Simplifying Surfaces with Color and Texture using Quadric Error Metrics

Michael Garland* Paul S. Heckbert†

Carnegie Mellon University

Abstract

There are a variety of application areas in which there is a need for simplifying complex polygonal surface models. These models often have material properties such as colors, textures, and surface normals. Our surface simplification algorithm, based on iterative edge contraction and quadric error metrics, can rapidly produce high quality approximations of such models. We present a natural extension of our original error metric that can account for a wide range of vertex attributes.

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

Keywords: surface simplification, multiresolution modeling, level of detail, quadric error metric, edge contraction, surface properties, discontinuity preservation

1 INTRODUCTION

Many applications in computer graphics and visualization can benefit from automatic simplification of complex polygonal models. Such models are usually not only geometrically complex, but they may also have various surface properties such as colors, textures, and surface normals. Scanning and acquisition methods often produce surface meshes that are much more dense than actually required for the intended application. Computer games and distributed virtual environments must often operate on systems where rendering and transmission capacity is highly constrained and therefore require strict control over the level of detail used in models. Realistic simulation systems typically have object databases that far exceed the capacity of even the most powerful graphics workstations.

In recent years, a variety of surface simplification algorithms have been developed. Among these, our algorithm, based on iterative edge contraction and quadric error metrics, provides a solution which is a practical mixture of efficiency and quality. In this paper, we present a generalization of the basic error metric developed in our previous paper [6] that is capable of simplifying surfaces with vertex properties such as color and texture.

2 BACKGROUND AND RELATED WORK

In this paper, we are primarily concerned with the problem of polygonal surface simplification. Given an initial triangulated surface, we want to generate a simplified model that, as much as possible, faithfully reproduces the features of the original.

An assortment of simplification algorithms have been proposed in recent years. Most of those which are applicable to arbitrary 3D surfaces can be broadly classified into 3 categories. *Vertex clustering* methods [17] spatially partition vertices and unify vertices within the same cluster. They are generally very fast and work on arbitrary collections of triangles, but can often produce relatively poor results. *Vertex decimation* algorithms [18] select unimportant vertices (based on local shape heuristics), remove them, and retriangulate the resulting holes. These methods tend to produce fair results and are reasonably efficient. *Iterative edge contraction* [6, 7, 8, 15, 16] has been widely used. Edges are ranked according to their cost, which is typically the amount of error introduced into the model as a result of contracting the edge. At each iteration the lowest-cost edge is contracted and the costs of neighboring edges are updated. The primary difference between the various contraction-based methods is the error metric used for ranking edges. These algorithms generally produce good results, although running time varies greatly between methods. Iterative edge contraction is also particularly attractive because it naturally leads to a useful multiresolution model representation [8, 9, 19].

There has been comparatively less work done on simplifying models with material properties. For restricted surface classes, such as height fields [5], very simple methods can work quite well. However, more general surfaces require more advanced techniques. Hoppe [8] explicitly included attributes in his error metric. Certain *et al.* [1] discussed adding surface color to a wavelet-based multiresolution framework, and Hughes *et al.* [10] investigated the simplification of Gouraud-shaded meshes produced by radiosity simulations. Cohen *et al.* [2] developed an algorithm capable of reparameterizing texture maps as a surface is simplified.

In this paper, we begin by restating our basic algorithm [6] and next provide an expanded analysis of its behavior. We then present the generalization of our quadric error metric to handle vertex attributes, and finish with discussion, results, and summary of our work.

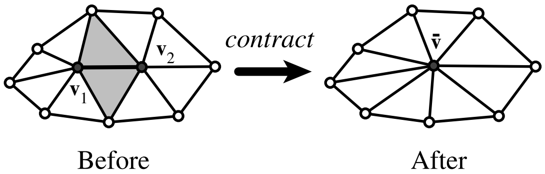


Figure 1: Contraction of the edge $(\mathbf{v}_1, \mathbf{v}_2)$ into a single point. The shaded triangles become degenerate and are removed during the contraction.

3 FUNDAMENTAL ALGORITHM

Our algorithm, originally developed in [6], belongs to the class of iterative edge contraction methods. Every edge is assigned a cost, namely the error resulting from its contraction. To efficiently track the lowest-cost edge, we maintain all edges in a heap keyed on cost. At each iteration, we can extract the lowest-cost edge and contract it. Our error metric offers a compromise between very fast low-quality methods and very slow high-quality methods. Moreover, it provides a useful characterization of the local shape and error of the current approximation.

The fundamental operation of our algorithm is *edge contraction*, which we will write $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$. To perform this simple con-

*garland@cs.cmu.edu; http://www.cs.cmu.edu/∼garland/; Computer Science Dept., Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213.

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

traction, we need to perform the following three steps: (1) move vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ to the position $\bar{\mathbf{v}}$, (2) replace all occurrences of $\mathbf{v}_2$ with $\mathbf{v}_1$, and (3) delete $\mathbf{v}_2$ and any degenerate faces. Figure 1 illustrates the effect of a single edge contraction. As has been recently observed [6, 15], we can just as easily consider the contraction of any arbitrary pair of vertices. However, we will not consider the case of non-edge pairs in this paper. Our experience has shown that, while greedy edge contraction produces consistently good results on many kinds of models, greedy contraction of arbitrary pairs is not as robust and does not perform as consistently.

3.1 Basic Quadric Error Metric

We characterize the geometric error of an approximation using the metric described in our previous paper [6] which is based on the metric of Ronfard and Rossignac [16]. Conceptually, we associate a set of planes with every vertex of the model. The error at that vertex is defined to be the sum of squared distances of the vertex to all the planes in its set. Each set is initialized with the faces incident to the corresponding vertex in the original model. When an edge is contracted into a single vertex, the resulting set is the union of the two sets associated with the endpoints.

However, these "sets" are purely conceptual; we do not represent them explicitly. Each face in the original model defines a plane which satisfies the equation[1] $\mathbf{n}^\mathsf{T}\mathbf{v} + d = 0$, where $\mathbf{n} = [n_x\ n_y\ n_z]^\mathsf{T}$ is a unit normal and d is a constant. The squared distance of a vertex $\mathbf{v} = [x\ y\ z]^\mathsf{T}$ to this plane is given by

$$D^2 = (\mathbf{n}^\mathsf{T}\mathbf{v}+d)^2 = (\mathbf{v}^\mathsf{T}\mathbf{n}+d)(\mathbf{n}^\mathsf{T}\mathbf{v}+d) = \mathbf{v}^\mathsf{T}(\mathbf{n}\mathbf{n}^\mathsf{T})\mathbf{v} + 2d\mathbf{n}^\mathsf{T}\mathbf{v} + d^2.$$

This is a quadratic form, plus a linear term, plus a constant. We can conveniently represent D^2 using the *quadric* Q:

$$Q = (\mathbf{A}, \mathbf{b}, c) = (\mathbf{n}\mathbf{n}^\mathsf{T}, d\mathbf{n}, d^2)$$

$$Q(\mathbf{v}) = \mathbf{v}^\mathsf{T}\mathbf{A}\mathbf{v} + 2\mathbf{b}^\mathsf{T}\mathbf{v} + c.$$

This requires 10 coefficients to store the symmetric 3×3 matrix $\mathbf{A}$, the 3-vector $\mathbf{b}$, and the scalar c.

The addition of quadrics can be naturally defined componentwise: $Q_1(\mathbf{v}) + Q_2(\mathbf{v}) = (Q_1 + Q_2)(\mathbf{v})$, where $(Q_1 + Q_2) = (\mathbf{A}_1 + \mathbf{A}_2, \mathbf{b}_1 + \mathbf{b}_2, c_1 + c_2)$. Thus, to compute the sum of squared distances to a set of planes, we only need one quadric which is the sum of the quadrics defined by each of the individual planes. And when contracting the edge $(\mathbf{v}_1, \mathbf{v}_2)$, the resulting quadric is merely the sum $Q = Q_1 + Q_2$. Furthermore, we can define the cost of a contraction $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$ as the error at $\bar{\mathbf{v}}$ which is $Q(\bar{\mathbf{v}}) = Q_1(\bar{\mathbf{v}}) + Q_2(\bar{\mathbf{v}})$.

This direct summing of quadrics can lead to the minor inaccuracy of double and triple counting [6] of the associated planes. However, we find the simple additive structure more appealing than the more complicated inclusion-exclusion rule [12] required to correct the problem. Also, to simplify the presentation we have assumed that all planes are uniformly weighted. However, in applications where surface triangles will be of significantly varying sizes, it is useful to weight quadrics by the area of the contributing triangle.

In practice, it is also necessary to check each proposed contraction and verify that it does not cause the mesh to fold over on itself. Previously, we suggested using a simple normal-flipping heuristic [6]. However, Edelsbrunner and Nekhayev [3] have developed a more effective procedure, as well as a technique for preventing topological changes to the mesh. A further check that the surface remains a manifold may be required if our algorithm is implemented using data structures that can only represent manifold surfaces.

[1] By convention, we will assume that all vectors are column vectors. Thus, $\mathbf{n}^\mathsf{T}\mathbf{v} = \mathbf{n}\cdot\mathbf{v}$ is the inner product of two vectors. We will generally use the transpose notation for the elements of the quadric equation and the dot notation elsewhere. The outer product $\mathbf{n}\mathbf{n}^\mathsf{T}$ produces a 3×3 matrix.

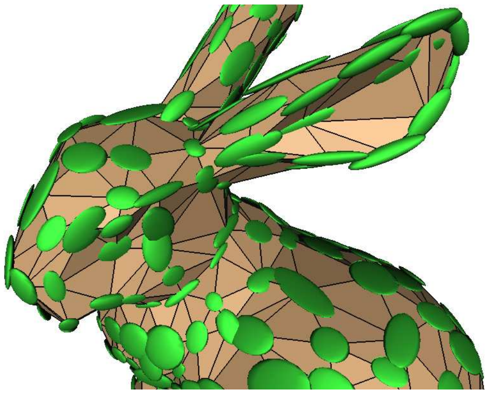


Figure 2: Result of simplifying a bunny model. Only 1.4% of the original faces remain. Centered around each vertex is an isosurface of the corresponding quadric.

3.2 Interpretation of Quadrics

The quadrics which we construct during simplification also possess a useful geometric interpretation. For a particular quadric, consider the level surface $Q(\mathbf{v}) = \epsilon$. This is the set of all points whose error with respect to Q is ϵ. The resulting isosurface is a (potentially degenerate) ellipsoid; the possible degenerate forms are cylinders and parallel planes. Algebraically, $\mathbf{A}$ is a symmetric positive semidefinite matrix whose eigenvalues and eigenvectors define the principal axes of the ellipsoids. Also note that, in statistical terminology, $\mathbf{A}$ is the sample covariance matrix of the face normals [11].

We have also suggested that quadrics characterize the local shape of the surface. This is apparent in Figure 2, which illustrates the quadric isosurfaces produced by the simplification of a bunny model. For vertices on creases, such as on the neck and ears, the ellipsoids are cigar shaped. They are elongated in the direction of the crease. In contrast, where the surface is less curved, such as on the forehead, the quadrics are thin and roughly circular, like pancakes. Intuitively, we might conclude that the quadrics will be elongated in directions of low curvature and thin in directions of high curvature. We can provide mathematical support for this conclusion. Let us suppose that our model is actually a differentiable manifold, the limit of an infinitely fine tessellation. We have shown [4] that, under suitable conditions, the two smallest eigenvalues of $\mathbf{A}$ are proportional to the squares of the principal curvatures [14] and the corresponding eigenvectors are the corresponding principal directions.

3.3 Vertex Placement

When considering the contraction of an edge $(\mathbf{v}_1, \mathbf{v}_2)$, we need some way of choosing the target position $\bar{\mathbf{v}}$. There are two primary policies to choose from, and the choice between them must be made with the intended application in mind. We must trade space efficiency against approximation quality.

Subset placement is the simplest strategy that we can adopt. We simply select one of the endpoints as the target position. In other words, we will contract one endpoint into the other. To choose between endpoints, we merely need to find the smaller of $Q(\mathbf{v}_1)$ and $Q(\mathbf{v}_2)$. Under this policy, any approximation which we produce will use a subset of the original vertices in their original positions.

We can often produce better approximations using *optimal placement*, which we have previously [6] recommended. For a given quadric Q, we can try to find the point $\bar{\mathbf{v}}$ such that $Q(\bar{\mathbf{v}})$ is minimal. Since $Q(\bar{\mathbf{v}})$ is quadratic, finding its minimum is a linear problem; the minimum occurs where $\partial Q/\partial x = \partial Q/\partial y = \partial Q/\partial z = 0$. By solving this system of equations, we find that the optimal position and its error are given by:

$$\bar{\mathbf{v}} = -\mathbf{A}^{-1}\mathbf{b} \qquad \text{and} \qquad Q(\bar{\mathbf{v}}) = -\mathbf{b}^\mathsf{T}\mathbf{A}^{-1}\mathbf{b} + c$$

Geometrically, the minimum $\bar{\mathbf{v}}$ will lie at the center of the concentric ellipsoidal isosurfaces of Q. Of course, the matrix $\mathbf{A}$ may not be invertible. In other words, there may be infinitely many minimal points. This is exactly the case where the level surfaces of Q are degenerate ellipsoids. In such circumstances, we can use subset placement as a fall-back strategy. Indeed, we can also consider other intermediate policies such as selecting between the endpoints and the midpoint, or we might choose to find the optimal position along the edge.

The choice of optimal vs. subset placement depends on the intended application. Optimal placement will tend to produce approximations which fit the original more closely. The resulting meshes also tend to be better shaped — triangles are more equilateral and their areas are more uniform. This is the best choice for generating fixed approximations of an original. However, if the goal is to produce some sort of adaptive representation [8, 9, 13, 19], subset placement may be preferable. The overall fit of the models will be inferior, but we can save significantly on storage. With optimal placement, we must store delta records with each contraction to encode the new vertex position. Using subset placement, we can eliminate such overhead entirely. Since this overhead grows linearly with the number of attributes, the space savings of subset placement can become substantial.

3.4 Homogeneous Variant

The original description of the quadric error metric [6] used an alternate notation. We can treat the quadric Q as a homogeneous matrix where

$$\mathbf{Q} = \begin{bmatrix} \boxed{\mathbf{A}} & \boxed{\mathbf{b}} \\ \boxed{\mathbf{b}^\mathsf{T}} & \boxed{c} \end{bmatrix}$$

This gives us the homogeneous quadratic form $Q(\mathbf{v}) = \mathbf{h}^\mathsf{T}\mathbf{Q}\mathbf{h}$ where $\mathbf{h}$ is the homogeneous vector $[\mathbf{v}\ 1]^\mathsf{T}$. We have found this less convenient than the form presented earlier because it requires us to move back and forth between regular and homogeneous coordinates. It is also slightly less efficient because all our matrix operations must be on 4×4 rather than 3×3 matrices; this is a noticeable difference, for example, when performing matrix inversions.

4 PRESERVING DISCONTINUITIES

Discontinuities of a model, such as creases, open boundaries, and borders between differently colored regions, are often among its most visually significant features. Therefore, their preservation is critical for producing quality approximations. The fundamental algorithm that we have just outlined can already handle shape discontinuities (e.g., creases), and we can easily extend it to handle boundary curves as well.

Surface shape discontinuities (where there is only C^0 continuity) are implicitly preserved by the error metric as described. For example, consider the sharp edges of a cube. A point on the edge of a cube will have contributing planes from both adjoining faces of the cube. Since these planes are perpendicular, the cost of moving the point along the edge is much lower than moving it away from the edge. Consequently, the algorithm will be strongly biased against altering the shape of these edges.

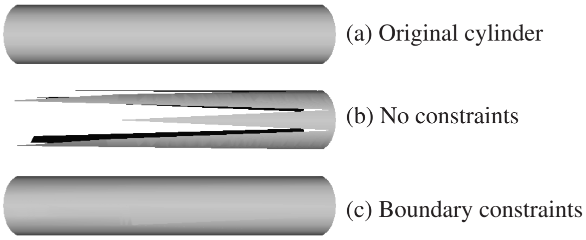


Figure 3: (a) Open-ended cylinder with 7,960 faces. (b) With unconstrained boundaries, this 2,460 face approximation quickly degenerates. (c) Using boundary constraints, the shape is preserved (also 2,460 faces).

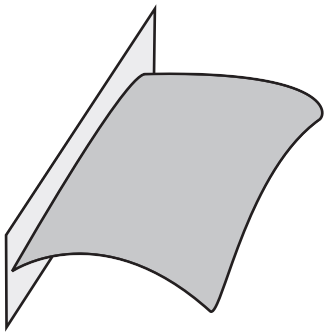


Figure 4: Sample boundary constraint plane. Every edge along the boundary defines a single constraint plane.

In contrast, the basic algorithm ignores boundary curves. Figure 3a shows a cylinder model with open boundaries at each end. Without any modification, the basic simplification algorithm will produce approximations such as shown in Figure 3b. This is clearly unacceptable.

Fortunately, we can easily incorporate boundary constraints into the existing framework. During initialization, we flag all boundary edges. For each face adjacent to a given boundary edge, we compute a plane through the edge that is perpendicular to the face. The perpendicular plane defines a boundary *constraint plane* (see Figure 4). We can form a quadric for this plane, just as with a regular face plane. We weight the resulting quadric by a large penalty factor, and add it into the initial quadric for each of the endpoints. Figure 3c shows the resulting approximation when boundary constraints are enabled. It has the same number of faces as Figure 3b; however, it is a far superior approximation because the boundaries have been properly preserved.

Boundaries may also occur in discrete surface attributes. Consider a surface similar to the one pictured in Figure 7 where each face would be assigned a color from a small discrete palette. Or perhaps we have a map 4-colored by country. Each edge dividing two faces of different colors can be marked as a boundary. This would cause our algorithm to try to faithfully preserve the borders between separate regions.

Naturally, there are limits to this approach. If, for instance, every face is assigned a slightly different color, or if we have a triangulated regular grid where every other triangle is a hole, the results will not be good. The implicit assumption is that boundaries are reasonably sparse.

5 SURFACE PROPERTIES

Many models have surface properties beyond simple geometry. In computer graphics, the most common are surface normals, colors, and textures. To produce approximations which faithfully represent the original, we must maintain these properties as well as the surface geometry. Our basic error metric, based on our earlier work [6] and restated earlier, only considers surface geometry when simplifying the model. This is an obvious shortcoming. However, we can formulate a natural extension of the basic metric which will incorporate surface properties defined as vertex attributes.

We will assume that each vertex, in addition to its position in space, has some associated values which describe other properties. As with geometric position, these values will be linearly interpolated over the faces of the model. Consequently, these properties must be continuous; they cannot be restricted to a discrete set of values. Furthermore, we will assume that distance between two property values is measured with the usual Euclidean metric.

As an ongoing example, we will consider a Gouraud-shaded model for which each vertex has an associated color value $[r\ g\ b]^\mathsf{T}$ where $0 \le r, g, b \le 1$ (see Figure 9).

We need some way to measure errors when surfaces have properties. One natural approach is to use a segregated error metric which measures errors in each attribute separately. This bears some resemblance to Hoppe's [8] energy function, which has separate terms for spatial and scalar attributes. Using our quadric-based algorithm, we could assign a separate quadric to each attribute. In this case, the storage cost of the quadrics would grow linearly with the number of attributes. For our colored surface example, we could measure error as $Q(\mathbf{v}_{\text{pos}}) + R(\mathbf{v}_{\text{rgb}})$ using separate quadrics Q and R for position ($\mathbf{v}_{\text{pos}} = [x\ y\ z]^\mathsf{T}$) and color ($\mathbf{v}_{\text{rgb}} = [r\ g\ b]^\mathsf{T}$), respectively. This will work acceptably with a subset placement policy, because the points we are choosing from (i.e., the two endpoints) are both known to be valid points on the surface. However, this approach will not work well at all with optimal placement because this simple error formula does not account for the cross correlation between position and color.

When using optimal placement, we allow vertex positions to move freely so as to achieve better surface approximations. The simplest scheme for maintaining attribute values would be to simply copy the attributes of $\mathbf{v}_1$ to $\bar{\mathbf{v}}$, but this fails quite noticeably (see Figure 10b). On the other hand, if the optimal position was constrained to lie along the edge, we could simply interpolate attributes along the edge. However, the optimal positions will, in general, lie near the original surface, but not on it. Consequently, we cannot simply interpolate the property values of the endpoints; we need some way to synthesize entirely new values based on the new position. To do this, we use higher dimensional quadrics whose extra coefficients will implicitly encode the cross correlation between the various properties.

We will treat each vertex as a vector $\mathbf{v} \in \mathbf{R}^n$. The first 3 components of $\mathbf{v}$ will be spatial coordinates, and the remaining components will be property values. In the case of our colored model, $n = 6$, and we would use 6-dimensional vectors of the form $[x\ y\ z\ r\ g\ b]^\mathsf{T}$. For consistent results, the model can be scaled so that it lies within the unit cube in $\mathbf{R}^n$. This ensures that the various properties (including position) have the same scale.

5.1 Generalized Error Metric

Having placed our vertices in n-dimensional space, we can formulate an extended version of our quadric error metric. Consider the triangle $T = (\mathbf{p}, \mathbf{q}, \mathbf{r})$. For our example of a colored surface, we would have $\mathbf{p} = [p_x\ p_y\ p_z\ p_r\ p_g\ p_b]^\mathsf{T}$ and so forth. Since we have assumed that all properties are linearly interpolated over triangles, these three n-dimensional points determine a 2-dimensional plane in $\mathbf{R}^n$. Given this, we can construct a quadric that will measure the squared distance of any point in $\mathbf{R}^n$ to this plane.

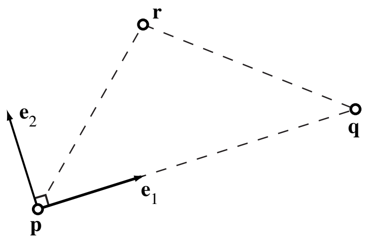


Figure 5: Orthonormal vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ define a local frame, with origin $\mathbf{p}$, for the 2-plane defined by the triangle $(\mathbf{p}, \mathbf{q}, \mathbf{r})$.

To derive the n-dimensional quadric for the 2-plane containing T, we begin by computing two orthonormal unit vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ which lie in the plane. Figure 5 shows a pictorial representation of these vectors. They are defined by the equations:

$$\mathbf{e}_1 = \frac{\mathbf{q} - \mathbf{p}}{\|\mathbf{q} - \mathbf{p}\|}$$

$$\mathbf{e}_2 = \frac{\mathbf{r} - \mathbf{p} - (\mathbf{e}_1 \cdot (\mathbf{r} - \mathbf{p}))\mathbf{e}_1}{\|\mathbf{r} - \mathbf{p} - (\mathbf{e}_1 \cdot (\mathbf{r} - \mathbf{p}))\mathbf{e}_1\|}.$$

This gives us two unit-length vectors which form two axes of a local frame with $\mathbf{p}$ as the origin. In principle, we could compute an entire local frame with axes $\mathbf{e}_1, \ldots, \mathbf{e}_n$. However, in order to compute distances to the 2-plane of T, we will only need to explicitly represent the axes in the plane; for the rest, it is enough to know that they exist and that they are all perpendicular to the plane in question.

Now, consider an arbitrary point $\mathbf{v} \in \mathbf{R}^n$. We are interested in the squared distance D^2 of $\mathbf{v}$ from the plane of T. Let $\mathbf{u} = \mathbf{p} - \mathbf{v}$; the squared length of $\mathbf{u}$ can be decomposed as

$$\|\mathbf{u}\|^2 = (\mathbf{u}\cdot\mathbf{e}_1)^2 + \cdots + (\mathbf{u}\cdot\mathbf{e}_n)^2,$$

which we can rewrite as

$$(\mathbf{u}\cdot\mathbf{e}_3)^2 + \cdots + (\mathbf{u}\cdot\mathbf{e}_n)^2 = \|\mathbf{u}\|^2 - (\mathbf{u}\cdot\mathbf{e}_1)^2 - (\mathbf{u}\cdot\mathbf{e}_2)^2.$$

Note that the left hand side is the squared length of $\mathbf{u}$ along all components perpendicular to the plane of T, in other words, the squared perpendicular distance of $\mathbf{v}$ to the plane of T. This is precisely the distance we are interested in:

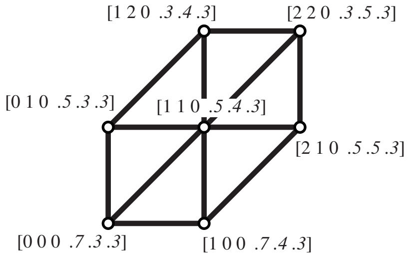
$$\begin{aligned} D^2 &= \|\mathbf{u}\|^2 - (\mathbf{u}^\mathsf{T}\mathbf{e}_1)^2 - (\mathbf{u}^\mathsf{T}\mathbf{e}_2)^2 \\ &= \mathbf{u}^\mathsf{T}\mathbf{u} - (\mathbf{u}^\mathsf{T}\mathbf{e}_1)(\mathbf{e}_1{}^\mathsf{T}\mathbf{u}) - (\mathbf{u}^\mathsf{T}\mathbf{e}_2)(\mathbf{e}_2{}^\mathsf{T}\mathbf{u}). \end{aligned}$$

By expanding and collecting terms, we arrive at the following formula:

$$\begin{aligned} D^2 = \ & \mathbf{v}^\mathsf{T}\mathbf{v} - 2\mathbf{p}^\mathsf{T}\mathbf{v} + \mathbf{p}\cdot\mathbf{p} \\ & -\mathbf{v}^\mathsf{T}(\mathbf{e}_1\mathbf{e}_1{}^\mathsf{T})\mathbf{v} + 2(\mathbf{p}\cdot\mathbf{e}_1)\mathbf{e}_1{}^\mathsf{T}\mathbf{v} - (\mathbf{p}\cdot\mathbf{e}_1)^2 \\ & -\mathbf{v}^\mathsf{T}(\mathbf{e}_2\mathbf{e}_2{}^\mathsf{T})\mathbf{v} + 2(\mathbf{p}\cdot\mathbf{e}_2)\mathbf{e}_2{}^\mathsf{T}\mathbf{v} - (\mathbf{p}\cdot\mathbf{e}_2)^2, \end{aligned}$$

which has the structure of our quadric metric. We can rewrite it as $D^2 = \mathbf{v}^\mathsf{T}\mathbf{A}\mathbf{v} + 2\mathbf{b}^\mathsf{T}\mathbf{v} + c$ where:

$$\begin{aligned} \mathbf{A} &= \mathbf{I} - \mathbf{e}_1\mathbf{e}_1{}^\mathsf{T} - \mathbf{e}_2\mathbf{e}_2{}^\mathsf{T} \\ \mathbf{b} &= (\mathbf{p}\cdot\mathbf{e}_1)\mathbf{e}_1 + (\mathbf{p}\cdot\mathbf{e}_2)\mathbf{e}_2 - \mathbf{p} \\ c &= \mathbf{p}\cdot\mathbf{p} - (\mathbf{p}\cdot\mathbf{e}_1)^2 - (\mathbf{p}\cdot\mathbf{e}_2)^2. \end{aligned}$$



$$\mathbf{A} = \begin{bmatrix} 0.06 & 0 & 0 & 0 & -0.59 & 0 \\ 0 & 0.23 & 0 & 1.15 & 0 & 0 \\ 0 & 0 & 6.00 & 0 & 0 & 0 \\ 0 & 1.15 & 0 & 5.77 & 0 & 0 \\ -0.59 & 0 & 0 & 0 & 5.94 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.00 \end{bmatrix}$$

Figure 6: At top is a simple mesh, of 6 triangles, with *xyzrgb* values at each vertex (*rgb* values in italic type). Summing quadrics for each face yields the matrix **A** shown.

In this generalized quadric, **A** is an $n \times n$ matrix, and **b** is an n-vector.

In Figure 6, we can see a simple example of how *xyzrgb* quadrics work. On top is the 6 triangle neighborhood of the central vertex. Each vertex is labeled with its *xyzrgb* coordinates. Note that r coordinate is highly correlated with y and that g is highly correlated with x. Below the mesh, we see the matrix **A** for the initial quadric associated with the central vertex. It is computed by summing the quadrics for the 6 surrounding faces. By examining the non-zero off-diagonal elements, we can see how the quadric records the high correlation between r & y and between g & x.

Because our generalized quadric error metric has exactly the same structure as before, we can substitute it into our fundamental algorithm very easily. We have changed the way quadrics are constructed from initial triangles, but we use them in an identical manner. In particular, we can use the same simple formulas for finding the optimal target of a contraction.

6 DISCUSSION

This generalized metric allows us to continue to use our optimal placement policy. It provides a means to synthesize new property values at vertex positions which do not lie anywhere on the original surface. Since, in most cases, our algorithm produces the best quality approximations using optimal placement, this is a significant advantage.

Unfortunately, for every property, we must add extra dimensions to the vertices and quadrics. Consequently, as we add more and more properties to a surface, the size of the quadric matrix grows quadratically in the size of the attributes, and the running time will also increase. We can summarize the space requirements for a few common cases as follows:

model type	vertex	**A**	# unique coefficients
geometry only	$[x\ y\ z]^\mathsf{T}$	3×3	$\binom{5}{2}=10$
geometry + 2D texture	$[x\ y\ z\ s\ t]^\mathsf{T}$	5×5	$\binom{7}{2}=21$
geom + color (Gouraud)	$[x\ y\ z\ r\ g\ b]^\mathsf{T}$	6×6	$\binom{8}{2}=28$
geometry + normals	$[x\ y\ z\ a\ b\ c]^\mathsf{T}$	6×6	$\binom{8}{2}=28$

However, this overhead is not extreme. For instance, we have run tests on models with both color and surface normals at every vertex; this requires 9-dimensional quadrics. A 30,000 face model can still be simplified in under 30 seconds on a PentiumPro 200 system.

We must also be able to deal with constraints on the range of property values. RGB colors, for instance, must be kept within the color cube. While our algorithm will not generate colors far outside the cube, since they would very poorly fit the data, it may generate colors that are slightly below 0 or above 1. In such cases, we merely clamp the offending colors to the nearest point on the color cube. Since only small distances are involved, this should not introduce any appreciable artifacts. The same caution must be applied to texture coordinates. For surface normals, we need to rescale the resulting vectors to have unit length.

In our presentation of the generalized error metric, we have weighted all vertex attributes equally. However, for optimal results one might wish to selectively weight certain attributes more than others. It seems doubtful that there is a single optimal weighting for a given set of attributes, but it can easily be offered as a user-selectable preference.

A prime application of our algorithm is the simplification of Gouraud-shaded meshes produced by radiosity simulations. Even the simplest geometries are often carved up into very small polygons to achieve high-quality shading. The resulting surfaces are ripe for simplification; typically, the surface geometry is heavily over-sampled and the colors vary smoothly over large areas.

6.1 Attribute Continuity

We have assumed that property values vary continuously over the surface, but in practice, this is not always the case. For example, consider simply wrapping a texture around a cylinder. There will be a seam where s wraps from 0 to 1. Consequently, every vertex along this seam must have two separate texture coordinates, and this will introduce a discontinuity at every one of these vertices.

To represent and simplify this textured cylinder with our current algorithm, we would need to replicate each vertex along the seam; each copy would have separate texture coordinates but identical positions. In other words, we must force the surface to be a topological plane rather than a cylinder. Using boundary constraints to maintain the seam, this might produce acceptable results.

However, this will not work for all cases. A model might conceivably have a distinct piece of texture for each face, and this would create a discontinuity at every edge. As mentioned in Section 4, the algorithm does not work well when constraints proliferate to this extent. A more complete solution would require that we allow multiple values for each property at each vertex. To apply our simplification algorithm, we would need to create multiple corresponding quadrics for each vertex. While we have not yet implemented this approach, we believe it would work well.

6.2 Euclidean Attributes

We have also assumed that the metric for measuring the difference in attributes is Euclidean. More specifically, we have assumed that attribute values at the vertices will be linearly interpolated over the faces of the model.

Perceptual color spaces are not Euclidean in RGB. However, we expect that the approximations produced by our algorithm will be displayed as Gouraud-shaded surfaces. Consequently, for display, colors will be linearly interpolated over faces. Thus, regardless of the non-Euclidean nature of RGB color space, our assumptions coincide with the way the results will be displayed.

Surface normals are another attribute type which might not seem to conform with our Euclidean assumptions. However, we contend that our algorithm will typically treat normals appropriately. Consider the Gauss mapping of the surface, where every vertex is mapped to the point on the unit sphere corresponding to the unit surface normal at that vertex. Simplifying surface normals as part of our generalized quadric metric is essentially equivalent to simplifying this spherical surface with the basic algorithm. So, as long as normals don't change too rapidly, our extended algorithm will produce good surface normals.

7 RESULTS

Figures 8a–e illustrate the performance of the fundamental algorithm on a very complex surface which is purely geometric. The original model contains 1,085,634 faces; the approximations shown contain 20,000 and 1,000 faces. Producing these approximations might require hours with some algorithms. However, on a PentiumPro 200 system, the running time of our algorithm was 46 seconds for initialization (constructing initial quadrics and ordering edges in a heap)[2] and 130 seconds for the actual simplification. During execution, the total memory used was roughly 200MB, of which 40MB were required to store the quadrics at each vertex (using 10 `doubles` each). The initial surface mesh is really very dense; the approximation shown in Figure 8b uses only about 2% of the original number of triangles yet preserves almost all of the surface features. In Figure 8d we see a very simple 1,000 face (0.01%) approximation. Most of the detail of the surface, such as the necklace and facial features, have been removed. However, the basic shape of the object is still faithfully preserved and, at very low resolutions, would be a suitable replacement.

7.1 Color

Figure 7 is a simple demonstration of our algorithm using 6-D *xyzrgb* quadrics. The original surface on the left is a piece of a sphere Gouraud-shaded with a swirl pattern; it also has an open boundary. The approximation is about 5% the size of the original. Notice that the structure of the mesh conforms closely to the color pattern. The mesh is fairly sparse in areas of constant color. Near the borders of the color pattern, not only is the triangle density higher, but the edges are properly aligned with the curve of the swirl pattern.

A more challenging Gouraud-shaded example is shown in Figure 9. The surface itself is a sphere built from points on a latitude-longitude grid. Each vertex is assigned a color based on the elevation of the Earth's surface at that point. Since it is so simple, the surface shape is preserved very accurately. More importantly, the coloring of the surface is also preserved well. Note that larger triangles appear in areas of constant or linear color variation, such as oceans, while smaller triangles occur along the coastlines.

[2] We have excluded the time required to read and write files to disk since these vary significantly based on the format used.

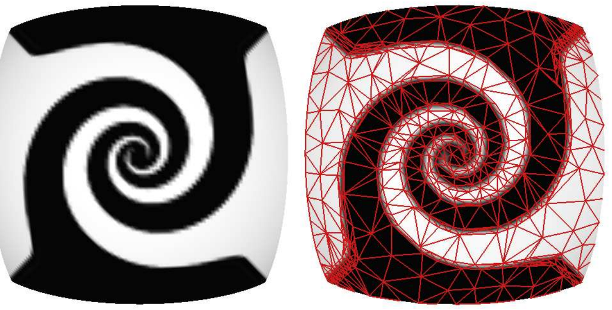


Figure 7: At left: a curved surface (18,050 faces) with colors at each vertex. At right: 1,000 face approximation. Notice that mesh edges follow the color contours.

7.2 Texture

Our extended algorithm can just as easily be used to simplify surfaces with texture maps. In Figure 10a, we are looking down at a square height field of the eastern half of North America. The surface is textured with a satellite photograph with height given by altitude and bathymetric data.

Figure 10b shows the result of simplifying this surface using optimal placement without regard for the texture coordinates. For each contraction, we simply propagate the texture coordinates of $\mathbf{v}_1$. As we would expect, this produces very poor results. Moving vertices without synthesizing correct texture coordinates causes the texture to warp like a rubber sheet. For example, note the substantial distortion around Florida and New England.

In contrast, the approximation shown in Figure 10c, with the same number of faces, was produced using an *xyzst* extended quadric. The algorithm produces nearly the same geometric surface as before. However, by using the extended metric, we have allowed the algorithm to synthesize appropriate texture coordinates for the new vertices. Unlike colored surfaces (such as Figure 7), the edges of the mesh do not align with features in the texture. The simplification algorithm never accesses the pixels of the texture; it merely tries to update texture coordinates so that the texture is mapped onto the surface in the same way as it was originally.

In the example of Figure 10, there is a direct correspondence between (x, y) and (s, t). This would provide an alternate way to synthesize new texture coordinates: for any vertex position, we can use x and y to compute appropriate s and t values. However, our extended algorithm provides a much more general solution.

8 SUMMARY AND FUTURE WORK

Our algorithm can quickly produce good quality approximations of polygonal surface models. It is one of the fastest algorithms available for producing quality approximations. The fundamental algorithm works well for purely geometric surfaces [6]. In this paper, we have explored our original algorithm further. We have provided more detailed analysis of the nature of quadrics, and we have demonstrated that it is capable of simplifying models of reasonably high complexity. Our original algorithm was limited to surfaces without surface properties such as color and texture. In this paper, we have presented a new generalized error metric, which substantially expands the set of models which can be simplified using our algorithm. As we have shown, it is capable of rapidly producing quality approximations that preserve both surface shape and associated surface properties.

There are certainly further improvements that could be made to the algorithm we have outlined here. A more complete system would also include support for multiple attribute values per vertex. This would, for instance, help solve the problems posed by texture seams and by surface normal discontinuities. We also believe that the memory usage of our implementation could be reduced.

Related information and a prototype implementation of our basic algorithm can be found at http://www.cs.cmu.edu/~garland/quadrics/.

9 ACKNOWLEDGEMENTS

We would like to thank Konrad Polthier for the interesting discussion of relating the quadric error metric and differential geometry. Thanks also to the Stanford Graphics Lab for making the models of the Buddha statue and the bunny available[3]. The model of the Earth is based on an image of the ETOPO5 Earth elevation dataset provided by the USGS[4]. This research was supported under NSF grants CCR-9357763 & CCR-9619853 and by the Schlumberger Foundation.

References

[1] Andrew Certain, Jovan Popović, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive multiresolution surface viewing. In *SIGGRAPH 96 Conference Proceedings*, pages 91–98, August 1996.

[2] Jonathan Cohen, Dinesh Manocha, and Marc Olano. Simplifying polygonal models using successive mappings. In *Proceedings IEEE Visualization '97*, pages 395–402, October 1997.

[3] Herbert Edelsbrunner and Dmitry V. Nekhayev. Topology and curvature preserving surface decimation. Technical report, Raindrop Geomagic, Inc., Champaign, IL, 1997. rgi-tech-97-010.

[4] Michael Garland. PhD thesis, Carnegie Mellon University, CS Department, 1998. To appear.

[5] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical report, CS Dept., Carnegie Mellon U., Sept. 1995. CMU-CS-95-181, http://www.cs.cmu.edu/˜garland/scape.

[6] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, August 1997. http://www.cs.cmu.edu/˜garland/quadrics/.

[7] André Guéziec. Surface simplification inside a tolerance volume. Technical report, Yorktown Heights, NY 10598, Mar. 1996. IBM Research Report RC 20440, http://www.watson.ibm.com:8080/search_paper.shtml.

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

[9] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Proc.*, pages 189–198, August 1997. http://research.microsoft.com/˜hoppe/.

[10] Merlin Hughes, Anselmo A. Lastra, and Edward Saxe. Simplification of global-illumination meshes. *Computer Graphics Forum*, 15(3):339–345, August 1996. Proc. Eurographics '96.

[11] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[12] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison Wesley, Reading, MA, Third edition, 1997.

[13] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Proc.*, pages 199–208, August 1997.

[14] Barrett O'Neill. *Elementary Differential Geometry*. Academic Press, Boston, 1966.

[15] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Proc.*, pages 217–224, 1997. http://research.microsoft.com/˜hoppe/.

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

[17] 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.

[18] 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.

[19] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of Visualization '96*, pages 327–334, October 1996.

[3] http://www-graphics.stanford.edu/data/3Dscanrep/

[4] http://geochange.er.usgs.gov/pub/sea_level/Contents/05mi_pres.html

Figure 8: Simplifying geometry only: A very complex model of 1,085,634 faces (a) is simplified to 20,000 faces (b–c) and 1,000 faces (d–e).

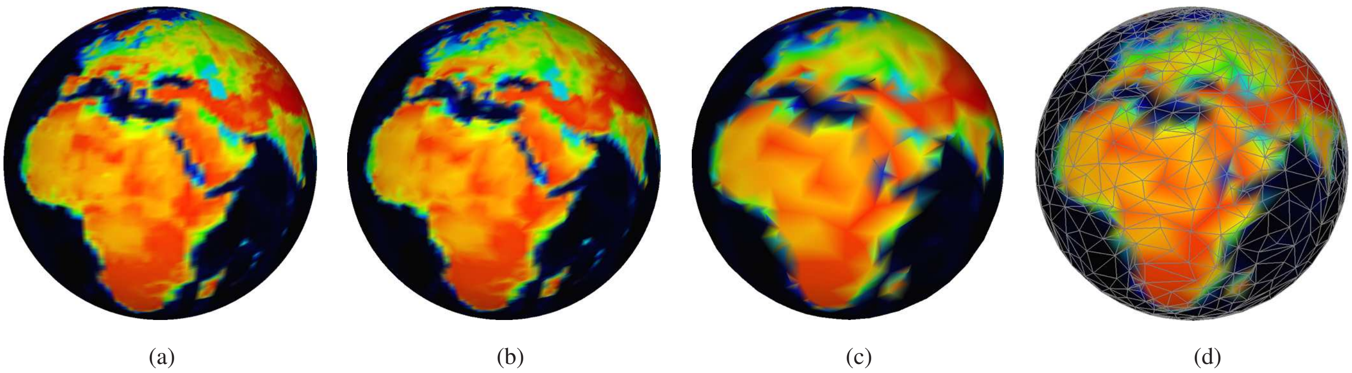


Figure 9: Simplifying geometry & color: A Gouraud-shaded surface of 73,728 faces (a) is reduced to 20,000 faces (b) and 3,000 faces (c–d).

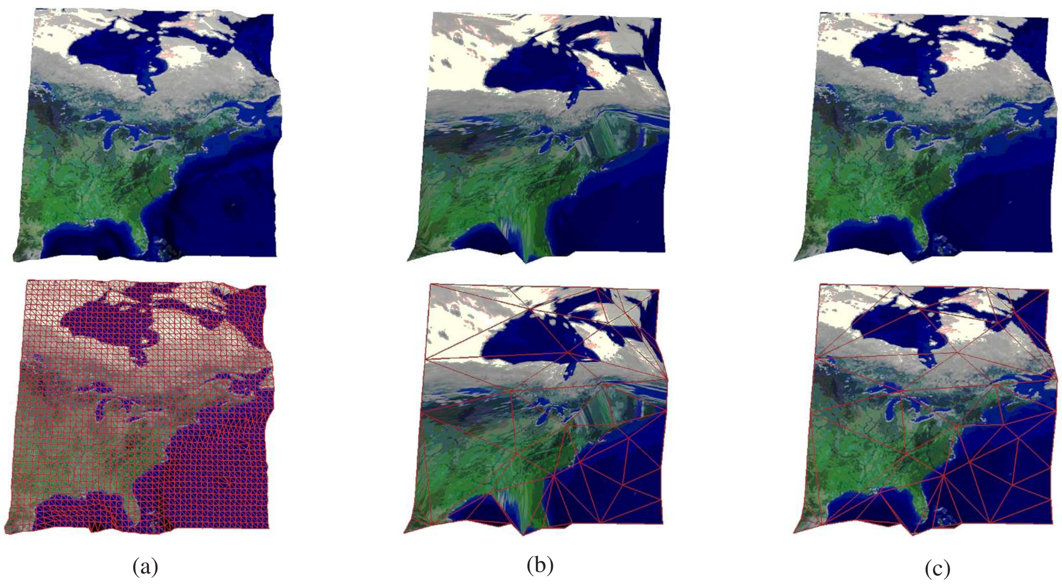


Figure 10: Geometry & texture: A 3,872 face model (a) reduced to 53 faces without (b) and with (c) updating texture coordinates.