Material-Preserving Progressive Mesh Using Geometry and Topology Simplification

Shu-Kai Yang Jung-Hong Chuang
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, Republic of China

Abstract

Level of detail (LOD) modeling or mesh reduction has been found useful in interactive walkthrough applications. Previous solutions in general employ either geometry or topology simplification, each alone has its own strength. Progressive meshing techniques based on edge or triangle collapsing have been recognized useful in continuous LOD, progressive refinement, and progressive transmision. We present a mesh reduction scheme that combines geometry and topology simplification, and produces a progressive mesh which generally has more than three vertices collapsed between two adjacent levels. The method also preserves the color discontinuity, which is perceptually important; but has been less studied in the literature.

Keywords: Virtual reality, level of detail, progressive mesh, material preserving, topology simplification

1 Introduction

In virtual reality applications, maintaining a fast and constant frame rate is crucial for achieving a smooth and realistic visual perception. Although graphics hardware has advanced remarkably, complex scenes defined by millions of polygons can easily exceed the maximum number of polygons that can be processed in real time. Moreover, the advance in graphics hardware stimulates the demand for displaying even larger data sets. One way to achieving a fast frame rate is to reduce the polygon flow that is sent to the graphics pipeline for shading. Traditional methods, which involve clipping, hierarchical traversal, and culling, are no longer effective for complex virtual environments. For rendering architecture buildings, methods have been proposed that aim to obtain a potential visible set by performing cell segmentation, cell-to-cell visibility, and finally eye-to-cell visibility [20]. For rendering environments with large occluders, the so called conservative visibility has been proposed to eliminate polygons behind the occluders [6].

When applied to complex scenes, these methods alone, however, may not achieve a fast frame rate simply because objects in the scene are modeled by just too many polygons. It has become apparent that in addition to the accurate geometric representation of models we do need specialized representations tailored for efficient graphical display. In practical point of view, when navigating in an 3D environment, a participant does not really see all objects in their full detail. For example, when an object is far away, small features might not be seen even the object itself is still a visually significant element in the scene. In this respect, level-of-detail (LOD) modeling has become a standard object representation, in which each object is associated with a set of representations with different levels of detail. When an object is rendered, its representation of a proper detail will be chosen depending on observer's distance, view angle, movement criteria, and so on.

Many algorithms proposed concentrate on preserving curvature and sharp edges; but assuming no modification on the topology. Algorithms of this kind might suffer from the difficulty on obtaining LOD with low-reduction rate. On the other hand, some algorithms achieve very low-reduction rate by employing topology simplification; but fail to have a good preservation on the shape. Progressive meshing have been emphasized aiming to provide a continuous LOD, progressive refinement, and progressive trsnamission. Current progressive meshing algorithms, however, tend to collapse only edges or triangles, and hence possess a very long sequence

of meshes. Another issue that is important but has been addressed less is the preserving of material property, especially the discontinuity of material attributes such as color.

In this paper, we are primarily concerned with how to achieve several desired features that come from several mesh simplification methods all together in a single reduction scheme. The method proposed employs both geometry and topology simplification, and in the meantime preserves the color discontinuity. The paper is organized as follows. Section 2 outlines related work on LOD generation. Our approach is described in Section 3. Implementation issues and experimental results are shown in Section 4. Finally, conclusions are drawn in Section 5.

2 Related Work

Many mesh simplification algorithms have been proposed. Most algorithms work by applying local geometry based criteria for simplifying small regions on the meshes [7, 10, 18]. The criteria are iteratively applied until the criteria are no longer satisfied or a user-specified reduction rate is achieved. Some algorithms optimize the geometric simplification globally over the whole mesh [5, 11, 21]. Most of these methods have satisfactory capability on preserving shape; but might fail to achieve very low reduction rate mainly due to their constraint on topology preserving. Some local geometry based algorithms have been extended to modify topology while doing geometry simplification [14, 17]; the topology simplification is, however, limited. The method that simplifies both geometry and topology is the vertex clustering algorithm [16]. The method is extremely general, as it works on any type of meshes, it can achieve very low reduction rate, and it can eliminate small geometric features and change the topology of a model. It, unfortunately, does poorly on shape preservation.

Traditional simplification methods yield meshes for a number of levels of detail, usually 5 to 10, for a given model. A limited number of levels of detail for selection often results in noticeable flickers when switching between different levels of detail. The so called *continuous level-of-detail* or *progressive mesh* is one of the useful representations to reduce the popping effect. Progressive mesh allows a smooth visual transition between various levels of detail. The progressive mesh can be obtained by removing vertices [17]; collapsing edges [10]; and collapsing triangles [9].

Schroeder et al. [18] have proposed a vertex decimation that simplifies the mesh by removing vertices. Vertices are first categorized to five categories, namely simple, interior edge, boundary, corner, and complex. Among the five categories, corner and complex vertices are retained while the rest can be removed if the vertex satisfies a distance criterion. Removing a vertex from the mesh creates a hole that can be filled by retriangulation. The decimation method have been extended to generate progressive meshes in [17]. In [12] the uniform cluster cells are replaced by so called *floating cells*, which are centered around their vertex of highest weight.

Hoppe [10] describes a progressive meshing method based on edge-collapsing operation. Edges are first ordered according to the cost that is the result of an energy minimization function. The cost in general measures the amount of error introduced into the model as the result of collapsing the edge. Edges are then repeatedly collapsed. At each collapsing the edge of the lowest cost is collapsed and the costs of adjacent edges are updated. Each edge collapsing yields a mesh with two triangles less then the mesh of previous level. The result is a *base mesh* together with a sequence of edge-collapsing records, each of which can be used to recover finer representation of the mesh. Edge-collapse methods that incorporate with different cost evaluations have been described in [7, 14, 15]

Gieng et al. [9] describes a progressive meshing method through triangle-collapsing operation. Triangles are sorted based on the approximate curvature for the underlying surface in the area of the triangle. The progressive mesh is obtained by repeatedly collapsing a number of triangles that can be collapsed simultaneously. Thus, the result is a bash mesh together with a sequence of a set of simultaneous triangle-collapsing operations.

Another issue that has not been addressed much is the preservation of material property. Two emphases on material preservation have been considered: the attribute associated with vertices over the mesh and attribute discontinuity. The attribute associated with vertices are preserved during simplification by texture mapping that is built using a mapping between the vertices of the original mesh and the simplified mesh [2, 3, 4, 13, 19]. Garland et al. [8] extends the quadric error metrics [7] to account for material attributes associated with vertices and attribute discontinuity.

3 Material-Preserving Progressive Mesh

3.1 Overview of the Method

Most traditional methods for generating progressive mesh based on geometric simplifications, such as edge or triangle collapsing and vertex decimation, and some local-topology modification as well. The proposed algorithm aims to produce an effective progressive mesh by (a) allowing more than three vertices to be collapsed or clustered at each level, (b) empolying geometric simplification as well as topology simplification that involves local and global topology modification, and (c) using effective criteria to preserve geometric shape, especially sharp feature, and color discontinuity.

The proposed algorithm begins with a preprocessing, in which each vertex is classified into five categories and evaluated to yield a weight and a priority value, then the bounding box of the given mesh is uniformly subdivided into cells of size τ. The algorithm then enters a simplification loop, in which each cycle yields a simplified mesh. In each cycle of mesh simplification loop, we do the following:

1. Select a vertex with the highest priority value to be the representative for the next clustering operation,
2. Create a floating cell of sie τ that is centered on the representative to confirm the spatial range of vertex clustering,
3. Start at the representative and generate the spanning tree for all vertices that are inside the floating cell and can be clustered to the representative,
4. Cluster all vertices in the spanning tree to the representative. Delete the triangles that contain two or three clustered vertices, and replace the clustered vertex by the representative for triangles that contain one clustered vertex,
5. Record the clustered vertices, vanishing triangles, and the vertex replacements,
6. Update the weights and priority values for the representative and its neighboring vertices.

The cycle is repeated until a user-specified reduction rate is reached. The loop yields a sequence of meshes M^n, $M^{n-1}, \cdots, M^0$, for some n, in which M^n is the original mesh and M^0 is the most simplified mesh called *base mesh*. The resulting progressive mesh consists of the base mesh M^0 and the sequence of recorded information necessary for the refinement.

3.2 Vertex Categorization

All vertices of the given mesh are classified into five categories based on the material of triangles incident to the vertex. Such a vertex categorization is different from that found in mesh decimation [18]. A vertex is a *simple vertex* if all triangles incident to it form a loop and are of the same material. A simple vertex is an *edge vertex* if visiting the loop of incident triangles encounters two material changes. A vertex is a *boundary vertex* if it is geometrically on the boundary of the mesh. A simple vertex is a *corner vertex* if visiting the loop of incident triangles encounters more than two material changes. A vertex is a *non-manifold vertex* if it possesses more than one loop of incident triangles. See Figure 1 for illustration. Vertex category for the vertex v is an integer denoted by $category(v)$ as shown in Figure 1.

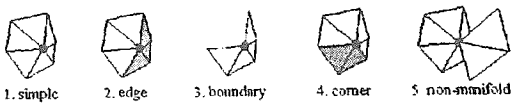


Figure 1: Vertex categories.

3.3 Priority Assignment, Penalty Function, and Weighting

Such a vertex categorization is useful in preserving material border and open boundary in the sense that vertices of lower category value can not cluster those with higher values. In Step 3 of the loop, we need to know if a vertex u can be clustered to representative v. To this end, besides vertex category, we need also to consider the perception importance of a vertex and the penalty of clustering a vertex to another. Here, the perception importance of a vertex v is denoted as $weight(v)$ and the penalty of clustering u to v is denoted by $penalty(u, v)$. We will show how to define $penalty(u, v)$ in such a way that we can claim "u can be clustered to v if $penalty(u, v) \leq \epsilon$, where ϵ is a model-independent constant". This will eliminate the difficulty of determining a threshold, which is usually model-dependent. Now, a vertex u can be clustered to v if the following three conditions are satisfied simultaneously:

1. $category(v) \geq category(u)$.
2. $weight(v) \geq weight(u)$ if $category(v) = category(u)$.

3. $penalty(u, v) \leq \epsilon$, for some fixed ϵ.

3.3.1 Priority Assignment

Selecting the vertex with the highest perception importance as the representative in Step 1 will end up with an undesirable progressive mesh since, using that progressive mesh, less perceptionally important portions will be refined first from the base mesh. Here, we define the *priority value* of a vertex v as the number of vertices that can be clustered into v. The vertex with the highest priority value will be selected as the representative for clustering so that geometrically or perceptionally important portions get refined first from the base mesh.

3.3.2 Penalty Function

The function $penalty(u, v)$ should be designed to preserve the geometric features such as sharp features and areas of high curvature. In reviewing such geometric criteria found in literature, many of them require user-specified parameters, which are in general model-dependent and hence require prior knowledge. In developing the $penalty(u, v)$, we have tried to alleviate such constraints.

For the penalty function of clustering u to v, we consider the following two factors:

1. $d = \|u - v\|$, representing the distance between u and v, and

2. $n = \max_f(\|n'_f - n_f\|)$, where f is a triangle that is adjacent to u; but not v, n_f and n'_f are unit normals of f before and after clustering u to v, respectively. The value n represents the maximum normal difference of relevant triangles resulting from clustering u to v.

In edge collapsing approach, the $penalty(u, v)$ is usually defined as a linear combination of d and n; i.e.,

$$penalty(u, v) = \alpha d + \beta n,$$

for some α and β. The edge (u, v) will be collapsed if $penalty(u, v)$ is less than or equal to a user-specified threshold ϵ. Such a criterion poses a few problems. First of all, the scale of d is model-dependent and value of n is in the range of $[0, 2]$. As a consequence, when the scale of d is large, $penalty(u, v)$ will not change no matter what n is; on the other hand, when the scale of d is small, the value of d cannot influence the value of $penalty(u, v)$ much. Secondly, a fixed (α, β) doesn't work for all models, especially due to the first problem. Hence, different models may require different pairs of (α, β). Thirdly, the threshold ϵ is model-dependent and it is hard to predict the reduction result for a particular ϵ.

To alleviate the problems, we formulate $penalty(u, v)$ as

$$penalty(u, v) = f(d)\, g(n),$$

and expect that f and g reveal the following three properties:

1. $g(n)$ remains very small when n is very small and increases rapidly when n is larger than a particular number. This will encourage the clustering to proceed when n is small and d is arbitrary, and will limit the clustering when n is large.

2. The scale of $f(d)$ should be model-independent such that $f(d)$ ranges inside a fixed interval. We expect that $f(d)$ is small when d is small, and remains fixed when d is not small; that is, when d is not small, the value of $penalty(u, v)$ will largely depend on $g(n)$. This will allow the clustering to proceed when d is small and n is arbitrary. Hence small geometric features of high curvature can be eliminated in the reduced mesh.

3. The choice of f and g should allow the threshold ϵ to be model-independent.

Ideal shapes of $f(d)$ and $g(n)$ are shown in Figure 2. Our choice is $f(d) = (\frac{d}{\tau})^{\frac{1}{4}}$ and $g(n) = e^n$; that is,

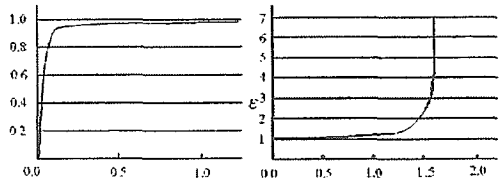


Figure 2: Ideal shapes of $f(d)$ and $g(n)$.

$$penalty(u, v) = (\frac{d}{\tau})^{\frac{1}{4}} e^n,$$

where τ is the cell size in the uniform subdivision. Figure 3 shows the shapes of $f(d)$ and $g(n)$. Since

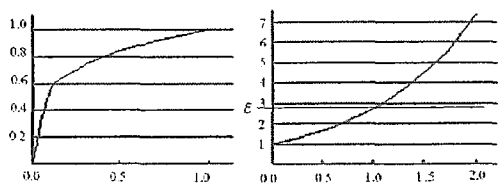


Figure 3: Shapes of $f(d) = (\frac{d}{\tau})^{\frac{1}{4}}$ and $g(n) = e^n$.

the clustered vertex must be inside the floating cell, the value of $\frac{d}{\tau}$ must be less than or equal to 1, and so does $f(d)$. Since $n \in [0,2]$, $g(n) = e^n \in [1, e^2]$. Consequently, for model of any scale, $penalty(u,v) = (\frac{d}{\tau})^{\frac{1}{4}} e^n$ must be inside $[0, e^2]$, and hence the size of ϵ is independent on the scale of the model and can be chosen by the system. Since for a given ϵ there exist a τ such that $penalty(u,v) = (\frac{d}{\tau})^{\frac{1}{4}} e^n$ is less than or equal to ϵ. The system begins with an ϵ and a τ, and doubles the value of τ when the user-specified reduction rate has not been achieved. Such a process repeats until the rdeuction rate is achieved.

3.3.3 Weighting Function

The weight of a vertex considered in [16] takes the combination of the maximum angle between incident edges and the maximum length of incident edges into account. Similarly, $weight(v)$ here is defined by

$$weight(v) = \max_u(\|u - v\|)e^{max_f(\|n_v - n_f\|)}$$

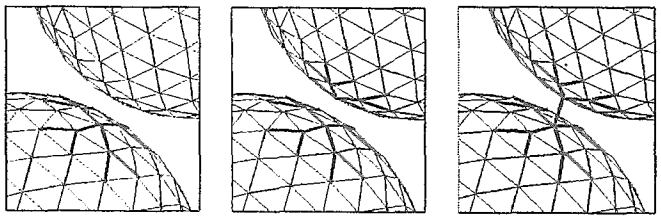
where u is the neighboring vertex of v, f is the triangle containing v, n_v and n_f are unit normals at v and of f, respectively.

3.4 Clustering by Spanning Tree Expansion

Starting from the representation v, we next seek all vertices that can be clustered to the representative. Those vertices must be inside the floating cell and satisfy three conditions described in previous section. This search of clustered vertices can be performed by the spanning tree expansion starting from the representative. Such a spanning tree can be constructed by a branch-first search over the mesh. Note that special cares must be taken in the tree expansion since loops exist on the mesh.

Separate meshes may exist inside a floating cell. When such a case happens, the spanning tree expansion repeats as follows. After the spanning tree T starting from the representation v is constructed, we find in a separated mesh a vertex v' that is closest to the representative v and construct the spanning tree starting from v'. The constructed spanning tree T' then becomes a child of v; that is, T' becomes a sub-tree of T under v. Such a search for v' continues until all separate meshes are considered. Figure 4 illustrates the construction of two separate spanning trees and how they are connected.

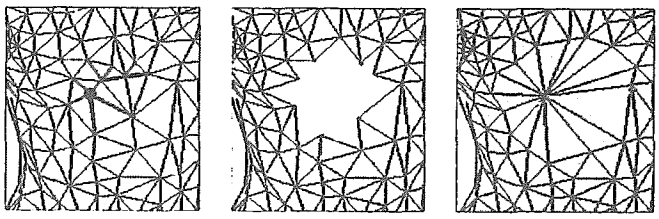
After the spanning tree is constructed, all vertices except the root are clustered to the root. Those triangles with two or three vertices clustered become



(a) Spanning tree T. (b) Spanning trees T and T'. (c) Connect T' to T.

Figure 4: Spanning trees on separate meshes and their connection.

degenerate and are removed from the mesh, while those with one vertex clustered are retained but with the clustered vertex replaced by the root (the representative). Figure 5 shows the triangle removal and the vertex replacement.



(a) Original mesh. (b) After triangle removal. (c) After vertex replacement.

Figure 5: Clustering operation.

3.5 Global Error Control and Error Accumulation

The penalty function proposed considers only local geometry and cannot yield a global error control, in which the displaced-distance of each vertex in the base mesh is required to be less than or equal to a user-specified tolerance. The computation for ensuring the global error control, however, is usually complicated. Here we consider an approximate global error control, which is similar to the method in [1]. We associate with each vertex u a value, called $shift_{max}(u)$, representing the maximum displaced-distance of the vertices clustered to u. Then when determining if u should be clus-

tered to v, we replace d in the $penalty(u,v)$ by $d+shift_{max}(u)$. Such a replacement will ensure that $d+shift_{max}(u)$ will be less than or equal to τ; otherwise $(d+shift_{max}(u))/\tau$ will be larger than 1, and in turns, $penalty(u,v) > 1$ will hold. Note that, after clustering u to v, the $shift_{max}(v)$ is updated to $shift_{max}(v) = \max_u(\|u-v\|+shift_{max}(u))$, for all u that is clustered to v.

A carefully modeled mesh often has more triangles in the areas of higher curvature. The simplification scheme, however, may repeadtedly try to simplify these areas of higher curvature. To avoid such repetition, we can associate each vertex v with a $shift_{accum}(v)$ and accumulate $\|u-v\|$ to $shift_{accum}(v)$ after clustering u to v; that is, $shift_{accum}(v) = \sum_u \|u-v\|$. In computing $penalty(u,v)$, we replace d by $d+shift_{accum}(v)$. If v has been repeatedly selected as a representative, the proposed scheme will limit the number of vertices that can be clustered to v and hence v will not serve as a representative after v has clustered many vertices. With this scheme, the shape of a high curvature area can be easily preserved by retaining more vertices. The global error control and error accumlation can be achieved simultaneously by replacing d in $penalty(u,v)$ with $d+\max(shift_{max}(u), shift_{accum}(v))$.

3.6 Progressive Mesh Generation and Defragmentation

The clustering proceeds until a user-specified reduction rate is obtained and produces a sequence of meshes, denoted as $M^n, M^{n-1}, \cdots, M^1, M^0$, for some n. All eliminated vetices and triangles, and vertex replacements for each clustering operation are recorded. Thus, using this sequence of recorded information, the original mesh can be recovered by refinning from M^0.

In our implementation, all triangles and vertices are stored in arrays. The refinement records are usually stored in scattered positions, which complicates the data structure and slows down the refinement, and hence should be defragmented. As shown in Figure 6, triangles in the base mesh M^0 are stored as the first part of the triangle array, the added triangles for each refinement are then stored in a slot-by-slot basis and in the order of refinement. Vertex array and vertex replacement array are arranged in a similar way. The refinement and coarsing of the progressive mesh can then be done by simply moving the index offsets to the triangle array and vertex array, and performing a number of vertex replacements indicated on the vertex replacement array. Using such a data structure, every refinement takes only $O(1)$ time to update the index offsets, and the refinement from M^0 to M^n takes $O(m)$ time, where m is the triangle number of the original mesh. Hence the array structure with defragmentation can simplify the data structure and speed up the refinement process.

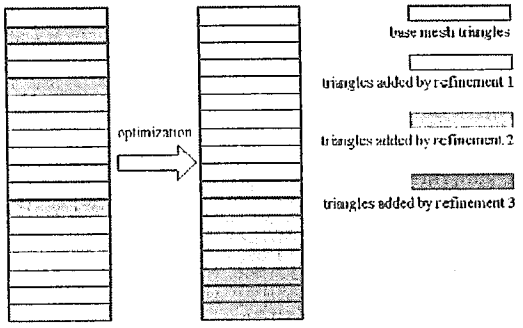


Figure 6: Defragmentation of triangle array.

3.7 Level Reduction

Although the progressive meshing is able to cluster more than three vertices, the resulting number of levels might be still too high. To further reduce the number of levels in the progressive mesh, we also repeatedly merge the pair of two adjacent refinements. As depicted in Figure 7, the refinements R_i and R_{i+1} are merged to R'_{i+1}, similarly the coarsings C_{i+1} and C_i are merged to C'_{i+1}. Such a mergence needs to take unions of added vertices, triangles, and vertex replacements for R_i and R_{i+1}. Before taking the union of vertex replacements for R_i and R_{i+1}, we need to do the following:

1. If f is an added triangle in R_i and in the meantime has one vertex replaced in R_{i+1}, the vertex replacement for f should be done before it becomes an added triangle in R'_{i+1}.

2. If f is a triangle having vertex v_0 replaced by v_1 in R_i and then v_1 is replaced by v_2 in R_{i+1}, then f will be a triangle in R'_{i+1} that has vertex v_0 replaced by v_2.

This mergence is repeatedly applied to two adjacent levels M^j and M^{j+1} that have the least sum of triangles added in R_j and R_{j+1} until a user-specified number of levels is reached. This order of mergence tends to make the changes of triangle number more uniform among levels. Note that, the level reduction also results in a more efficient refinement and coarsing process between levels since vertex replacements between adjacent levels have been merged.

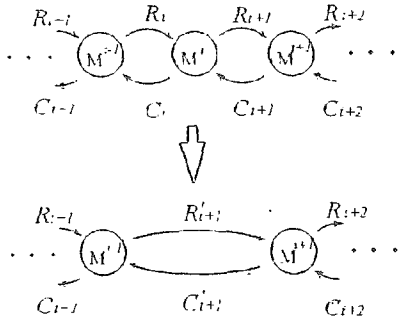


Figure 7: Merge two refinements and coarsing steps.

Models	Size(M^n)	Size(M^0)	Time in sec.	# of levels	Triangle number
Canstick	4150	498	1	734	4.98
Footbones	4204	456	1	417	8.89
Easter	4976	500	1	450	9.95
Beethoven	5030	500	2	1105	4.10
Cow	5604	496	2	501	10.60
Streetlamp	6526	498	6	1225	6.80
Teaport	9216	456	9	831	10.49
Spider	9266	498	3	867	10.14
Dog	33665	499	40	1624	20.56
Bunny	69451	500	32	3484	19.79

Triangle number: average number of triangle clustered in each level.

Table 1: Computation efficiency and clustering performance.

4 Experimental Results

The proposed method has been implemented using C language and several models have been tested on a PC with Pentium II 233MHz CPU, 64M RAM, and Window NT 4.0.

We first demostrate the computation efficiency of our method. As shown in Table 1, each of ten models has been simplified to about 500 triangles in 1 to 40 seconds. In each clustering step, in average 4.10 to 20.56 triangles are clustered.

The number of levels is in general dependent on model's geometry and the number of triangles in its original mesh. For the test models, 412 to 3484 levels are obtained. Levels of all models are also reduced to 100 levels. On each model, we have performed 20000 runs each of which refines the mesh from M^0 to M^n followed by simplifying from M^n to M^0. Table 2 shows the timing data of this experiment for the progressive meshes with original levels and reduced levels. For progressive meshes with higher number of levels, the efficiency gain from level reduction is more significant.

We show figures of some tested models, including Beethoven scupture, bunny, footbone, spider, and easter. Beethoven scupture has many geometric features in hair, on face, on necktie and cloth. Using geometric simplification alone, low reduction rate will be hardly obtainable due to the preservation of small

Models	# of levels of levels	Average refining time in second	Reduced #	Average refining time in second
Canstick	734	0.000675	100	0.000500
Footbones	417	0.000550	100	0.000450
Easter	450	0.000650	100	0.000525
Beethoven	1105	0.000900	100	0.000675
Cow	501	0.000725	100	0.000550
Streetlamp	1225	0.001375	100	0.001325
Teaport	831	0.001175	100	0.000875
Spider	867	0.001175	100	0.000700
Dog	1624	0.003775	100	0.002875
Bunny	3484	0.014300	100	0.004550

Table 2: Average time in second for refining from M^0 to M^n.

geometric features. On the other hand, cluster approach may eliminate small geometric features in the beginning of the clustering process. The proposed penalty function is in general able to sensitively detect the effect of normal change, and in the meantime neglect the normal change when the vertex distance is small. So small geometric features on Beethoven scupture are retained in the early phase of the simplification process, and are eliminated when a low reduction rate is required. See Figure 8 for the images of Beethoven scupture with four levels. Figure 9 shows images of bunny model. Bunny model also has many small protrudent features, which are retained for models with 10000 triangles, and graduately smoothed out in model with 1000 triangles, and totally smoothed out in model with 491 triangles. The bunny model with 100 triangles still preserves its global shape.

The topologies in footbones and spider model are modified or simplified in our test. In footbones model with 1950 triangles, disjoint but close bones are connected. In models with 1034 and 412 triangles, long or thin bones are simplified to 2D triangles or 1D segments. In spider model with 2032 triangles, joints on the foot are smoothed out. In model with 1022 triangles, foots and tentacles are simplified to 2D triangles. In model with 500 triangles, tentacles and some tail parts of the foot are simplified to 1D segments and disappear. Figures 10 and 11 show the reduced meshes and their shaded images of footbones and spider model. The preservation of color discontinuity is depicted in Figure 12. The color discontinuity on the easter model is maintained in the sense that the border line is simplified as well when the mesh is reduced.

4.1 Discussion

The proposed $penalty(u, v)$ has shown its good capability in preserving geometric features and its ability to eliminate small features such as that on cheese-like surfaces. By using the connecting spanning trees found in separate meshes inside athe floating cell, disjoint meshes can be joined. More, due to the nature

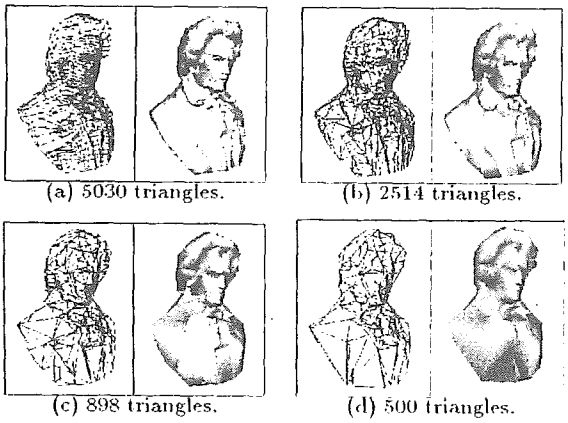


Figure 8: Beethoven's scupture.

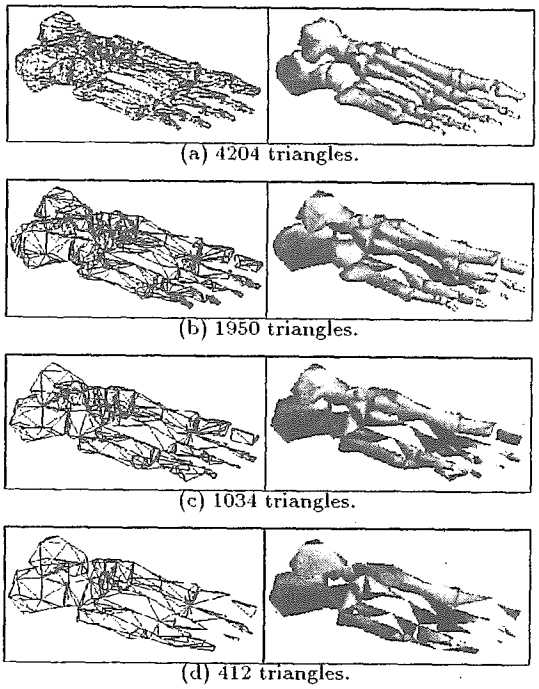


Figure 10: Footbones model.

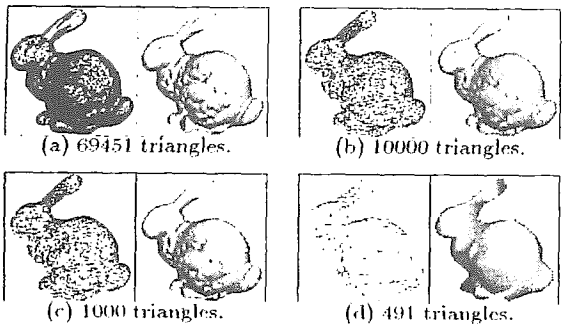


Figure 9: Bunny model.

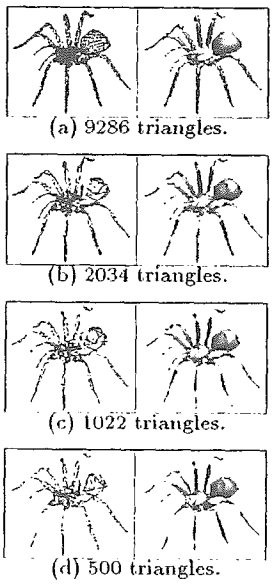


Figure 11: Spider model.

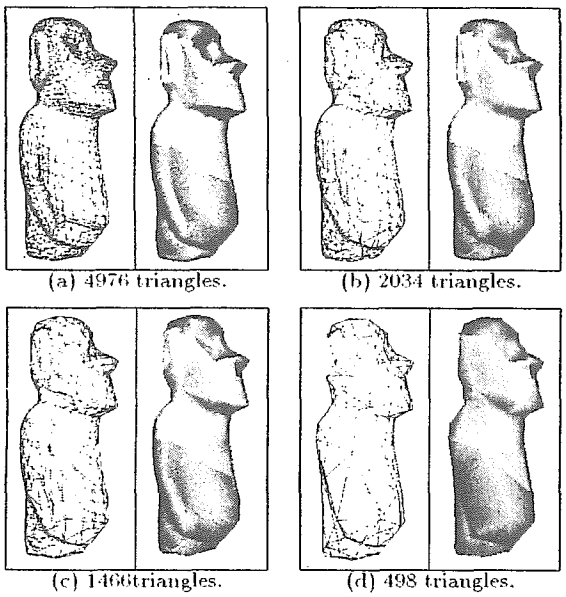


Figure 12: Easter model.

of the clustering operation, a triangle can be reduced to an edge or a vertex, and hence is eliminated. So thin portions of a given mesh can be reduced to a 2D polygon, or a 1D line segment and hence is eliminated.

5 Conclusions

We have proposed a mesh reduction method that combines the geometry and topology simplification, and their advantages as well. The method produces a progressive mesh by using a more general collapsing scheme, in which more than three vertices can be clustered in each reduction step. The method also preserves the discontinuity of color attribute (color borders), which is perceptually important but has been less addressed in previous methods. The method begins with a preprocessing, in which each vertex is classified into five categories and evaluated to yield a weight and a priority value, then the bounding box of the given mesh is uniformly subdivided. The algorithm produces a progressive mesh by repeatedly applying the simplification step, each yields a simplified mesh. In each simplification step, we (1) select a vertex with the highest priority value to be the representative for the next clustering operation, (2) create a floating cell centering on the representative to confirm the spatial range of vertex clustering, (3) search for vertices that can be clustered to the representative, (4) cluster vertices and delete degenerate triangles, and finally (5) record the clustered vertices, vanishing triangles, and the vertex replacements. We have found that the proposed penalty function can effectively preserve geometry features and is able to eliminate small features of high curvature. The involved clustering operation has also shown its strength on topology simplification. The color discontinuity is also effectively preserved in such a way that the color border is simplified as well when the mesh is reduced. As the future research directions, we are currently extending the proposed method to support a view-dependent refinement of progressive mesh.

References

[1] A. Ciampalini, P. Cignoni, C. Mantani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13:228–246, 1997.

[2] P Cignoni, C. Montani, C. Rocchini, and R Scopigno. A general method for preserving attribute values on simplified meshes. In *Proceedings of IEEE Visualization '98*, pages 59–66, 1998.

[3] J. D. Cohen, D. Manocha, and M. Olano. Simplifying polygonal models using successive mappings. In *Proceedings of IEEE Visualization '97*, pages 395–402, 1996.

[4] J. D. Cohen, M. Olano, and D. Manocha. Appearence-preserving simplification. In *Proceedings of ACM Siggraph '98*, pages 115–122, 1998.

[5] J. D. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. Wright. Simplification envelopes. In *Proceedings of ACM Siggraph '96*, pages 119–128, 1996.

[6] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of ACM Symposium on Interactive 3D Graphics '97*, pages 83–90, 1997.

[7] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of ACM Siggraph '97*, pages 209–224, 1997.

[8] M. Garland and P. S. Heckbert. Simplifying surface with color and texture using quadric error metrics. In *Proceedings of IEEE Visualization '98*, pages 263–269, 1998.

[9] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145-161, 1998.

[10] H. Hoppe. Progressive meshes. In *Proceedings of ACM Siggraph '96*, 1996.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH '93*, pages 19-26, 1993.

[12] K. L. Low and T. S. Tan. Model simplification using vertex-clustering. In *Proceedings of ACM Symposium on Interactive 3D Graphics '97*, pages 75-81, 1997.

[13] M. Maruya. Generating texture map from object-surface texture data. In *Proceedings of Eurographics '95*, pages 397–405, 1995.

[14] J. Popović and H. Hoppe. Progressive simplicial complexes. In *Proceedings of ACM Siggraph '97*, pages 217–224, 1997.

[15] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. In *Proceedings of Eurographics '96*, 1996.

[16] J. Rossignac and P. Borrel. Multi-resolution 3-D approximations for rendering complex scenes. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, Berlin, 1993.

[17] W. J. Schroeder. A topology modifying progressive decimation algorithm. In *Proceedings of IEEE Visualization '97*, 1997.

[18] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics (Proceedings of Siggraph)*, 26(2):65–70, 1992.

[19] M. Soucy, G. Godin, and M. Rioux. A texture-mapping approach for the compression of colored 3D triangulations. *The Visual Computer*, 12:503–504, 1996.

[20] S. J. Teller and T. A. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of ACM Siggraph)*, 25(4):61–69, 1991.

[21] G. Turk. Re-tiling polygonal meshes. *Computer Graphics (Proceedings of Siggraph)*, 26(2):55–64, 1992.