pair contractions. When edge contractions alone are used, the small segments at the ends of the toes collapse into single points; this creates the impression that the toes are slowly receding back into the foot. On the other hand, using the more general pair contractions allows separate bone segments to be merged. As you can see in Figure 17, the toes are being merged into single long segments. Figure 19 shows some of the pairs initially selected as valid. Notice how they span the gaps between bone segments. Finally, Figure 18 shows another illustration of the level surfaces of the error quadrics accumulated during simplification (this is the model seen in Figure 16).

The benefits of aggregation are not solely visual. Aggregation can also produce objectively lower approximation error. Figure 7

[4] This data was collected on an SGI Indigo2 with a 195 MHz R10000 processor and 128 Mbytes of memory.

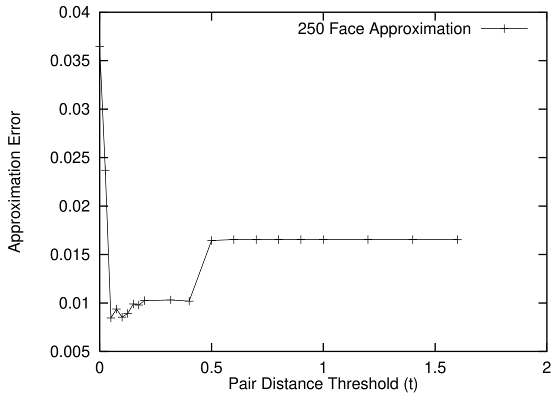

Figure 7: The effect of pair thresholds. Approximation error E_{250} is shown as a function of t for a 250 face approximation of the foot model (Fig. 14). Pair contractions resulting in aggregation can significantly reduce approximation error.

shows the error E_{250} for 250 face approximations of the foot model using various values of t. A fairly wide range of values all produce objectively better approximations than are achieved by the edge contraction only approximation ($t = 0$). Notice, however, that increasing t does not always improve the approximation. This is due to the nature of our quadric error metric. Locally, the distance to a set of planes is a reasonable approximation of the distance to a set of faces. However, because planes have infinite extent and faces do not, the error metric becomes less reliable as we move farther away.

8 Discussion

Our algorithm provides a mix of efficiency, quality, and generality not found in earlier algorithms. While certain other algorithms are faster or generate higher quality approximations than ours, they typically do not meet the capability of our algorithm in all three areas. The only algorithm capable of simplifying arbitrary polygonal models, vertex clustering [8], does not reliably produce high quality approximations. None of the higher quality methods available [2, 7, 3] support aggregation. Both [2] and [3] seem to be significantly more time consuming than our algorithm. The results of [7] are most similar to our own because we use a very similar error approximation. However, our system uses a more efficient means to track plane sets, and it incorporates some enhancements such as boundary preservation.

There remain various aspects of our algorithm that could be improved upon. We have used a fairly simple scheme for selecting valid pairs. It is quite possible that a more sophisticated adaptive scheme could produce better results. We have not addressed the issue of surface properties such as color. One possible solution is to treat each vertex as a vector (x, y, z, r, g, b), construct 7×7 quadrics, and simplify. We believe this could work well, but the added size and complexity make it less attractive than the basic algorithm.

Although it generally performs well, our algorithm has a couple of clear weaknesses. First, as mentioned earlier, measuring error as a distance to a set of planes only works well in a suitably local neighborhood. Second, the information accumulated in the quadrics is essentially implicit, which can at times be problematic. Suppose we join together two cubes and would like to remove the planes associated with the now defunct interior faces. Not only is it, in general, difficult to determine what faces are defunct, there is no clear way to reliably remove the appropriate planes from the quadrics. As a result, our algorithm does not do as good a job at simplification with aggregation as we would like.

9 Conclusion

We have described a surface simplification algorithm which is capable of rapidly producing high fidelity approximations of polygonal models. Our algorithm uses iterative pair contractions to simplify models and quadric error metrics to track the approximate error of the model as it is being simplified. The quadrics stored with the final vertices can also be used to characterize the overall shape of the surface.

Our algorithm has the ability to join unconnected sections of models while still maintaining fairly high quality results. While most previous algorithms are also inherently limited to manifold surfaces, our system is quite capable of handling and simplifying non-manifold objects. Finally, our algorithm provides a useful middle ground between very fast, low-quality methods such as vertex clustering [8] and very slow, high-quality methods such as mesh optimization [3].

References

[1] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *SIGGRAPH '96 Proc.*, pages 119–128, Aug. 1996. http://www.cs.unc.edu/~geom/envelope.html.

[2] André Guéziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.

[3] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96 Proc.*, pages 99–108, Aug. 1996. http://www.research.microsoft.com/research/graphics/hoppe/.

[4] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proc.*, pages 19–26, Aug. 1993. http://www.research.microsoft.com/research/graphics/hoppe/.

[5] Alan D. Kalvin and Russell H. Taylor. Superfaces:polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), May 1996. http://www.computer.org/pubs/cg&a/articles/g30064.pdf.

[6] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Proc.*, August 1997.

[7] Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3), Aug. 1996. Proc. Eurographics '96.

[8] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

[9] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, July 1992.

[10] Marc Soucy and Denis Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63(1):1–14, 1996.

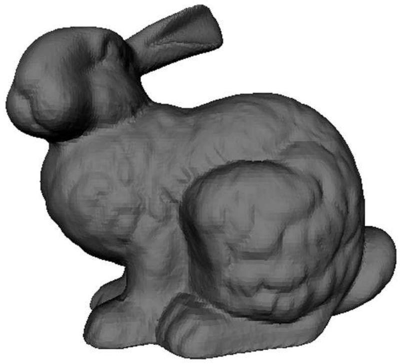

Figure 8: Original bunny model with 69,451 triangles. Rendered using flat shading just as in approximations below.

Figure 9: An approximation using only 1,000 triangles (generated in 15 seconds).

Figure 10: An approximation using only 100 triangles (generated in 15 seconds).

Figure 11: 1,000 face approximation. Error ellipsoids for each vertex are shown in green.

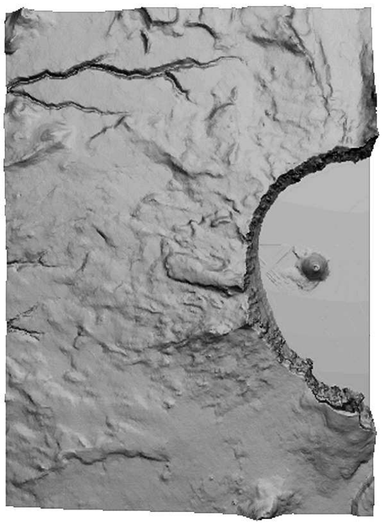

Figure 12: Terrain model of Crater Lake (199,114 faces).

Figure 13: Simplified model with 999 faces (took 46 seconds).

Figure 14: **Original.** Bones of a human's left foot (4,204 faces). Note the many separate bone segments.

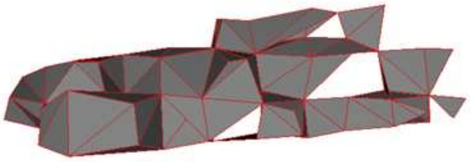

Figure 15: **Uniform Vertex Clustering.** 262 face approximation ($11 \times 4 \times 4$ grid). Indiscriminate joining destroys approximation quality.

Figure 16: **Edge Contractions.** 250 face approximation. Bone segments at the ends of the toes have disappeared; the toes appear to be receding back into the foot.

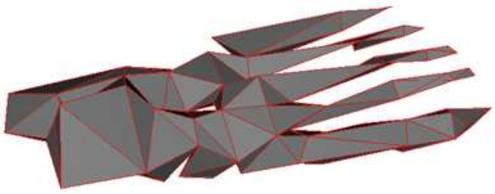

Figure 17: **Pair Contractions.** 250 face approximation ($t = 0.318$). Toes are being merged into larger solid components. No receding artifacts. This model now contains 61 non-manifold edges.

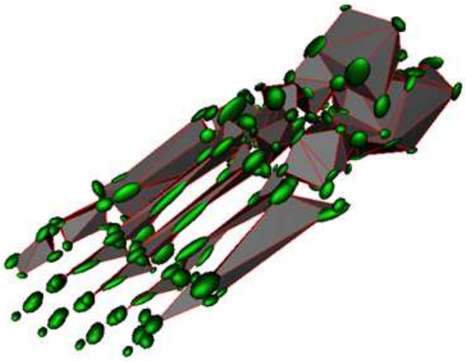

Figure 18: Level surfaces of the error quadrics at the vertices of the approximation shown in Figure 16.

Figure 19: Pairs selected as valid during initialization (for Fig. 17). Red pairs are edges; green pairs are non-edges.