Multiresolution Streaming Mesh with Shape Preserving and QoS-like Controlling

Bing-Yu Chen
robin@is.s.u-tokyo.ac.jp

Tomoyuki Nishita
nis@is.s.u-tokyo.ac.jp

Department of Information Science, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan

ABSTRACT

How to transmit 3D meshes efficiently has become an important topic on Web3D platform, since there are more and more people need to use 3D models on the Internet. The data size of a geometric 3D model is usually large for being able to represent more details of the model, although we do not need to use such a detail model in most cases. Hence, to offer 3D model which shape and features could still be recognized easily with less data size is necessary. Additionally, the network bandwidth of the Internet is not stable actually, how much data is suitable for Internet is also a question. Therefore, we propose a new multiresolution streaming mesh for Internet transmission with QoS-like (Quality of Service) controlling in this paper.

While transmitting the streaming mesh with our system, the server first delivers a simplified mesh model with the data size according to the current network bandwidth. If the user at the client side needs to use a more detail model, the server then sends some necessary patches to the client, so that the client program could show the detail model progressively. Our approach is different from previous works, for Web3D utilization, the size of the patch data which is used for reconstructing the original 3D model is less, and the shape and features of the simplified model could still be recognized easily. Moreover, our method needs no complex computations, to generate this streaming mesh on demand is possible. With the QoS-like controlling, the transmission rate between the server and the client has been controlled automatically and the users could get the 3D models with proper qualities as their network situations.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Client/server*; I.3.2 [**Computer Graphics**]: Graphics Systems – *Distributed/network graphics*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – *Curve, surface, solid, and object representations*.

Keywords

Streaming Mesh, Mesh Simplification, Level of Detail, Geometric Quality of Service.

Web3D'02, February 24-26, 2002, Tempe, Arizona, USA.

1. INTRODUCTION

Recently, more and more users on the Internet want to have the supports from 3D graphics, since the machine performance and network bandwidth are getting better than before. In the Internet Graphics field, the utilization of geometric 3D models is much usual, so that how to transmit 3D mesh data through the Internet efficiently has become an important topic on Web3D platform, since the data size of a 3D model is usually large. If a user wants to download a 3D model through the Internet, he or she must pay much time to wait for getting the whole data, even if the model is not really he or she wants after a long time waiting. Moreover, the users on the Internet usually do not need to use such a detail model in most cases, sometimes they just download the model to check if it is what they need. Hence, to offer a simplified mesh model which shape and features could still be recognized easily is necessary. Moreover, since the network bandwidth of the Internet is not stable actually, how much data is suitable to represent a 3D model for Internet uses is also a question.

Therefore, we develop a new multiresolution streaming mesh for Internet transmission with QoS-like controlling in this paper. When a user needs to use a 3D model encoded with the streaming mesh format on the Internet the server first delivers a simplified mesh model which shape and features could still be recognized easily to the user with the data size according to the QoS-like controlling. QoS [3] is a protocol of computer network technologies, which provides the mechanics to differentiate traffic, so that users can get different quality of video or audio due to different bandwidth of the network environment with the same continuity. In this paper, we use the same concepts to provide the 3D models with different resolutions to the users at the client side, and the users will pay the same waiting time to get the models with different resolutions. Since this is not the definition of the original QoS, we call this approach as "QoS-like" in this paper.

Then, if the user needs to use the model with more details, the server will then transmit some necessary patches to the client also with the QoS-like controlling, so that the client program could show the detail model progressively, and the user also gets different progressive patches due to the current network bandwidth. Finally, if the user really needs the original model, after receiving all the patches, the system then reconstructs the original 3D model with no loss and no retransmission. The patch here means the packaged information which is used to reconstruct the original model.

There are three processes to construct the streaming mesh. The first process is the 3D mesh simplification, the second one is the storage methodology for the simplified mesh and the patches which will be used to reconstruct the original model, and the last one is to transmit the streaming mesh with QoS-like controlling and reconstruct the original 3D model. The mesh simplification algorithm of our streaming mesh is first categorizing all vertices into n levels, and finding the connective relationships between the vertices of each level. By using several *half edge collapse* operations, which will be described in Section 4.2, the original 3D model is processed to be a simplified 3D model. Furthermore, the vertices of the simplified model are the subset of the vertices of the original model, i.e. the vertex positions are the same, and the connective relationships between the vertices are also the same. Hence, the data size of the patches used to reconstruct the original model is less.

Moreover, to use a simple 3D model is more usual than to use a huge model, so we preserve the shape of the 3D model during the 3D mesh simplification process. Furthermore, since our approach does not need any complex computation, to generate the streaming mesh on the fly is possible. With the QoS-like controlling, the transmission rate between the server and the client has been controlled automatically and the users could get the 3D models with proper qualities as their network bandwidth.

Observing the development of the Internet, we believe that "pay-per-use" software will be realized in the near future. Under this new paradigm, we may need to distribute applications from servers to clients on different platforms. Therefore, we decided to develop all the algorithms of the streaming mesh by using pure Java [1] programming language for its hardware-neutral features, and wide availability on many hardware platforms, even for embedded systems, such as mobile phones or PDAs (personal data assistant). Moreover, the 3D graphics rendering is done by jGL [4] which is a 3D graphics library for Java with OpenGL-like API (application-programming interface) and also developed by us.

2. RELATED RESEARCHES

There are several researches about transmitting 3D models on the Internet. Some of them are transmitting the compressed 3D model which provides fine shape with few data size. Others are transmitting the simplified 3D model first, and then using the transmitted simplified model to reconstruct the lossless original 3D model progressively. Our approach is the latter one and also needs a 3D mesh simplification process. There are several kinds of 3D mesh simplification methods [10], but not all of them are satisfied to reconstruct the original 3D model. In this section, we would like to introduce some suitable ones.

PM (Progressive Meshes) is the most famous method for 3D mesh simplification based on *edge collapse* or *edge contraction* operation. This algorithm is provided by Sander at el. [20], Hoppe [13] [14] [15] and Hoppe at el. [16]. Simply speaking, this 3D mesh simplification method is to find the minimum value of an energy function which includes the vertex positions, vertex connectivity, normal vectors, and texture mapping coordinates. A derived method, QEM (Quadric Error Metrics), has been provided by Hoppe [12], Garland at el. [8] [9] and Ronfard at el. [19], to make the calculation faster by calculating the error quadrics of new vertices.

The main process of PM is the *edge collapse* operation, and the energy function is minimized by using a nested optimization method. The outer loop is used for optimizing over the topology of the 3D mesh, and the inner loop is used for optimizing over the vertex positions. Hence, the vertex connectivity of the simplified model is the same with that of the original model, but the vertex positions will maybe be changed.

Another famous algorithm is based on the *vertex decimation* operation, which is provided by Alliez at el. [2], Lee at el. [18], Turk [23] and Schroeder at el. [22]. This algorithm is different from the PM method; there is no any energy function which maybe needs complex computations. But, the shape of the simplified 3D model will maybe be changed and hardly recognized.

The main processes of these *vertex decimation* algorithms are finding the removable vertices, and using the *vertex removal* operations to remove all the removable vertices which are independent with each other. Two vertices are called independent if they are not adjacent. Finally, re-triangulating the remaining holes left by removing the vertices. Hence, the vertices of the simplified model are the subset of vertices of the original model, i.e. the vertex positions are the same, but the vertex connectivity will maybe be changed.

3. NOTATION

A geometric 3D model is usually represented as triangular meshes[1]. Each triangular mesh is associated by three vertices. Like other 3D model representations, a geometric 3D model M could be represented as the formulas in Figure 1. In the formulas, V is the set of vertex positions v_i, $i \in [1, m]$, where m is the number of vertices, defining the shape of the triangular meshes. F means the vertex connectivity of the 3D model. D is the set of discrete attributes d_f, like the materials, associated with the faces f, and S is the set of scalar attributes $s_{(v_i, f)}$, like the normal vectors, associated with the wedges $w = (v_i, f)$. Hence, the geometry of the 3D meshes could be represented as the image $f_V(|F|)$, where $f_V : \Re^m \to \Re^3$ is a linear mapping.

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

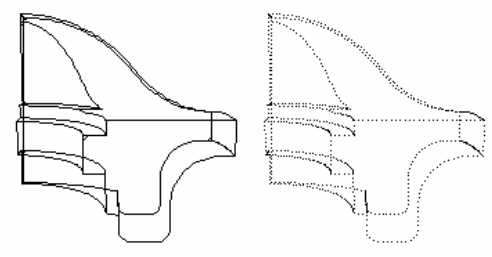
Figure 1: The representation of a geometric 3D mesh.

An edge $e = \{j,k\}$ is called a *boundary edge* if there is only one face $f = \{j,k,l\}$ with $e \subset f$. An edge $\{j,k\}$ is called a *sharp edge*[2] 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. $s_{(v_j, f_l)} \neq s_{(v_j, f_r)}$ or $s_{(v_k, f_l)} \neq s_{(v_k, f_r)}$. If an edge $\{j,k\}$ is a sharp edge,

[1] We convert all non-triangular meshes into triangular ones before doing the 3D mesh simplification process in this paper.

[2] The "sharp" edge here is not only used to indicate the sharp of the geometric differences, but also for the edges which adjacent faces contain different materials.

the two endpoints of the edge $\{j\}$ and $\{k\}$ will be called *corner vertices*[3]. Figure 2 is the examples for showing the (a) sharp edges and (b) corner vertices.



(a) **(b)**

Figure 2: The examples of (a) sharp edges and (b) corner vertices.

An edge $\{j,k\}$ is called a *base edge* of level i, $i \in [1,n]$, where n denotes the number of levels from the original model to the simplified model, if its adjacent wedges have the same scalar attributes, i.e. $s_{(v_j,f_l)} = s_{(v_j,f_r)}$ and $s_{(v_k,f_l)} = s_{(v_k,f_r)}$, and the weight of this edge is bigger than the threshold e_i of level i, where the difference of the two endpoints' scalar attributes is called the *weight* of the edge. Figures 9 (a) ~ (d) show the examples of the base edges in different levels. The vertices set V_i, which contains all endpoints of the base edges of level i, corner vertices, and some necessary points used to connect all the above vertices are called *base vertices* of level i. Therefore, mesh M^i is called a simplified mesh of level i if it is associated from the vertices of the set V_i, where $\{V_i\}_{i=1}^{n}$, $V_i \subset V_{i+1}$, $V_n = V$.

Moreover, if the difference between the vertex number of V and V_1 is n', there are n' steps from the original mesh to simplified mesh, and each step needs one *half edge collapse* operation. On the other hand, when reconstructing from the simplified mesh to the original mesh, n' times of the *vertex split* operations are needed. Notice that the number of levels n and the number of steps n' are different. If the steps operated in one level i is i_i, then the summation is $\sum_{i=1}^{n} i_i = n'$.

4. MESH SIMPLIFICATION

The first process of 3D mesh simplification is finding the sharp edges. Then, using the threshold e_{n-1} to find the base edges of level $n-1$. The endpoints of the sharp edges and base edges are specified as the un-removable vertices. Otherwise, the other vertices and the edges used to connect them are removable. By removing the removable vertices and edges after some necessary tests, the first simplified mesh of level $n-1$ is done.

Next, use the same threshold to search the base edges again. If there are still some removable vertices and edges, removing these vertices and edges as above to get the next simplified mesh of the same level. Repeat the loop until there is no removable vertex and edge; the final simplified mesh of level $n-1$ is got. Then, use this simplified mesh to generate the simplified mesh of the next level by using the same loop as above. Once we got to the level 1, the procedure of 3D mesh simplification is completed.

[3] The set of the corner vertices V_C is a subset of the set of all the vertices V (i.e. $V_C \subseteq V$).

Additionally, the number of levels, which is the ending condition, could be set according to simplified level of the 3D model. If there is no such ending condition, the simplification process will be stopped when there is no removable vertex and edge.

4.1 Removable Vertex Selection

To find the removable vertices and edges, we find the sharp edges and base edges of current level first, and mark all the endpoints of the sharp edges and base edges as the un-removable vertices, and mark the sharp edges and base edges as the un-removable edges also. As shown in Figure 3, if there are more than one removable edges linked with the removable vertex located in the center, to test which edge removed is better is necessary. For this reason, we use a *priority queue* and put removable edges with its weight as its priority into it, which has been described in Section 3 and has been calculated during finding the base edges. Therefore, the edge with smaller weight will be removed earlier.

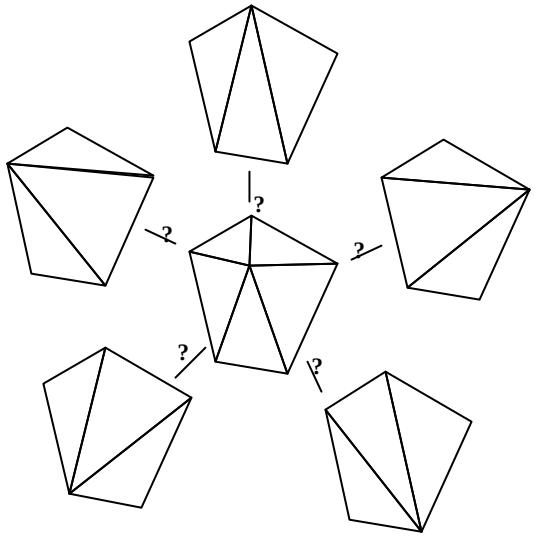


Figure 3: When removing the vertex located in the center, one of its adjacent edges will also be removed. Here we choose the edge with a lower *weight*. In this case, we will choose the upper arrow.

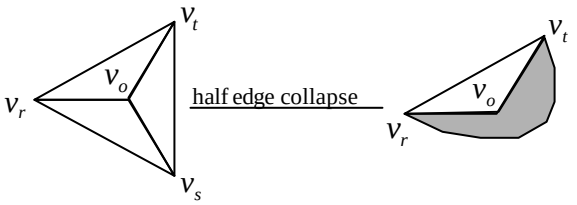


Figure 4: After collapsing the half edge $\overline{v_t v_s}$, i.e. combining the vertex v_s with vertex v_t, the vertex v_s is removed, and the triangle $\Delta v_o v_s v_r$ has been deformed to be $\Delta v_o v_t v_r$, which is the reverse triangle of triangle $\Delta v_o v_r v_t$. Then, the flipping error is occurred.

Then, get one removable edge from the priority queue and try to combine one endpoint of the edge to the other. If there will be a flipping error as shown in Figure 4 after deforming some faces due to the half edge collapse operation, this operation will be given up.

If the removable edge has been passed all tests, the half edge collapse will be operated as the following section.

4.2 Half Edge Collapse

The "*half edge collapse*" operation is a special case of the "*edge collapse*" operation, and if a vertex removed with particular re-triangulation of the remaining hole, the resulting mesh is also the same as the one after doing the "*half edge collapse*" operation. If we wish to get a simplified mesh with good compressing rate, the number of patches must be large. Therefore, we use the "*half edge collapse*" operation in this paper for minimizing the data size of each patch which effects the network transmission significantly

Before doing the half edge collapse operation, it is necessary to store the information of the removing vertex for reconstructing the original model. Then, collapse the edge, remove the vertex, and deform the faces associated with the removed vertex as shown in Figure 5. In this figure, the triangles located in the bottom of the removed vertex v_s is deformed after collapsing the half edge $\overline{v_t v_s}$, i.e. combining the vertex v_s with vertex v_t, and the vertex v_s, face $\Delta v_t v_a v_s$, and face $\Delta v_t v_s v_t$ are removed.

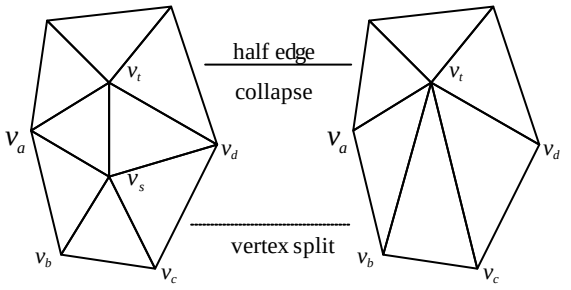


Figure 5: The examples of (a) half edge collapse and (b) vertex split operations.

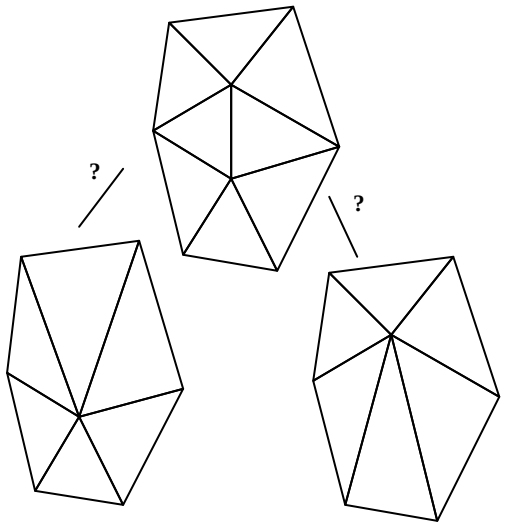


Figure 6: When removing the edge located in the center, one of its endpoints will be removed also. Here we removed the one due to the *deviation* of its adjacent faces' normal vectors.

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, we use half edge collapse instead of using normal edge collapse. Hence, when removing one edge, to test which endpoint is better one to be removed is necessary as shown in Figure 6. Therefore, the policy which we used here is if either (1) vertex v_t is an un-removable vertex, or (2) the *deviation* of adjacent faces' normal vectors of vertex v_t is greater than that of vertex v_s, the vertex v_s is removed.

5. MESH RECONSTRUCTION

The simplified mesh and the patches which have been stored when doing the 3D mesh simplification process are the components of the streaming mesh. When using the streaming mesh on the Internet, the simplified mesh is sent first. After sending the simplified mesh, the patches is sent progressively with QoS-like controlling; then the original 3D model could be reconstructed without data lost by using the vertex slip operation.

5.1 Vertex Split

If there are n' steps for reconstructing the original mesh from the simplified mesh, n' patches are needed. Each of the patches contains the geometric position and attributes of one removed vertex in the simplification process and other necessary information which will be explained in this section. As described in Section 4.2, when doing the 3D mesh simplification from the mesh of step $i'+1$ to the mesh of step i', the half edge collapse operation has been used. When doing the mesh reconstruction process from the mesh of step i' to the mesh of step $i'+1$, we use the vertex split operation on the other hand.

Besides the position and attributes of the vertex which will be added into the model, the other necessary information for the vertex split operation is used for finding the vertex which will be split. As shown in Figure 7, the vertex v_t is such a vertex which will be split, and the triangles bellowed the edge $\overline{v_a v_t}$ and edge $\overline{v_t v_d}$ will be deformed. Hence, each patch also contains the triangle index of f_l, the vertex index of vertex v_t in face f_l, and the number of triangles from the face f_l to face f_r. Therefore, we can use the triangle index of f_l and the vertex index to find out the vertex v_t, which will be split, and use the neighborhood information to find out all the triangles from face f_l to face f_r, which will be deformed after the vertex split operation.

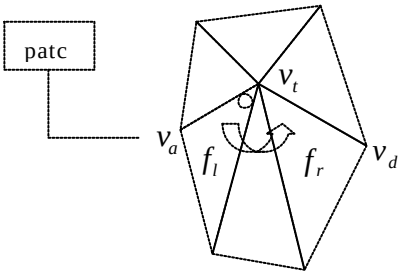


Figure 7: Using the transmitted patch, the vertex which will be split is found. Moreover, the triangles which will be deformed due to the vertex split operation will also be decided by using the same patch.

When the mesh of step i' becomes the mesh of step $i'+1$, we first find out the vertex which will be split by using the transmitted patch. As shown in Figure 5, we use the information contained in the transmitted patch to find the vertex v_a, vertex v_d,

and vertex v_t, then add a new vertex v_s and edge $\overline{v_s v_t}$ to the meshes, finally deform the triangles located in the bottom of edge $\overline{v_a v_s}$ and edge $\overline{v_s v_d}$.

5.2 QoS-like Controlling

Although we have decoded a geometric 3D model into a streaming mesh which contains one simplified mesh and several patches. To transmit the streaming mesh through the Internet efficiently is still a problem, since the real network bandwidth between the server and the client is unknown and unstable. If the server delivers all the patches in the same time, the users at the client side must still wait for downloading them. On the other hand, if the server sends only one patch at one time, the overhead of the network package's header and the synchronization between the server and the client will make the transmission rate worse [7] as shown in Table 3. Hence, to make a flow control for monitoring the patches' transmissions is necessary. In our experiment, we use HTTP (Hypertext Transfer Protocol) as the transmission protocol, since it could pass trough almost all the firewall limitation, although the synchronization between the server and the client costs a lot of time when making the connection.

A useful and interesting concept of QoS is to provide different quality of medias over the flexible network bandwidth. Here, we use the same concept to provide the 3D models with different resolutions to the users at the client side, and the users will pay the same waiting time to get different progressive models.

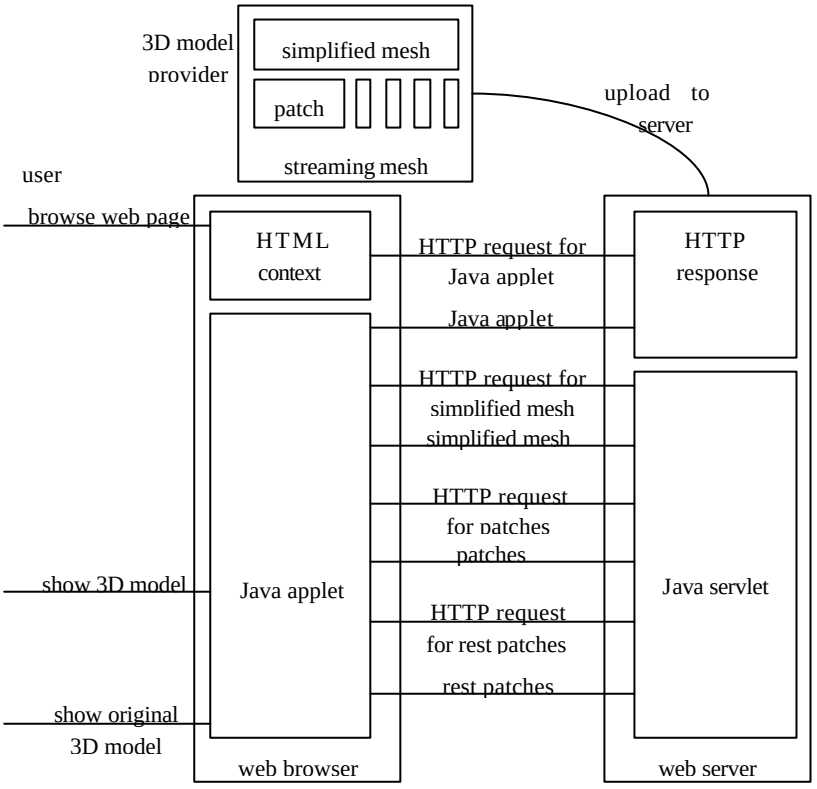


Figure 8: System hierarchy and network communication diagram.

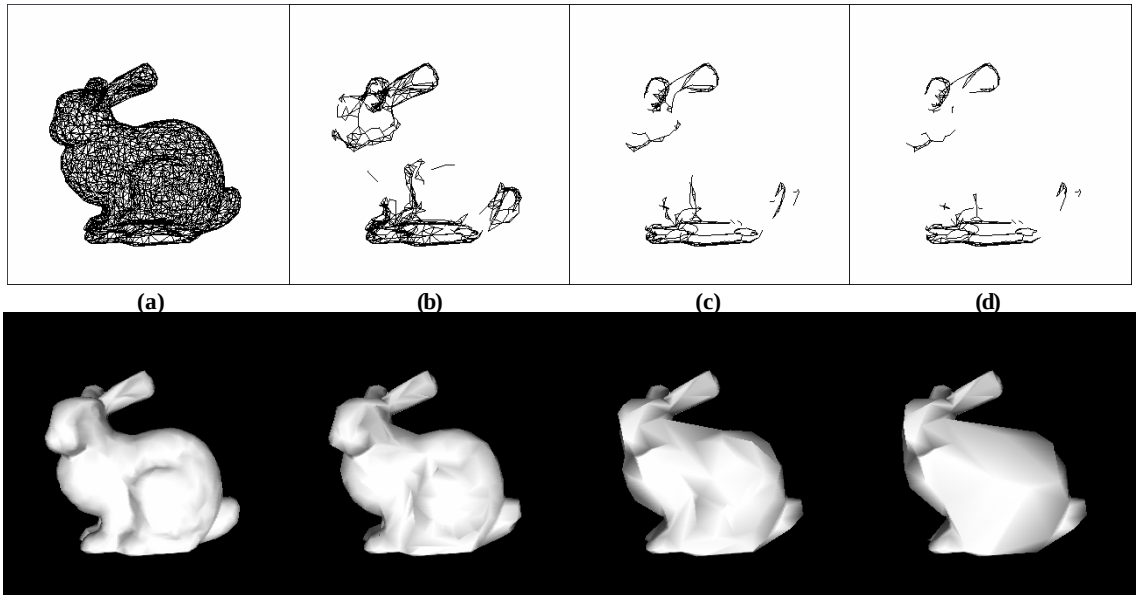
The system hierarchy and the network communication diagram are shown in Figure 8. When the 3D model provider creates a model encoded with streaming mesh method which contains a simplified mesh and several patches, all he or she has to do is to upload it onto the web server and the users will use it via the inline applet by a web browser. The web browser first sends a HTTP request to the web server to download the Java applet which is the tool to display the streaming mesh. After the applet is running on the client machine, the applet will then send a HTTP request to the web server for downloading the simplified mesh which is the first part of the streaming mesh, and calculate the effective bandwidth between the server and the client by using the downloaded file size and the transmission time. Then, the applet will make a HTTP connection again with a Java servlet on the web server, and download the proper number of patches according to the calculated network bandwidth, and the bandwidth will be recalculated again by using the downloading file size, transmission time, and the previous network bandwidth. Hence, the client applet could show the 3D model with proper data size due to the current network bandwidth.

Then, if the user needs to use the model with more details, the client applet will send a HTTP request again to the server, and the server servlet will transmit the patches to the client also according to the calculated network bandwidth, so that the client applet could show the detail model progressively and recalculate the network bandwidth again. Finally, if the user really needs the original model, after the server servlet will send all the rest patches to the client, and then the client applet could reconstructs the original 3D model with no loss and no retransmission.

Additionally, the Java servlet is coded by using Sun Java™2 SDK, Enterprise Edition v1.2.1.

6. RESULTS

Figure 9 shows the base edges and simplified models from the level of the original model to the final level of the simplified model of a 3D model "bunny" in different levels. Figure 10 shows the comparisons of the original and simplified models of other 3D models, even for the simplified 3D model, the shape and features could still be recognized easily. Table 1 lists the face number and vertex number of the original and simplified models and also the performance to generate the simplified models, which includes the time for simplifying the original model and compressing the resulting ASCII file, for different 3D models. The testing platform is a notebook PC with Intel Mobile Pentium III 850MHz CPU and 256MB memory, the Java environment is Sun Java™2 SDK, Standard Edition v1.3.1. Since we wish to generate the simplified model which shape and features could still be recognized easily, the compress rate of the model with several features is worse than other models, like the 3D model "hand".

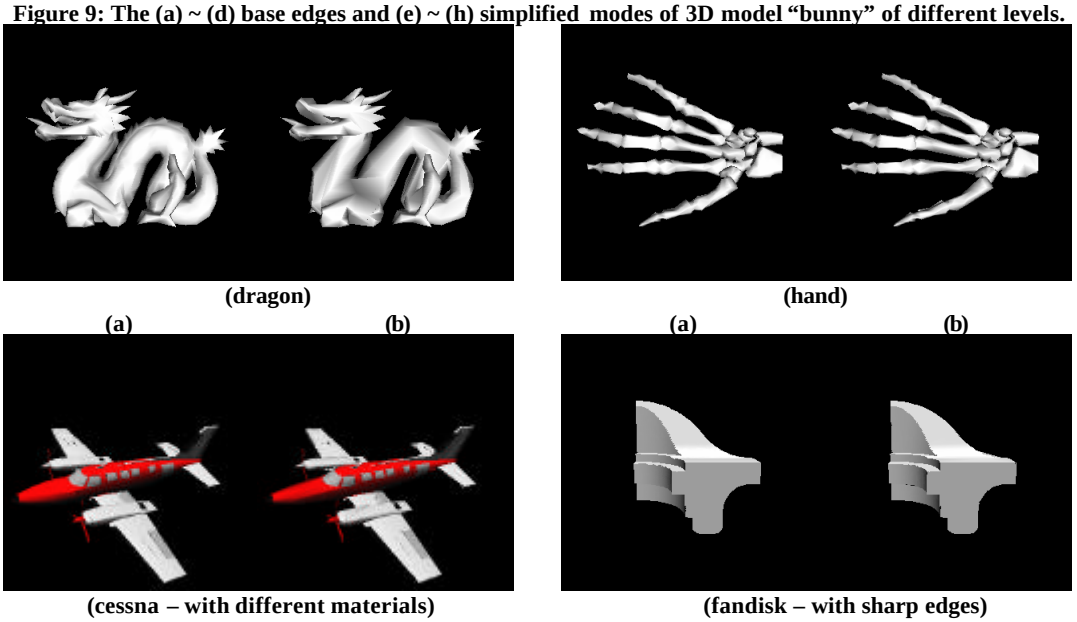


(a) (b) (c) (d)

(e) (f) (g) (h)

Level 17: f = 2915 Level 13: f = 1145 Level 9: f = 669 Level 1: f = 531

Figure 9: The (a) ~ (d) base edges and (e) ~ (h) simplified modes of 3D model "bunny" of different levels.



(dragon) **(hand)**

(a) (b) (a) (b)

(cessna – with different materials) **(fandisk – with sharp edges)**

(a) (b) (a) (b)

Figure 10: The examples of (a) original 3D model and (b) simplified results.

Table 1: Comparisons of original and simplified models.

model	original model		simplified model		time (ms)
	#faces	#vertices	#faces	#vertices	
bunny	2,915	1,494	531	302	681
dragon	2,730	1,257	1,704	744	540
hand	2,130	1,055	1,852	916	370
cessna	13,546	6,795	5,720	2,882	5,040
fandisk	12,946	6,475	1,518	761	6,477

Table 2: Comparisons of real file sizes.

model	original model (bytes)	simplified model (bytes)	one patch (bytes)
bunny	73,238	13,984	33.451
dragon	63,811	37,328	34.058
hand	50,715	42,960	33.856
cessna	528,002	183,435	25.653
fandisk	280,979	63,345	25.089

Table 2 shows the real file sizes of the original model and the streaming mesh, which contains a simplified model and several patches, of different 3D models. For comparisons, we have converted the file format of the original model to be the same as the simplified mesh (a compressed 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 as shown in Table 1. All of the patches are also stored as a compressed ASCII file; the listed file size is the average size of one patch. Furthermore, since we used only pure Java programming language to develop all the algorithms, it is possible to use our testing program on the web[4]. Moreover, the 3D graphics engine jGL[5] is be used which is also developed by pure Java.

Table 3: Comparisons of transmitting performance with different type of network connection.

type of network connection	time (ms)			
	original model	simplified model	all patches at once	one patch at once
100BaseT	411	45	150	206,558
10BaseT	982	190	761	207,814
PHS(64K)	10,956	2,143	6,039	
file size (bytes)	71,962	13,644	39,859	121,459

The transmitting result for downloading the original and simplified modes of 3D model "bunny" is shown in Table 3. The difference of downloading all the patches at once and the sum of the time for transmitting each patch separately could also be known obviously. Since the latter one needs additional overhead for network package's header, compressed file's header, and the synchronization between the server and the client, the file size and the transmission rate is much worse than the other one. This is also a proving for the necessary of the QoS-like mechanism.

[4] *http://nis-lab.is.s.u-tokyo.ac.jp/~robin/jSM*

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

Table 4: Comparisons of transmitting and reconstructing performances of 3D model "bunny" with PHS.

model	#faces	#vertices	time (ms)
original model	2,915	1,494	10,956
simplified model	531	302	2,143
reconstructed models	1,145	609	4,545
	1,739	906	6,737
	2,915	1,494	11,182

The performance which includes the transmission and reconstruction for 3D model "bunny" by using PHS (Personal Handyphone System) is shown in Table 4. The performances for getting the reconstructed models are also listed in it. The time for showing the simplified model is much less than showing the original one. To show the reconstructed model as Figure 9 (f) does cost only half of the time to show the original one, and the differences between the two models are hardly recognized. Even for reconstructing the original model from the simplified one and all of the patches, the performance is just a little worse than showing the original model directly.

Table 5: The transmitted 3D models' resolution comparison of 3D model "bunny" with different type of connections.

type of connection	transmitted 3D models		
	time (ms)	#faces	#vertices
100BaseT	195	2,915	1,494
10BaseT	761	1,707	890
PHS(64K)	2,143	531	302

Table 5 shows the experimented result of the QoS-like controlling. When transmitting the 3D model "bunny" with different network connections, the client side receives difference 3D models with difference resolutions. The user will get the original model at once when using 100BaseT, but the user using PHS could only see the simplified one. While using 10BaseT, the user receives the proper resolution of the 3D model much better than the PHS user but worse than the 100BaseT user.

7. CONCLUSION AND FUTURE WORK

In this paper, we have presented a new multiresolution streaming mesh for Internet transmission with QoS-like controlling. When using the streaming mesh on the Internet, the user first received a simplified mesh which shape and features could still be recognized easily with less data size due to the current network bandwidth the user faces. If it is necessary to use the model with more details, the server then transmits some patches to the client for reconstructing the original 3D model progressively, and the flow control is done by a QoS-like mechanism. Finally, if the user really needs the original 3D model, after receiving all the patches, he or she will get the lossless original 3D model.

By using this client/server model, the users could receive different 3D models in different resolutions according to their current network bandwidth. Moreover, the 3D model provider does not need to prepare so many different 3D models on the server side. He or she is asked to put only one 3D model with the streaming mesh format on the server, and the server program will talk with the client program to offer the proper 3D model.

The compress rate of some models which have several features is worse than other models, so how to combine the un-removable vertices to decrease the rest vertex number is important. To simplify a 3D model with texture mapping is also our future work. Since we have categorized all the vertices into several levels when doing the 3D mesh simplification process, it is possible to make a hierarchy structure of removed vertices. This information could be used to render the 3D model with LOD (level-of-detail) and support other researches about geometric meshes.

8. ACKNOWLEDGEMENTS

We would like expressing our appreciation to Prof. Jieqing Feng, who gives us many useful suggestions about the geometric modeling. Furthermore, the 3D mesh models used in this paper is downloaded from Large Models Archive[6] at Georgia Tech. and the FTP site[7] of H. Hoppe in Microsoft Research. Thanks for their kindly supports here.

9. REFERENCES

[1] *The Source for Java™ Technology*. Sun Microsystems, Inc., http://java.sun.com, 2001.

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

[3] B. Braden, D. Clark and S. Shenker. *RFC 1633: Integrated Services in the Internet Architecture: an Overview*. The Internet Society, 1994.

[4] B.-Y. Chen and T. Nishita. jGL and its Applications as a Web3D Platform. In *ACM Web3D 2001 Conference Proceedings*, pages 85-91, 2001.

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

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

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*. The Internet Society, 1999.

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

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

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

[11] P. S. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms. In *ACM SIGGRAPH 97 Conference, Multiresolution Surface Modeling Course Notes*, 1997.

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

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

[14] H. Hoppe. View-dependent Refinement of Progressive Meshes. In *ACM SIGGRAPH 97 Conference Proceedings*, pages 189-198, 1997.

[15] H. Hoppe. Progressive Meshes. In *ACM SIGGRAPH 96 Conference Proceedings*, pages 99-108, 1996.

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

[17] A. Khodakovsky, P. Schröder and W. Sweldens. Progressive Geometry Compression. In *ACM SIGGRAPH 2000 Conference Proceedings*, pages 271-278, 2000.

[18] A. Lee, W. Sweldens, P. Schoröder, L. Cowsar and D. Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *ACM SIGGRAPH 98 Conference Proceedings*, pages 95-104, 1998.

[19] R. Ronfard and J. Rossignac. Full-range Approximation of Triangulated Polyhedra. In *Computer Graphics Forum (EUROGRAPHICS 96 Conference Proceedings)*, Vol. 15, No. 3, pages 67-76, 1996.

[20] P. Sander, J. Snyder, S. Gortler and H. Hoppe. Texture Mapping Progressive Meshes. In *ACM SIGGRAPH 2001 Conference Proceedings*, pages 409-416, 2001.

[21] D. To, R. Lau and M. Green. An Adaptive Multi-Resolution Method for Progressive Model Transmission. In *Presence: Teleoperators and Virtual Environments*, Vol. 10, No. 1, pages 62-74, 2001.

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

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

[6] *http://www.cc.gatech.edu/projects/large_models*

[7] *ftp://ftp.research.microsoft.com/users/hhoppe/data*