An Efficient Mesh Simplification Method with Feature Detection for Unstructured Meshes and Web Graphics

Bing-Yu Chen
The University of Tokyo
robin@is.s.u-tokyo.ac.jp

Tomoyuki Nishita
The University of Tokyo
nis@is.s.u-tokyo.ac.jp

Abstract

This paper presents an efficient method of mesh simplification for geometric 3D models. The transmission of 3D models on the Internet is an important task. The data size of a 3D model is usually large to enable more detail to be represented. Hence, it is necessary to represent the 3D model while keeping the data size small and preserving its features, even if the meshes that constitute the model are unstructured. Although there are many methods for simplifying the meshes, most of them are time-consuming. Our approach is to obtain an adequate simplified model in a short amount of time. Therefore, the model provider can check the simplified result interactively before uploading to the server. After transmitting the simplified model, if the user at the client needs to get more details, by transmitting some necessary information, the progressively increasing model detail and the original model without losses could be reconstructed.

1. Introduction

Obviously, there are more and more users that would like 3D graphics supports on the Web as the machine performance and network bandwidth improve over time. For Web Graphics (a new platform based on the Web), geometric 3D models are widely used. Therefore, how to transmit the mesh data, which constitutes the model, efficiently through the Internet, has become an important task, since the data size is usually large. If a user wants to use a geometric 3D model on a Web page, retrieving the data set is time-consuming. Unfortunately, the user on the Internet usually does not need to use such a detailed model in most cases. Hence, to offer a simplified model which has easily recognizable shapes and features of the original model is necessary. Moreover, if the user then decides that more model details are needed, then the subsequent download needs to be quick and with no retransmission of information.

Although there are many methods for simplifying geometric 3D models [7] [11] [20], most of them are generally time-consuming due to model optimization. These time-consuming methods are generally expected to be used directly if the model provider wishes to check the simplified model by changing some of the parameters interactively. Moreover, the simplified models created by some previous methods can not be easily recognized nor used to reconstruct the original model. We also presented a method for simplifying geometric 3D models and transmitting them with a QoS-like (Quality of Service) controlling method as described in [3]. However, the data size of the simplified model is still too large or the simplified model is hardly recognizable if it contains only a few faces.

The basic idea of our method is to segment the unstructured meshes of a geometric 3D model into several parts first by using feature detection methods, and then simplifying each part of the meshes iteratively. Therefore, our approach can preserve the shape and features of the original model after simplification. Furthermore, since the methods used for detecting the features of the model are simple, the performance of our approach is also better than other previous methods. Hence, when a 3D model provider wishes to upload a geometric 3D model onto a Web server using our approach, the provider could first use our system to check what resolution of simplified model is to be used by the users, and could change some of the parameters to obtain interactively a better simplified model. On the client site, the user first receives the simplified model from the Web server. Then, if the user needs to use the model with more details, the server will then transmit the additional necessary information, which is capsulated as some patches, to the client so that the client program can show increasing model detail progressively. Finally, if the user really needs the original model, after receiving all of the patches, the system is then able to reconstruct the original 3D model with no losses and no retransmission of information.

Moreover, to make our approach widely used in the Web world, we have developed all of the algorithms using exclusively the Java[1] programming language for its hardware-neutral features and wide availability on many hardware platforms. Additionally, the 3D graphics rendering is done by jGL, which is a 3D graphics library for Java with an OpenGL-like API (Application Programming Interface) provided by Chen and Nishita [2].

2. Previous work

Many researches have been carried out on simplifying the meshes of geometric 3D models. Some of them provide almost optimized meshes which can represent fine shapes and preserve the features of the original model with a small data size [4] [10]. Others simplify the meshes iteratively and store the removed information which can then be used to progressively reconstruct the lossless original model. Since our motivation is to transmit the geometric 3D model through the Internet, it is necessary to reconstruct the original model with a small amount of data transmission. Therefore, our approach belongs to the latter category. However, to allow it to be used for Web Graphics, where the run-time performance is more important than providing an almost perfect model, a more efficient method is needed. In this section, some related methods are introduced briefly.

PM (Progressive Meshes) is a famous method for 3D mesh simplification and is based on the *edge collapse* or *edge contraction* operation provided by Hoppe [12] [13] and Hoppe et al. [15]. Although this method could result in an almost optimized simplified model, it is well known to be time-consuming. A derived method, QEM (Quadric Error Metrics), has been provided by Hoppe [14], Garland and Heckbert [8] [9] to make the calculation faster. The heuristic function used by QEM is geometry-based, since it calculates the geometric distance between the newly generated vertex and the faces which are deformed before generating it. Although QEM could enhance the run-time performance of the simplification process, the simplified model is sometimes hardly recognizable due to the over-simplification. An image-based heuristic function is presented by Lindstrom and Turk [19]. It captures 20 images in every simplifying step and is obviously time-consuming. Chen and Nishita also proposed a simple method for mesh simplification based on the edge collapse operation [3]. However, without feature detection, the shape of the simplified model may hardly be recognizable.

Other algorithms are based on the *vertex decimation* operation, which are provided by Alliez et al. [1], Turk [22], and Schroeder et al. [21]. This algorithm is different from the previous ones; it could simplify a 3D model fast, but the shape of the simplified model maybe be changed and is hardly recognizable.

Therefore, an efficient mesh simplification method with feature detection for making the simplified model remain recognizable is necessary.

3. Feature edge detection

In our algorithm, to simplify a geometric 3D model efficiently and at the same time preserve its features, it is necessary to find out the feature edges of the model so that the model may be simplified by removing the non-feature edges while preserving its features.

There are two kinds of feature edges in our approach. One is the *sharp edge* due to the sharpness of the geometric differences, and also the edges of adjacent faces containing different material properties. The other is the *base edge* detected from the unstructured meshes if there is no sharp edge contained in the meshes. Before describing the two kinds of feature edges, the notation used in this paper is first introduced.

3.1 Notation

The notation used in this paper is shown in Figure 1 as [3]. A geometric 3D model M is represented as a formula containing 4 components: V is the set of vertex v_i, $i \in [1, m]$, where m is the number of vertices, and defining the shape of the meshes in $\Re^3$. F is the vertex connectivity of the meshes. D is the set of discrete attributes d_f, like the material property, associated with the face f, and S is the set of scalar attributes $s_{(v_i,f)}$, like the normal vector, associated with the wedge $w = (v_i, f)$.

$$\boxed{\begin{array}{l} M = (V, F, D, S) \\ V = \{v_i\}_{i=1}^{m}, v_i \in \Re^3 \\ F = \{f = \{j,k,l\} \mid v_j, v_k, v_l \in V\}, \ |F| \subset \Re^m \\ D = \{d_f \mid f \in F\} \\ S = \{s_{(v_i,f)} \mid i \in f\} \end{array}}$$

Figure 1. The representation of a 3D model.

3.2 Sharp edge definition

If a geometric 3D model is constructed with several different material properties as shown in Figure 2 (b) or contains some pre-defined sharp edges as shown in Figure 2 (a), the sharp edges of the model could be detected easily. An edge $e = \{i, j\}$ is called a *boundary edge* if there is

[1] *http://java.sun.com/*

only one face $f = \{i, j, k\}$ with $e \subset f$. An edge $e = \{i, j\}$ is called a *sharp edge* if either (1) it is a boundary edge, (2) its two adjacent faces f_l and f_r have different discrete attributes, i.e. $d_{f_l} \neq d_{f_r}$, or (3) its adjacent wedges have different scalar attributes, i.e. $s_{(v_i, f_l)} \neq s_{(v_i, f_r)}$ or $s_{(v_j, f_l)} \neq s_{(v_j, f_r)}$.

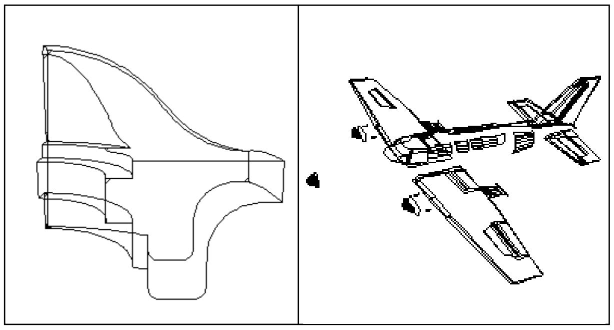


Figure 2. The examples of sharp edges.

3.3 Base edge detection

If there is no sharp edge or there are some features hidden in the meshes, the detection of base edges from the unstructured meshes is necessary. To detect the base edges, we use the ESOD (Extended Second Order Difference) operator as [16]. In this section, an operator called SOD (Second Order Difference) and the ESOD operator are described first.

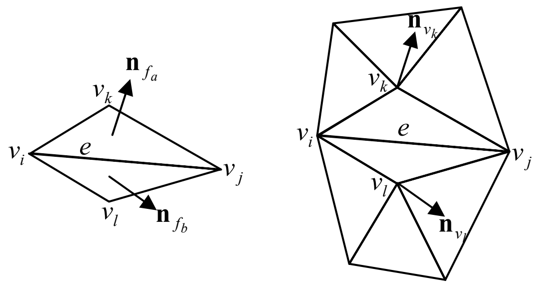


Figure 3. The (a) SOD and (b) ESOD operators.

The SOD operator is the simplest method for detecting the features from unstructured meshes. It assigns a weight to every edge $e = \{i, j\}$ defined by the normal vectors of its adjacent faces as $w(e) = \mathbf{n}_{f_a} \cdot \mathbf{n}_{f_b}$. For example, in Figure 3 (a), where $f_a = \{i, j, k\}$ and $f_b = \{i, l, j\}$ are the adjacent faces of the edge e, and the normal vector for face $f = \{i, j, k\}$ is defined as:

$$\mathbf{n}_f = (v_j - v_i) \times (v_k - v_i) / \|(v_j - v_i) \times (v_k - v_i)\|.$$

The ESOD operator extends the SOD operator. Instead of using the normal vectors of the adjacent faces of edge e, it uses the average normal vectors computed from the faces on the 1-ring of the vertices v_k and v_l as shown in Figure 3 (b). So the weight of edge $e = \{i, j\}$ is defined as $w(e) = \mathbf{n}_{v_k} \cdot \mathbf{n}_{v_l}$, where the normal vector for vertex v is defined as:

$$\mathbf{n}_v = \sum_{v \in f} area(f) \cdot \mathbf{n}_f \Big/ \sum_{v \in f} area(f),$$

where $area(f)$ means the area of face f.

Therefore, to define a proper threshold $\varepsilon = [-1,1]$, it is possible to determine the features from unstructured meshes. To use the SOD or ESOD operator to segment the unstructured meshes is also a time-consuming task as [17]. Instead of using the SOD or ESOD operator to extract the features, we use the ESOD operator to find out the *virtual feature edge*. As shown by the dotted line in Figure 4, the edge $e = \{i, j\}$ is called the *virtual edge* of edge $\{k, l\}$ if the edge does not exist, and there exists two faces $\{i, l, k\} \subset F$ and $\{j, k, l\} \subset F$. Furthermore, a virtual edge is called a *virtual feature edge* if its weight calculated by the ESOD operator satisfies:

$$w(e) = \mathbf{n}_{v_k} \cdot \mathbf{n}_{v_l} < \varepsilon.$$

In this case, the edge $\{k, l\}$ is called a *base edge*. Notice that, although we use the same name as in [3], the definition of the base edge is different since we did not consider the detection of features in [3].

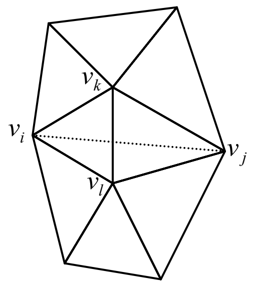


Figure 4. Base edge detection.

4. Mesh simplification and reconstruction

The procedure for our mesh simplification method is listed in Figure 5. The first step is to search for sharp edges. Since the sharp edges are the edges with pre-specified features, they are used to segment the meshes into several parts. Each part of the meshes is simplified independently, and the simplification for the geometric 3D model with sharp edges is described in Section 4.1.

Then, the normal vector of each vertex is calculated, along with the weight of each edge besides the sharp edges. The weight of the edge is assigned the weight of its virtual edge using the ESOD operator, so that by using a threshold $\varepsilon = [-1,1]$, it is possible to detect the base edges of the meshes. As is shown in the example in Figure 4, if the edge $\{i, j\}$ (the dotted line) is a virtual feature edge, the faces above the dotted line and the faces opposite the line could not be merged during the simplification process. Therefore, the endpoints of the base edges, edge $\{k, l\}$ in Figure 4 for instance, are specified as un-removable vertices. Otherwise, the vertices and the edges used to connect them are removable. These removable edges are pushed into a priority queue with their weights being candidate edges.

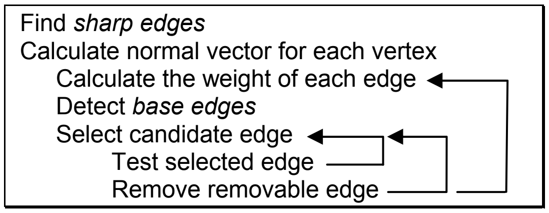


Figure 5. The procedure of our method.

After removing the current removable edges, the weights of the edges which the connectivities have been changed due to the simplification process should be recalculated, and also need to be checked to see if they become base edges or not. Then, attempts are made again to remove the removable edges, as described above, until there are no removable edges.

4.1 Simplification with sharp edge

To simplify a geometric 3D model with pre-defined sharp edges, which are specified when the model is generated, while preserving its features, we first search for the pre-defined sharp edges due to sharp geometric differences and also for the edges which are adjacent to faces which contain different material properties. Then, the endpoints of the sharp edges are marked as un-removable vertices. Therefore, the 3D model is segmented into several parts due to the sharp edges. To simplify each part independently does not make the pre-defined features disappear.

If a vertex belongs to two sharp edges, the vertex is set to be removable with one of the two sharp edges. To remove a vertex which belongs to two sharp edges, the two sharp edges are made into one. Furthermore, to preserve the pre-defined features of the 3D model, when removing the vertex between two sharp edges, we still calculate the weight of these two sharp edges, as described in the previous section. Therefore, only the sharp edges which are not significant are removed.

4.2 Selected removable edge testing

After pushing all of the removable edges into a priority queue, all of the edges in the priority queue are candidates for removal. Before removing the edges, it is necessary to do some tests to check if the removing process passes the following tests.

4.2.1 Preserving mesh inversion. To remove an edge from meshes implies that the neighboring faces on the 1-ring of the endpoints of the edge are deformed. This action may cause the faces to fold over on each other. To avoid this type of mesh inversion, it is necessary to test the edge before removing it. When we get one removable edge from the priority queue, we compare the normal vector of each of the neighboring faces before and after removing. If the normal vector flips, this edge removing is not performed.

4.2.2 Preserving topology. To remove an edge from the meshes also implies that one or two of its adjacent faces are removed. This action may cause the vertex connection of the neighboring faces of the removing edge to change. Since our algorithm is working for 2-manifolds with boundary, it is necessary to test if this edge removing changed the topology of the meshes or not. When we get one removable edge, we check the neighborhood relationship of the neighboring faces of the removing faces before and after the edge removing. If it causes the topology to change, then this edge removing is not allowed.

4.2.3 Preserving model shape. Once the edge obtained from the priority queue passes the above tests, it is necessary to test which vertex could be removed if we used the *half edge collapse* operation to remove the edge, as will be described in Section 4.3. Hence, it is necessary to test which endpoint is the better one to remove when removing one edge. If one of the endpoints is un-removable, i.e. it is also the endpoint of a sharp edge or a base edge, we remove the other one. If both of the endpoints of the edge are removable, we compare the deviation of the neighboring faces' normal vectors which are on the 1-ring of the endpoints, and then we remove the vertex which is located on the faces that are flatter than the ones located by the other one.

If the removable edge has passed all of the tests, then the half edge collapse operation will be operated as described in the following section.

4.3 Half edge collapse

The *half edge collapse* operation, as shown in Figure 6, is a special case of the *edge collapse* operation. If a vertex removed with a particular re-triangulation of the remaining hole when using the *vertex decimation* operation, the resulting mesh is also the same as the one after doing the half edge collapse operation.

The information used for reconstructing the original model is stored as a *patch*. To minimize the data size of the patch which will be sent to the client side to reconstruct the original model and significantly affects the network transmission, we use the half edge collapse operation instead of using the edge collapse operation. Using the half edge collapse operation reduces the size of the patch compared to the edge collapse operation. This is because, when using the half edge collapse operation, there is only one vertex removed. No vertex is added into the meshes but the edge collapse operation removes two vertices and adds one vertex.

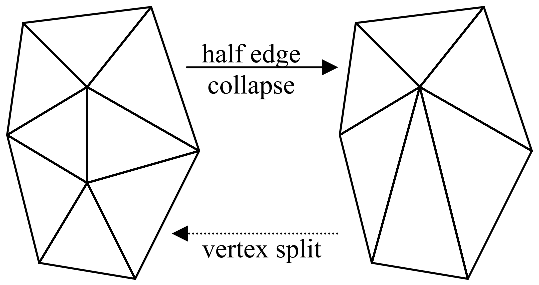


Figure 6. Half edge collapse and vertex split.

```
struct patch {
  int flclw;                  // a face in neighborhood of the patch
  struct {
    short vlr_rot:6;          // encoding to find another vertex
    short vs_index:2;         // index (0..2) within the patch
  } code;
  VertexAttribD vad_l;
  WedgeAttribD wad_l;
};
```

Figure 7. Patch data structure.

Compared with the vertex split (Vsplit) data structure described in [13], since we detect the sharp edges before doing the mesh simplification process, the size of our patch data structure listed in Figure 7 is only half of the size of the Vsplit data structure, even when we use the edge collapse operation. This is because by using the half edge collapse operation, we still could get as good a simplified model as the one generated using the edge collapse operation. Therefore, in our algorithm, we still use the half edge collapse operation to minimize the size of the patch, although the difference in the patch size is only a few as a result of using the two operations. Since the size of the patch used in our method is smaller than other similar data structure used in other ones, the data size transmitted over the Internet is therefore also smaller.

4.4 Mesh reconstruction

The simplified mesh and the patches which have been stored when doing the mesh simplification process are uploaded onto the Web server and provided for the users to use at the client site. It is named the *streaming mesh*. When using the streaming mesh on the Internet, the simplified mesh is sent first. After sending the simplified mesh, some patches are sent progressively with QoS-like controlling as described in [3], so that the original 3D model may be reconstructed without loss of data using the *vertex split* operation, as shown in Figure 6.

5. Result

Figures 10 (b) ~ (d) show the simplified models, which are generated with different thresholds ε, of the bunny model shown in Figure 10 (a). Figure 10 (e) shows the simplified model when we ignore the base edge detection, since there are some boundary edges at the bottom of the model, which will be detected as sharp edges. The shape of this part remains recognizable. Without detecting the base edges of the 3D model, the model may be over-simplified. As a result, the shape of the simplified model could not be recognized. This is a common problem of other previous methods. Figure 11 shows comparisons of the original models and simplified results of other geometric 3D models. Even for the simplified models, the shape and features could still be recognized easily. Since there are several sharp edges contained in the models shown in Figures 11 (c) and (d), we ignore the base edge detection during the simplification process. The number of faces of each model and the thresholds used for simplification are also shown in Figures 10 and 11.

Table 1. Comparison of file sizes & performances.

model	original model (bytes)	simplified model (bytes)	one patch (bytes)	time (ms)
bunny	75,142	5,746	34.424	1,141
dragon	65,586	18,137	35.939	961
hand	53,519	3,991	35.684	951
cessna	553,685	148,320	30.379	6,219
fandisk	295,092	35,765	29.083	15,102
tiger	347,587	9,099	34.925	15,852

Table 1 lists the file sizes of the original models and the simplified ones of different geometric 3D models shown in Figures 10 and 11, respectively. The simplified bunny model used in Table 1 is the model shown in Figure 10 (d). The run-time performances required to generate the simplified models and the average patch size are also shown in Table 1. The testing platform is a notebook PC with an Intel Mobile Pentium III 850MHz CPU and 256MB memory, the Java environment is Sun Java2 SDK,

SDK, Standard Edition v1.4.1_01. Since we wish to generate a simplified model with shapes and features that are easily recognized, the compression rate of the model with several pre-defined features is worse than for other models, for example the model shown in Figure 11 (c).

For comparison, we have converted the file format of the original model to be the same as the simplified mesh (a gzipped ASCII file). The number of patches for reconstructing the original model from the simplified one is the difference of the vertex number of the original model and the simplified one. All of the patches are also stored as a gzipped ASCII file. Furthermore, since we use only pure Java programming language to develop all of the algorithms, it is possible to use our testing program via our Web site[2]. Moreover, the 3D graphics engine - jGL[3] is used which is also developed with pure Java.

The model shown in Figure 11 (e) is generated from the model shown in Figure 12 (a) by applying the $\sqrt{3}$-subdivision algorithm provided by Kobbelt [17] three times. The model shown in Figure 12 (c) is the model generated by applying the same subdivision algorithm only once. If the models reconstructed from the simplified model shown in Figure 11 (j) have the same number of faces as the models shown in Figures 12 (a) and (c), the results shown in Figures 12 (b) and (d) are similar. These comparisons show that our algorithm could result in a well-reconstructed model even there are only a few patches applied to the simplified model.

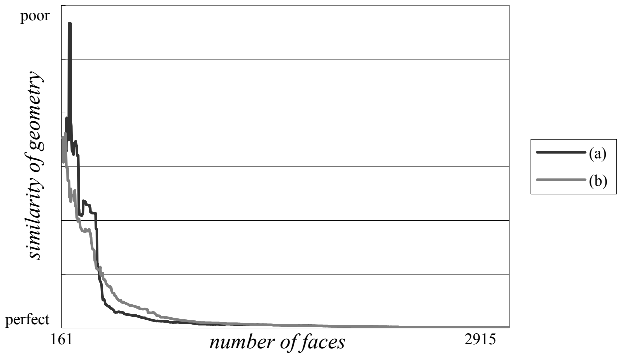


Figure 8. Similarity of geometric approximation.

For comparing the models reconstructed from the simplified model with the original one, we use the simplified model shown in Figure 10 (d), which is generated by our method with $\varepsilon = -0.2$, and its original model is shown in Figure 10 (a). The results are as the curves (a) shown in Figures 8 and 9. In order to compare with other previous methods, we also used the QEM method to simplify the bunny model shown in Figure 10 (a), and made the same comparisons as those of our approach. The results are as the curves (b) shown in Figures 8 and 9.

Figure 8 shows the similarity of the geometrical approximation of each reconstructed model and the original one. To emphasize the difference of the two models in each comparison, the measurement used in Figure 8 is based on the L_2 norm: the average squared distance from the vertices of the original model to the surface of the reconstructed one used to compare the difference between the two models. Figure 9 shows the comparisons for similarity in appearance of each of the reconstructed models and the original models. To compare the similarity in appearance, we compute the differences between the rendered images of the original model and the reconstructed ones by setting the camera at 6 different positions as in [19].

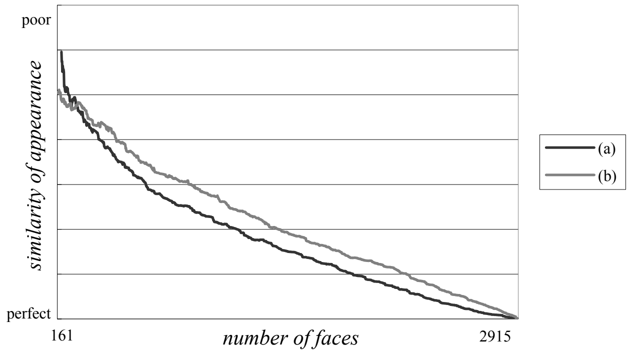


Figure 9. Similarity of appearance.

Using the QEM method, we could get a simplified model that contains less faces (number of faces is 161 in this case) than our approach, but the qualities of the models reconstructed from the simplified one are almost the same. However, to generate the simplified model using the QEM method is much more time-consuming than using our method. For example, simplifying the bunny model shown in Figure 10 (a) took more than 4.5 seconds, but took less than 1.2 seconds using our approach, as shown in Table 1. Moreover, the data size of the patch used for reconstructing the original model in our method is much smaller than other previous methods.

6. Conclusions

In this paper, a new, simple, and efficient mesh simplification method for unstructured meshes and Web Graphics is presented. Although our approach is not able to generate an almost optimized simplified model, to make the shape and features of the simplified model still recognizable is the main purpose of our method. However, a simplified model generated by some previous methods sometimes becomes only a polyhedron. This is useless in prac-

[2] *http://nis-lab.is.s.u-tokyo.ac.jp/~robin/jMS/*
[3] *http://nis-lab.is.s.u-tokyo.ac.jp/~robin/jGL/*

tice. Since our method could provide the simplified model of fewer than 20KB on average, this kind of small size could be transmitted efficiently through the Internet.

Moreover, the size of the patch used for reconstructing the original model is less than that used in other previous methods. For example, when using the data structure as the PM method, the data size is twice as large as our patch size, if both of them are stored as binary files. Therefore, the client site could receive more mesh data from the server by using our approach compared to other methods. In practice, by using our previous QoS-like transmission method, the user using our system could get a better model than with other systems, since he or she could receive more mesh data if the transmission time is the same. Finally, if the user at the client site really needs the original 3D model, after receiving all of the patches, he or she could still get the lossless original model without any redundant data transmission.

Additionally, since our method is efficient, the model provider can change the threshold interactively to get a proper simplified model with the appropriate data size so that the provider can guarantee what resolution of simplified model will be transmitted to the users.

7. Acknowledgements

The authors would like to thank Yutaka Ono, who prepared several 3D models for this paper.

8. References

[1] P. Alliez and M. Desbrun, "Progressive Compression for Lossless Transmission of Triangle Meshes", *ACM SIGGRAPH 2001 Conference Proceedings*, 2001, pp. 195-202.

[2] B.-Y. Chen and T. Nishita, "jGL and Its Applications as a Web3D Platform", *ACM Web3D 2001 Conference Proceedings*, 2001, pp. 85-91.

[3] B.-Y. Chen and T. Nishita, "Multiresolution Streaming Mesh with Shape Preserving and QoS-like Controlling", *ACM Web3D 2002 Conference Proceedings*, 2002, pp. 35-42.

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

[5] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes", *ACM SIGGRAPH 95 Conference Proceedings*, 1995, pp. 173-182.

[6] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.

[7] M. Garland. "Multiresolution Modeling: Survey & Future Opportunities", *Eurographics 99 State of the Art Report*, 1999.

[8] M. Garland and P. S. Heckbert, "Surface Simplification Using Quadric Error Metrics", *ACM SIGGRAPH 97 Conference Proceedings*, 1997, pp. 209-216.

[9] M. Garland and P. S. Heckbert, "Simplifying Surfaces with Color and Texture Using Quadric Error Metrics", *IEEE Visualization 98 Conference Proceedings*, 1998, pp. 263-269.

[10] M. Garland, A. Willmott, and P. S. Heckbert, "Hierarchical Face Clustering on Polygonal Surfaces", *ACM Interactive 3D Graphics 2001 Conference Proceedings*, 2001, pp. 49-58.

[11] P. S. Heckbert and M. Garland, "Survey of Polygonal Surface Simplification Algorithms", *Multiresolution Surface Modeling (ACM SIGGRAPH 97 Course Notes #25)*, 1997.

[12] H. Hoppe, "Progressive Meshes", *ACM SIGGRAPH 96 Conference Proceedings*, 1996, pp. 99-108.

[13] H. Hoppe, "Efficient Implementation of Progressive Meshes", *Computer & Graphics*, Vol. 22, No. 1, 1998, pp. 27-36.

[14] H. Hoppe, "New Quadric Metric for Simplifying Meshes with Appearance Attributes", *IEEE Visualization 99 Conference Proceedings*, 1999, pp. 59-66.

[15] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization", *ACM SIGGRAPH 93 Conference Proceedings*, 1993, pp. 19-26.

[16] A. Hubeli and M. Gross, "Multiresolution Feature Extraction from Unstructured Meshes", *IEEE Visualization 2001 Conference Proceedings*, 2001, pp. 287-294.

[17] L. Kobbelt, "$\sqrt{3}$-subdivision", *ACM SIGGRAPH 2000 Conference Proceedings*, 2000, pp. 103-112.

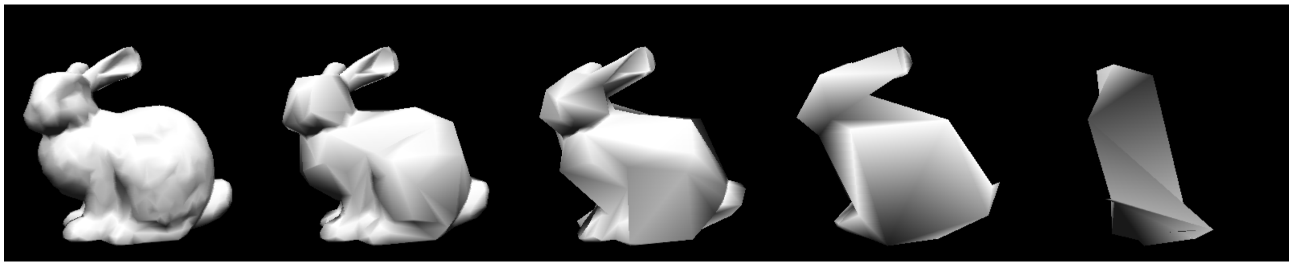
[18] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least Squares Conformal Maps for Automatic Texture Atlas Generation", *ACM Transactions on Graphics (SIGGRAPH 2002 Conference Proceedings)*, Vol. 21, No. 3, 2002, pp. 362-371.

[19] P. Lindstrom and G. Turk, "Image-driven Simplification", *ACM Transactions on Graphics*, Vol. 19, No. 3, 2000, pp. 204-241.

[20] D. P. Luebke, "A Developer's Survey of Polygonal Simplification Algorithms", *IEEE Computer Graphics and Applications*, Vol. 21, No. 3, 2001, pp. 24-35.

[21] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of Triangle Meshes", *ACM Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, Vol. 26, No. 2, 1992, pp. 65-70.

[22] G. Turk, "Re-tiling Polygonal Surfaces", *ACM Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, Vol. 26, No. 2, 1992, pp. 55-64.



#f = 2915 (a) original model

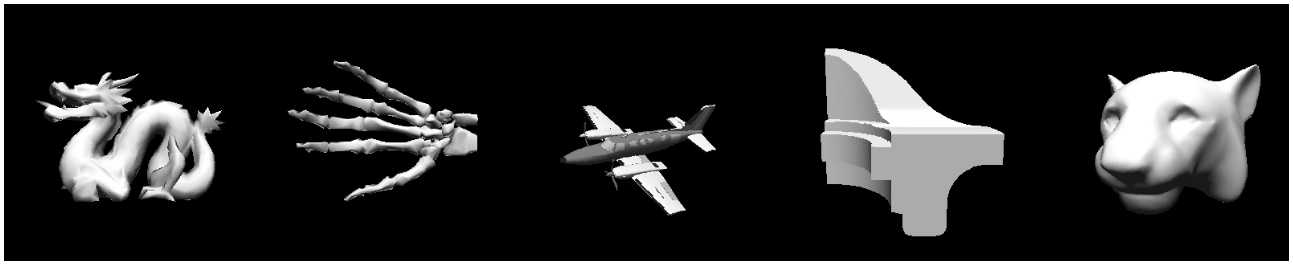
ε = 0.8: #f = 1145 (b)

ε = 0.5: #f = 487 (c)

ε = -0.2: #f = 185 (d)

#f = 103 (e) no base edge

Figure 10: Comparisons of (a) original bunny model, (b) ~ (d) simplified models with different thresholds, and (e) simplified model without base edge detection.



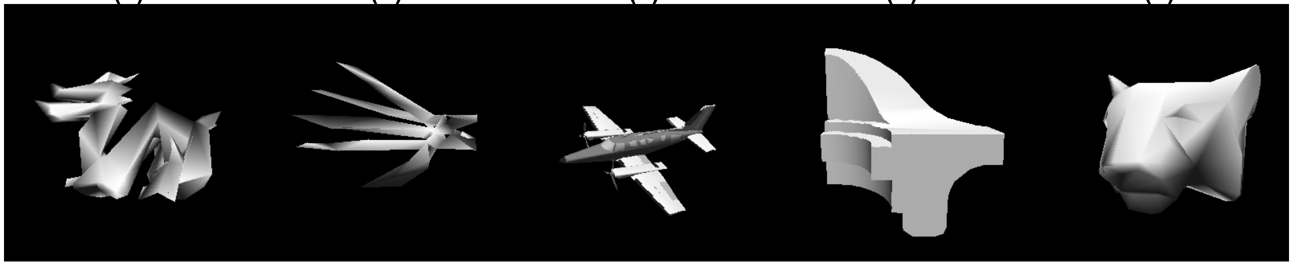
#f = 2730 (a)

#f = 2130 (b)

#f = 13546 (c)

#f = 12946 (d)

#f = 13608 (e)



ε = -0.3: #f = 892 (f) dragon

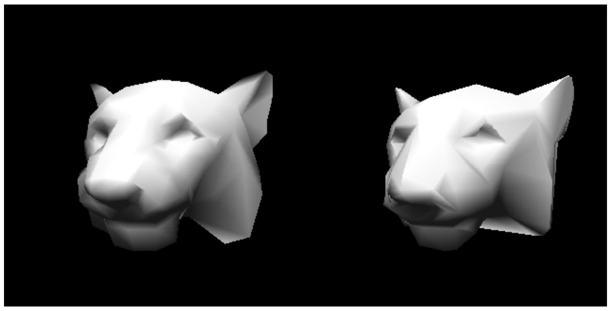
ε = -0.9: #f = 182 (g) hand

#f = 3838 (h) cessna - with materials

#f = 772 (i) fandisk - with sharp edges

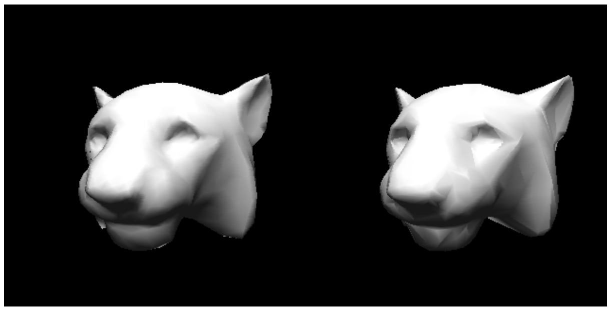
ε = 0.7: #f = 376 (j) tiger

Figure 11. Comparisons of (a) ~ (e) original models, and (f) ~ (j) simplified results.



#f = 504 (a) original model

#f = 504 (b)



#f = 1512 (c) subdivided from (a)

#f = 1512 (d)

Figure 12. Comparisons of (a) original model, (c) subdivided model from (a), and (b) (d) reconstructed models from Figure 11 (j).