Polygonal Mesh Simplification with Face Color and Boundary Edge Preservation Using Quadric Error Metric

Chin-Shyurng Fahn

Department of Computer Science and Information Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan 106, Republic of China
csfahn@mail.csie.ntust.edu.tw

Hung-Kuang Chen

Department of Electronic Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan 106, Republic of China

Yi-Haur Shiau

Software and Development Division
National Center for High-Performance Computing
Hsinchu, Taiwan 300, Republic of China

Abstract

In the applications such as scientific and medical visualization, highly detailed polygonal meshes are needed. Rendering these polygonal meshes usually exceeds the capabilities of graphics hardware. To improve rendering efficiency and maintain proper interactivity, the polygonal mesh simplification technique is commonly used to reduce the number of polygons of the mesh and to construct the multi-resolution representation. In this paper, we propose a new and simple constraint scheme based on the quadric error metric proposed by Garland and Heckbert [1] to preserve face colors and boundary edges during the simplification process. In addition, our method generates progressive meshes that store the polygonal mesh in a continuous multi-resolution representation. According to our experimental results, this new method is successful in preserving face colors and boundary edges. Moreover, we compare the latency of resolution changes for the progressive meshes of various models.

1. Introduction

Complex polygonal mesh models with a huge amount of data are often generated from various kinds of applications, for example, scientific and medical visualization systems, mechanical CAD systems, and automatic modeling systems. Rendering such complex meshes at an interactive rate usually exceeds the hardware capabilities. Hence, to maintain the interactivity, a polygonal mesh simplification algorithm is required to effectively lower model complexity and preserve visual appearance.

In the polygonal mesh simplification problem, we are given an over-sampling polygonal mesh $M(K, V)$, where K represents the topology of the mesh and V represents the geometry. The purpose of simplification is to find a compact representation $M'(K', V')$, which can be cast into two optimization frameworks. First, we can set up a maximum tolerable deviation between V and V', and minimize $|K|$. Second, given a minimum $|K|$, the problem becomes finding a compact representation that minimizes the deviation between V and V'.

Of the two frameworks, the former is useful in CAD and scientific applications where the control of model quality is more significant than that of rendering efficiency. The simplification algorithms based on this framework usually have to bound the maximum error. The latter is applied mostly in real-time rendering applications where the polygon budget for each frame is known in advance.

There are numerous ways to find M', for instance, the filtering method such as wavelets can be employed to simplify the polygonal mesh by separating the high frequency part from the low frequency part, or we can apply some simplification operations combined with an error approximation scheme repeatedly to simplify the polygonal mesh. To properly simplify the model, it is often desirable to have an error control scheme to guide the simplification and accurately approximate the deviation. In most previous works, the simplification incorporated with the error control scheme is only applied to the surface geometry. However, the visual appearance of an object relies not only on the shape of the object but also on the other features such as colors, textures, materials, and bumps. Thus, to attain the best mesh quality after simplification, an error control scheme that can correctly bound the deviation of the surface geometry as well as preserve the surface properties is very important.

In this paper, we will propose a simple, fast, and memory efficient approach to approximating the surface geometry based on the error approximation scheme called *quadric error metric* proposed by Garland and Heckbert [1], [2], which preserves face colors and boundary edges using a new and simple constraint scheme. Compared with the aforementioned approach, our scheme is simpler and needs less memory. According to our experimental results, the new method preserves face colors and boundary edges well under a very good control. Moreover, it can generate progressive meshes that store the polygonal mesh in a continuous multi-resolution representation. Such progressive meshes can perform rapid conversions between different resolutions.

2. Background and related works

The issue of polygonal mesh simplification has been studied for years. We will briefly review some of the works here by roughly categorizing the mesh simplification algorithms as follows. For a more complete review, we suggest readers surveying the literatures [3]-[5]. The progressive mesh representation is also introduced later.

2.1. Mesh simplification algorithms

Wavelet-based methods: Lounsbery [6] extended the subdivision surface techniques to multi-resolution analysis for arbitrary topological types. This scheme is only applicable to a restricted class of meshes with subdivision connectivity. Eck et al. [7] devised a method to overcome the subdivision connectivity restriction, which is capable of converting an arbitrary mesh to a multi-resolution representation. A recent approach improved this one to manage both surface geometry and colors [8].

Vertex clustering methods: Vertex clustering algorithms [9], [10] approximate the original surface based on a clustering scheme that partitions the vertex set into some clusters in accordance with geometric proximity. Vertices within the same cluster are grouped into a single vertex. The clustering grids can be either uniform [9] or non-uniform [10]. These algorithms are simple and fast. Moreover, they can accept non-manifold polygonal meshes and are profitable for quick construction of vertex hierarchies of the simplified models. By recursively merging clusters, a sequence of fine to coarse meshes can be organized in a tree-like manner. This allows selective refinement for view-dependent simplification [11]. An alternative solution proposed by Lindstrom [12] improves the vertex-grading phase by recording geometric information in the non-empty cells during a single traversal of the input mesh. The method avoids the need of storing the vertex grades and maintaining the connectivity used for computing them.

Vertex decimation methods: Based on multiple filtering passes, this class of algorithms is first proposed by Schroeder et al. [13]. It locally analyzes the geometry and topology of the mesh and removes vertices according to the minimal distance or curvature angle criterion. Some other works evaluate the global error caused by each vertex decimation and re-triangulation [14]-[18].

Edge collapse methods: Hoppe et al. [19] first cast the mesh simplification problem as an optimization one. The three simplification operations are edge collapse, vertex split, and edge swap. These operations are iteratively employed to minimize an energy function that directly measures the deviation of the target mesh from the original. The result usually has good quality; however, since the optimization is only applied to the surface geometry, other attributes such as colors, normals, and texture coordinates are not considered.

In Hoppe's later work [20], he proposed the *progressive mesh* (PM). This paper presented a method that simplifies a polygonal mesh through a series of edge collapse operations incorporated with an energy minimization scheme. The resulting mesh is a continuous multi-resolution representation composed of a coarse approximation together with the records of vertex split operations. This idea is simple and successful. It preserves not only the geometry but also scalar attributes and discontinuities. With this data structure, some attractive features, for example, geo-morph and selective refinement, are also provided. Therefore, it is adopted by most of the following works.

In the light of our knowledge, the progressive mesh has been implemented by Microsoft in their DirectX SDX, which is widely used by game and entertainment industry as their application platform. In this paper, we also have our method generate the progressive mesh. Readers interested in implementing the progressive mesh can refer to the literature [21].

2.2. Progressive mesh

A PM representation describes an arbitrary triangle mesh $\hat{\mathbf{M}}$ stored as a coarse mesh $\mathbf{M}^0$ associated with a sequence of n vertex split (*vsplit*) transformations that indicate how to incrementally refine $\mathbf{M}^0$ back to the original mesh $\hat{\mathbf{M}}$. In the refinement process, each *vsplit* transform adds an extra vertex to the mesh. Consequently, the PM representation of $\hat{\mathbf{M}}$ can define a sequence of meshes $\mathbf{M}^0$, $\mathbf{M}^1$, …, and $\mathbf{M}^n$ ($\hat{\mathbf{M}}$) of increasing resolution, and after each *vsplit* transformation, a new approximation of the original mesh $\hat{\mathbf{M}}$ is then created.

A PM representation for $\hat{\mathbf{M}}$ is obtained from a carefully simplifying procedure through n successive edge collapse (*ecol*) transformations. In the process of an edge collapse transformation *ecol*($\mathbf{v}_s$, $\mathbf{v}_t$), two vertices of the edge, $\mathbf{v}_s$ and $\mathbf{v}_t$, are merged to a new single vertex $\mathbf{v}_s$. The vertex $\mathbf{v}_t$ and its two adjacent faces ($\mathbf{v}_s$, $\mathbf{v}_t$, $\mathbf{v}_l$) and ($\mathbf{v}_t$, $\mathbf{v}_s$, $\mathbf{v}_r$) are removed. A *vsplit* transformation is the inverse operation of the *ecol* transformation, which adds a new vertex $\mathbf{v}_t$ near vertex $\mathbf{v}_s$ and generates two new faces ($\mathbf{v}_s$, $\mathbf{v}_t$, $\mathbf{v}_l$) and ($\mathbf{v}_t$, $\mathbf{v}_s$, $\mathbf{v}_r$).

Figure 1 illustrates the *vsplit* and *ecol* transformations for both boundary and non-boundary edges. As shown in Fig. 1(a), if an *ecol* operation is applied to a non-boundary edge, two faces are removed from the original mesh; on the other hand, if the edge is a boundary edge, only one face is removed as Fig. 1(b) shows. In the reverse process, if a *vsplit* operation is applied to a non-boundary vertex, two new faces are brought into the mesh; otherwise, only one face is added.

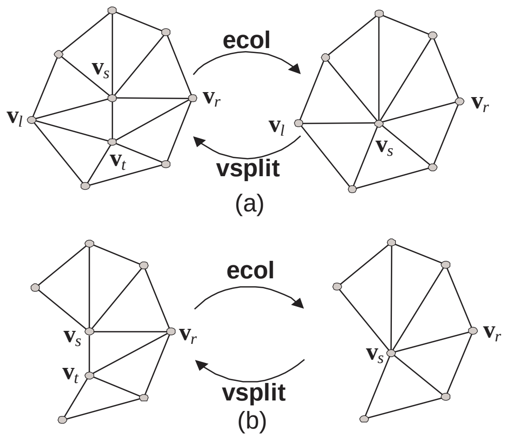


Figure 1 Illustration of vertex split and edge collapse transformations.

Therefore, an original mesh $\hat{\mathbf{M}} = \mathbf{M}^n$ can be iteratively simplified into a coarser mesh $\mathbf{M}^0$ by a sequence of n *ecol* records: $\hat{\mathbf{M}} = \mathbf{M}^n \xrightarrow{ecol_{n-1}} \cdots \xrightarrow{ecol_0} \mathbf{M}^0$. Because edge collapse transformations are invertible, we can represent an arbitrary triangle mesh $\hat{\mathbf{M}}$ as a simple mesh M^0 together with a sequence of n *vsplit* records: $\mathbf{M}^0 \xrightarrow{vsplit_0} \cdots \xrightarrow{vsplit_{n-1}} \mathbf{M}^n = \hat{\mathbf{M}}$. Accordingly, the PM representation of $\hat{\mathbf{M}}$ can be expressed by ($\mathbf{M}^0$, {$vsplit_0$, …, $vsplit_{n\text{-}1}$}). Totally n different approximating meshes $(\mathbf{M}^i \,|\, i = 0, \ldots, n\text{-}1)$ with increasing accuracy are implicitly defined in a PM, where $\mathbf{M}^i$ is accomplished by applying the operations $vsplit_0$, $vsplit_1$, …, and $vsplit_{i\text{-}1}$ to $\mathbf{M}^0$.

3. Review of quadric error metric

The *quadric error metric* initially proposed by Garland and Heckbert [1] is intrinsically a modified form of the plane-based error metric proposed by Ronfard and Rossignac [17], which measures the sum of squared distances from a set of planes to a given vertex. The quadric error metric utilizes matrix techniques to efficiently speedup the computation, and is currently regarded as one of the fastest iterative simplification algorithms. Since our method is based on this error metric, we will describe its mathematic reasoning below.

3.1. Plane-based error metric

Ronfard and Rossignac [17] associated each vertex with a set of planes spanned by its adjacent faces and evaluated the contraction cost concerning the set of planes. The contraction cost for each vertex is defined as the sum of squared distances from the vertex to its associated set of planes. After each contraction, the new vertex is associated with the mergence of the two sets of planes. Because the computation of measuring the distances from the vertex to the planes is lower than that of measuring distances to faces, this metric is much efficient.

Garland and Heckbert developed a more efficient algorithm [1] by iteratively contracting vertex pairs according to the error quadric metric defined on each vertex. Rather than restricting the contraction to the two vertices sharing the same edge, the vertex pair contraction is generalized to arbitrary vertex pairs satisfying a given threshold rule. That is, a vertex pair $(\mathbf{v}_i, \mathbf{v}_j)$, as shown in Fig. 2, is valid for contraction if either $(\mathbf{v}_i, \mathbf{v}_j)$ is an edge or $|\mathbf{v}_i\text{-}\,\mathbf{v}_j|$ is less than a given threshold. The quadric is closely related to the error metric provided by Ronfard and Rossignac [17]. Instead, the contraction cost defined on each vertex is in the quadratic form to replace the sum of squared distances to its associated planes. The algorithm offers a compromise between very fast but low quality and very slow but high quality.

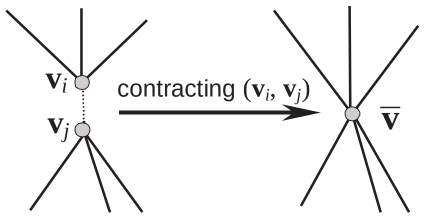


Figure 2 Illustration of a non-edge pair contraction.

In a more recent work [2], Garland and Heckbert generalized their quadric error metric to process vertex attributes. In their work, the vertex attributes are handled by augmenting the dimension of the quadric matrix. If m additional attributes are to be considered, their algorithm requires a total of $(4+m)(5+m)/2$ extra storage for keeping the coefficients. To improve this, Hoppe proposed a new quadric error metric [22] that manipulates vertex attributes and their discontinuities, and incorporated it with memoryless and volume preservation schemes. This method has better quality and saves more memory; however, it still takes a higher dimensional quadric for preserving vertex attributes.

3.2. Quadric error metric

In our work, we adopt the quadric error metric proposed by Garland and Heckbert [1] to evaluate the contraction cost of each edge collapse operation. Before describing the new constraint scheme, we will briefly discuss the original quadric error metric in this subsection.

Each triangle in the original mesh defines a plane. In R^3, the plane equation can be written in a general form as $ax + by + cz + d = 0$. The same equation can be also written as $\mathbf{n}^{\mathrm{T}}\mathbf{v} + d = 0$, where $\mathbf{n} = [a\ b\ c]^{\mathrm{T}}$ is the unit surface normal of the plane and d is a scalar constant. The squared distance from a given vertex $\mathbf{v} = [x\ y\ z]^{\mathrm{T}}$ to the plane is depicted as

$$D^2(\mathbf{v}) = (\mathbf{n}^{\mathrm{T}}\mathbf{v} + d)^2 = (ax + by + cz + d)^2. \quad (1)$$

Therefore, the sum of squared distances from a vertex $\mathbf{v}$ to the set of planes P is expressed by

$$E_P(\mathbf{v}) = \sum_i D^2(\mathbf{v}) = \sum_i (\mathbf{n}^{\mathrm{T}}\mathbf{v} + d)^2. \quad (2)$$

When an edge is contracted, the two endpoints of the edge are merged to a new vertex. Hence, the two sets of planes associated with the old vertices are also merged and associated with the new vertex.

In the work of Ronfard and Rossignac [17], the contraction cost of each vertex is calculated by taking the maximum squared distance over the set of planes. However, Garland and Heckbert considered the contraction cost of each vertex as the sum of squared distances to its associated planes $E_P(\mathbf{v})$ in order to derive a more efficient representation called *quadric* [1]. They rewrote Eq. (1) as

$$\begin{aligned} D^2(\mathbf{v}) &= (\mathbf{n}^{\mathrm{T}}\mathbf{v} + d)^2 = (\mathbf{v}^{\mathrm{T}}\mathbf{n} + d)(\mathbf{n}^{\mathrm{T}}\mathbf{v} + d) \\ &= (\mathbf{v}^{\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{v} + 2d\mathbf{n}^{\mathrm{T}}\mathbf{v} + d^2), \end{aligned} \quad (3)$$

where $\mathbf{n}\mathbf{n}^{\mathrm{T}} = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{bmatrix}$.

For a given plane $\mathbf{n}^{\mathrm{T}}\mathbf{v} + d = 0$, its quadric $\mathbf{Q}$ is defined as

$$\mathbf{Q} = (\mathbf{A}, \mathbf{b}, c) = (\mathbf{n}\mathbf{n}^{\mathrm{T}}, d\mathbf{n}, d^2), \quad (4)$$

where $\mathbf{A}$ is a 3×3 symmetric matrix, $\mathbf{b}$ is a 3×1 vector, and c is a scalar constant. Thus, the squared distance from a vertex $\mathbf{v}$ to the plane can be given by the quadric

$$\mathbf{Q}(\mathbf{v}) = D^2(\mathbf{v}) = \mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathrm{T}}\mathbf{v} + c. \quad (5)$$

The addition of two quadrics can be naturally defined as

$$\mathbf{Q}_i(\mathbf{v}) + \mathbf{Q}_j(\mathbf{v}) = (\mathbf{Q}_i + \mathbf{Q}_j)(\mathbf{v}), \quad (6)$$

where $\mathbf{Q}_i + \mathbf{Q}_j = (\mathbf{A}_i + \mathbf{A}_j, \mathbf{b}_i + \mathbf{b}_j, c_i + c_j)$.

Moreover, Garland and Heckbert defined the initial quadric $\mathbf{Q}$ associated with a vertex $\mathbf{v}$ as the weighted sum of the fundamental quadrics of its adjacent planes, i.e., $\mathbf{Q}(\mathbf{v}) = \sum_i w_i \mathbf{Q}_i(\mathbf{v})$, where w_i is the area of the i-th adjacent face. They also defined the contraction cost of $(\mathbf{v}_i, \mathbf{v}_j) \to \overline{\mathbf{v}}$ as

$$\mathbf{Q}_i(\overline{\mathbf{v}}) + \mathbf{Q}_j(\overline{\mathbf{v}}) = (\mathbf{Q}_i + \mathbf{Q}_j)(\overline{\mathbf{v}}). \quad (7)$$

The placement of the new vertex $\overline{\mathbf{v}}$ is decided by minimizing the quadric for each contraction, where the minimum occurs when

$$\nabla\mathbf{Q}(\overline{\mathbf{v}}) = 2\mathbf{A}\overline{\mathbf{v}} + 2\mathbf{b} = 0. \quad (8)$$

By solving Eq. (8), the optimal position is

$$\overline{\mathbf{v}} = -\mathbf{A}^{-1}\mathbf{b}. \quad (9)$$

In consequence, the quadric for this new vertex is

$$\mathbf{Q}(\overline{\mathbf{v}}) = \mathbf{b}^{\mathrm{T}}\overline{\mathbf{v}} + c = -\mathbf{b}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{b} + c. \quad (10)$$

If $\mathbf{A}$ is singular, the inverse matrix $\mathbf{A}^{-1}$ does not exist. Under this condition, no optimal placement exists. Alternatively, a fallback method is used to select a sub-optimal placement of $\overline{\mathbf{v}}$. In our algorithm, we simply choose the midpoint of $\mathbf{v}_i$ and $\mathbf{v}_j$ as the new position $\overline{\mathbf{v}}$, if $\mathbf{A}$ is singular.

4. Our proposed method

In the previous section, we have briefly reviewed the quadric error metric [1] that was later extended to the *generalized quadric error metric* [2] to preserve scalar vertex attributes. The preservation of vertex attributes such as vertex colors, vertex normals, texture coordinates, and other scalar attributes has been studied in [2], [15], [20]-[26]. In most applications, some features are discrete and usually associated with faces rather than vertices, for example, face colors, boundary edges, and sharp features.

4.1. Algorithm outline

Our mesh simplification method is based on the quadric error metric proposed by Garland and Heckbert [1]. In the main, the algorithm consists of three stages achieving pre-setup, pre-computation, and contraction successively.

Step 1 to Step 3 constitutes the first stage; in which, we only consider edged vertex pairs as contraction candidates. From the candidates, feature edges and boundary edges are labeled according to the schemes described in the following two subsections.

The second stage, depicted from Step 4 to Step 8, comprises five sub-stages: the quadric computation for each vertex, the quadric weight adjustment for feature and boundary edges, the placement of the new vertex for each contraction candidate, the contraction cost calculation, and the construction of a min-heap that prioritizes the contractions.

The contraction of edged vertex pairs is actually executed in the final stage, which includes Step 9 and Step 10. The algorithm is stated as follows:

Step 1. Collect all edges $(\mathbf{v}_i, \mathbf{v}_j)$ as contraction candidates.

Step 2. Determine feature edges.

Step 3. Label all edges as interior edges, simple edges, or boundary edges.

Step 4. Compute an initial quadric $\mathbf{Q}_p$ for each plane p.

Step 5. For the quadric $\mathbf{Q}_i$ of each vertex $\mathbf{v}_i$, it is the sum of all quadric $\mathbf{Q}_p$ derived from the set of planes associated with vertex $\mathbf{v}_i$.

1) Add large weights to the quadrics of feature edges.
2) Add large weights to the quadrics of boundary edges.

Step 6. For each contraction candidate:

1) Compute the quadric $\mathbf{Q}$ of the new vertex $\bar{\mathbf{v}}$: $\mathbf{Q} = \mathbf{Q}_i + \mathbf{Q}_j$.
2) Select a target position $\bar{\mathbf{v}}$:
   (1) If $\mathbf{A}$ is not singular, the target position is given by $\bar{\mathbf{v}} = -\mathbf{A}^{-1}\mathbf{b}$.
   (2) If $\mathbf{A}$ is singular, the midpoint of the $\mathbf{v}_i$ and $\mathbf{v}_j$ acts as the target position.

Step 7. Compute the contraction cost $\mathbf{Q}(\bar{\mathbf{v}})$.

Step 8. Construct a min-heap upon $\mathbf{Q}(\bar{\mathbf{v}})$ to prioritize the contractions.

Step 9. Edged vertex pair contraction:

1) Remove the least cost vertex pair $(\mathbf{v}_i, \mathbf{v}_j)$ from the heap.
2) Perform the contraction $(\mathbf{v}_i, \mathbf{v}_j) \rightarrow \bar{\mathbf{v}}$.
3) Re-compute the quadrics and costs of $\bar{\mathbf{v}}$ and its neighbors as Step 6.
4) Rebuild the heap.

Step 10. Iterate Step 7 to Step 9 until the desired approximation is reached.

4.2. Face color preservation

Given a polygonal mesh, suppose that each face has a color value, $[r\ g\ b]^{\mathrm{T}}$, where $0 \le r, g, b \le 1$. Our algorithm is initially to label each edge as either a normal or feature edge using the following criterion: if the difference of the color values of the two triangles incident to the edge exceeds a threshold τ, we designate the edge as a feature edge, and the remaining edges are called normal edges.

Referring to Fig. 3, assume the colors corresponding to the triangles incident to the thick line be $\mathbf{c}_1 = [r_1\ g_1\ b_1]^{\mathrm{T}}$ and $\mathbf{c}_2 = [r_2\ g_2\ b_2]^{\mathrm{T}}$. Label the thick line as a feature edge if $|r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2| > \tau$; otherwise, we label it as a normal edge. For each feature edge, we penalize it by adding a large weight to the quadrics of both endpoints of the edge.

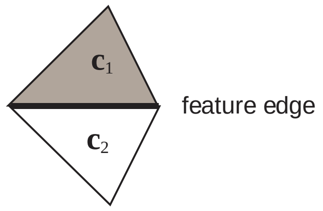


Figure 3 Illustration of a feature edge.

It is noted that although we only show how to preserve face colors in this paper, our method can be easily extended to handle vertex attributes as well. By taking the maximum difference of the attribute values of the vertices adjacent to a given vertex, we can classify all vertices of a polygonal mesh into feature or normal vertices.

4.3. Boundary edge preservation

For a polygonal mesh such as terrain height fields, the boundary edges must be preserved during the simplification process. In such a case, we apply a similar method as described earlier to preserve boundary edges. We first label all edges as interior edges, simple edges, or boundary edges. An interior edge is an edge that has none of its two endpoints on the boundary. A simple edge has only one endpoint on the boundary. And a boundary edge has both of its endpoints on the boundary. The three types of edges are illustrated in Fig. 4, where the thick lines are boundary edges, the dotted lines are simple edges, and the thin lines are interior edges.

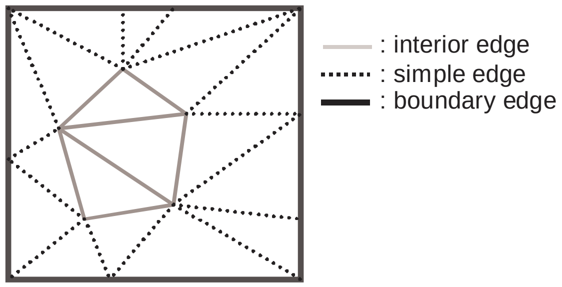


Figure 4 The three types of edges.

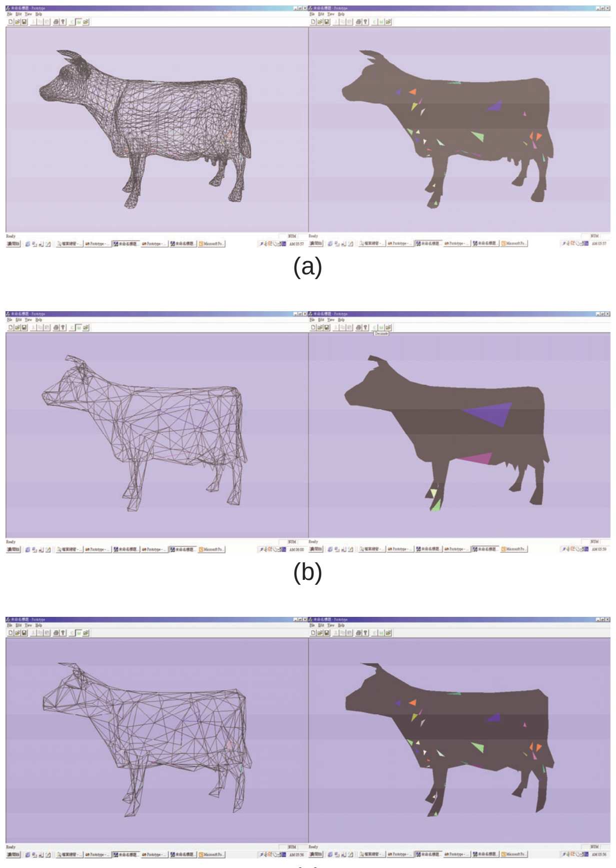
To preserve the boundary edges, we adjust the weights of the quadrics of the two endpoints of an edge such that the contraction cost of a boundary edge is the largest but that of an interior edge is the smallest. However, this approach suffers from a disadvantage that gaps might appear when the simple edges are contracted. To solve this problem, we restrict the placement of the new vertex to the boundary vertex.

5. Experimental results

In this section, we will discuss and present some experimental results from our work. All the program codes written for the experiments are implemented with OpenGL and Visual C++ 6.0, which are executed on a personal computer equipped with 600MHz AMD Athlon processor and ASUS V3800 graphics board under Microsoft Windows 98 operating system. The test models are all collected from the public domain. Thanks to those who offered these models. First, we demonstrate how our algorithm performs when face colors are considered. Next presented are the results from our method in preserving boundary edges as well as feature edges. Finally, we compare the latency of resolution changes for the progressive meshes of various models.

5.1. Face color preservation of the Cow model

Figure 5 shows the mesh simplification on the Stanford Cow model. Note that in the Cow model, we have randomly colored 30 triangles. Figure 5(a) shows the original Cow model with 5,804 triangles where 30 triangles are colored. The simplification results are illustrated in Figs. 5(b) and 5(c), where the former shows the result simplified without preserving face colors and the latter shows the result simplified with our method. It is quite obvious that our method works very well in preserving both the surface geometry and face colors of the model.

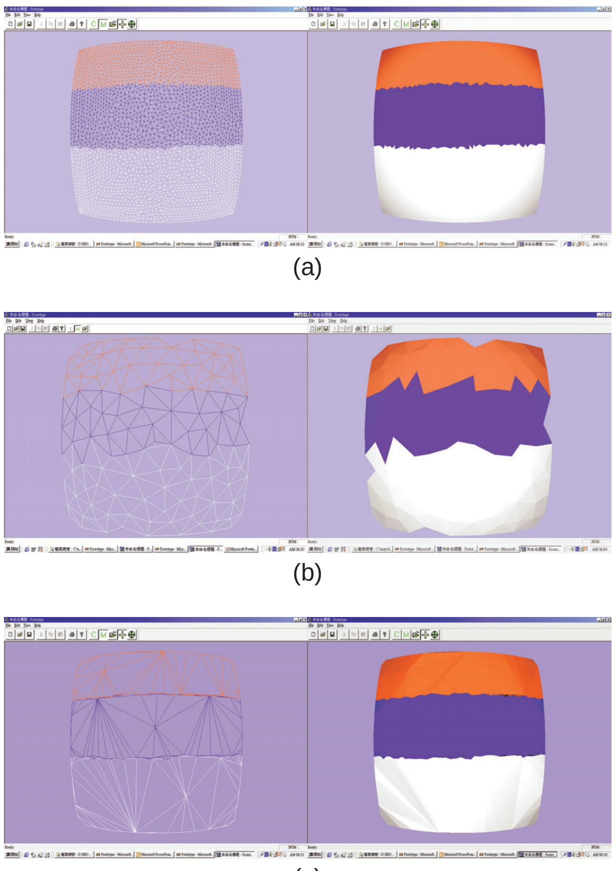


(c)

Figure 5 The Cow model: (a) the original mesh consisting of 5,804 triangles, including 30 colored triangles; (b) the simplified mesh consisting of 500 triangles without face color preservation; (c) the simplified mesh consisting of 500 triangles with face color preservation.

5.2. Boundary edge preservation of the Curve model

To gain further insight of the capability of our method in preserving boundary edges, we use the Curve model as the test model, and the simplification results are shown in Fig. 6. Figure 6(a) presents the original Curve model that possesses 5,000 triangles; Fig. 6(b) is a simplified version comprising 200 triangles whose boundary edges are not preserved; Fig. 6(c) is a simplified version resulting from our method, which has the same number of triangles as in Fig. 6(b). We can see that both face colors and boundary edges are preserved very well.



(a)

(b)

(c)

Figure 6 The Curve model: (a) the original mesh consisting of 5,000 triangles; (b) the simplified mesh consisting of 200 triangles without both face color and boundary edge preservation; (c) the simplified mesh consisting of 200 triangles with both face color and boundary edge preservation.

5.3. Latency comparison

In this experiment, we adopt Slope, Car, Bones, Cow, Bunny, and Curve models retrieved from the public domain to test the efficiency of the progressive meshes. We first simplify each of the listed polygonal meshes to 200 triangles then measure the latency by iteratively changing the resolution back and forth from the coarsest to the finest 100 times. The averaged results are recorded in Table 1, from which we can easily find that the latency of such resolution changes is increased roughly proportional to the number of triangles of a polygonal mesh. As opposed to Garland and Heckberts' approach in preserving vertex attributes, our method needs only to enlarge the weight values of the initial quadrics of the two endpoints instead of increasing the dimension of the quadric matrix. Thus, our method employs only ten floating-point numbers to represent the quadric.

Table 1 Comparison of the Latency of Resolution Changes for the Progressive Meshes of Various Models

Model	Original mesh (No. of triangles)	Average Latency (sec.)
Slope	2,432	0.013
Car	2,732	0.014
Bones	4,204	0.025
Cow	5,804	0.040
Bunny	14,999	0.119
Curve	18,050	0.128

6. Conclusions

In this paper, we have presented a method of polygonal mesh simplification with face color and boundary edge preservation based on the quadric error metric. The method creates high fidelity approximations by generating a progressive mesh from a series of edge contractions guided by the quadric error metric. In addition to the high quality approximation of the surface geometry, the proposed method also preserves face colors and boundary edges with a new and simple constraint scheme.

As we have shown in Section 5, the quality of the resulting simplified meshes from our experiments is very satisfactory. The proposed method is simple, fast, and memory efficient. However, there is a plenty of space for further improvement. Note that the attributes that we deal with, in this paper, are discrete attributes associated with the faces of a polygonal mesh. For the scalar attributes associated with the vertices of the polygonal mesh, our algorithm is not suitable.

A few existing algorithms can preserve scalar attributes [2], [15], [20]-[26]. Most of them often spend a lot of memory space and need some complicated surface re-mapping or re-sampling schemes to produce texture patches. Hence, it is worthwhile to develop a fast and memory efficient algorithm to preserve both scalar and discrete attributes in the future work.

Furthermore, in our method, the weights and the constraints are adjusted manually. This adjustment is tedious and heuristic; thus, an automatic weight adjustment scheme is preferable. Moreover, if the polygonal mesh contains more than one kind of features to preserve, for example, face colors and boundary edges in our case, providing a way to calculate the relative weights between the features automatically can help us to do the trade-off when further simplification is required.

References

[1] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 209-216, 1997.

[2] M. Garland and P. S. Heckbert, "Simplifying surfaces with color and texture using quadric error metrics," in *Proc. of IEEE Conf. on Visualization*, pp. 263-269, 1998.

[3] E. Puppo and R. Scopigno, "Simplification, LOD and Multi-resolution: Principles and Applications," in *Tutorial Notes of the Annual Conf. of the European Association for Computer Graphics*, 1997.

[4] P. Cignoni, C. Montani, and R. Scopigno, "A comparison of mesh simplification algorithms," Computers & Graphics, Vol. 22, No. 1, pp. 37-54, 1998.

[5] M. Garland, "Multi-resolution modeling: survey and future opportunities," in *State of the Art Reports of the Annual Conf. of the European Association for Computer Graphics*, pp. 111-131, 1999.

[6] J. M. Lounsbery, *Multi-resolution analysis for surfaces of arbitrary topological type*, Ph.D. Dissertation, Dept. of Computer Science, Univ. of Washington, Seattle, WA, 1994.

[7] M. Eck et al., "Multi-resolution analysis of arbitrary meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 173-181, 1995.

[8] A. Certain et al., "Interactive multi-resolution surface viewing," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 91-98, 1996.

[9] J. Rossignac and P. Borrel, "Multi-resolution 3D approximations for rendering complex scenes," in *Geometric Modeling in Computer Graphics*, B. Falcidieno, and T. Kunii, Eds., pp. 455-465, Heidelberg: Springer Verlag, 1993.

[10] K. L. Low and T. S. Tan, "Model simplification using vertex clustering," in *Proc. of the ACM Symp. on Interactive 3D Graphics*, pp. 75-82, 1997.

[11] D. Luebke and C. Erikson, "View-dependent simplification of arbitrary polygonal environments," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 199-208, 1997.

[12] P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 259-262, 2000.

[13] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 65-70, 1992.

[14] A. Ciampalini et al., *Multi-resolution Decimation Based on Global Error*, Technical Report C96-021, Centro Nazionale Universitario di Calcolo Elettronico– National Research Council of Italy, Pisa, Italy, 1996.

[15] C. Bajaj and D. Schikore, "Error-bounded reduction of triangle meshes with multivariate data," Proc. of SPIE, Vol. 2656, pp. 34-45, 1996.

[16] R. Klein, G. Liebich, and W. Straßer, "Mesh reduction with error control," in *Proc. of IEEE Conf. on Visualization*, pp. 311-318, 1996.

[17] R. Ronfard and J. Rossignac, "Full-range approximation of triangulated polyhedral," in *Proc. of the Annual Conf. of the European Association for Computer Graphics*, 1996.

[18] J. Cohen et al., "Simplification envelopes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 119-128, 1996.

[19] H. Hoppe et al., "Mesh optimization," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 19-26, 1993.

[20] H. Hoppe, "Progressive meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 99-108, 1996.

[21] H. Hoppe, "Efficient implementation of progressive meshes," Computers & Graphics, Vol. 22, No. 1, pp. 27-36, 1998.

[22] H. Hoppe, "New quadric metric for simplifying meshes with appearance attributes," in *Proc. of IEEE Conf. on Visualization*, pp. 59-66, 1999.

[23] P. Cignoni et al., "A general method for preserving attribute values on simplified meshes," in *Proc. of IEEE Conf. on Visualization*, pp. 59-66, 1998.

[24] J. Cohen, D. Manocha, and M. Olano, "Simplifying polygonal models using successive mappings," in *Proc. of IEEE Conf. on Visualization*, pp. 395-402, 1997.

[25] J. Cohen, M. Olano, and D. Manocha, "Appearance preserving simplification," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 115-122, 1998.

[26] P. Sander et al., "Texture mapping progressive meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, pp. 409-416, 2001.