Danny To
Rynson W. H. Lau
[dannyto, rynson]@cs.cityu.edu.hk
Department of Computer Science
City University of Hong Kong
Hong Kong

Mark Green
mark@cs.ualberta.ca
Department of Computer Science
University of Alberta, Canada

An Adaptive Multiresolution Method for Progressive Model Transmission

Abstract

Although there are many adaptive (or view-dependent) multiresolution methods, support for progressive transmission and reconstruction has not been addressed. A major reason for this is that most of these methods require a large portion of the hierarchical data structure to be available at the client before rendering starts. This is due to the dependency constraints among neighboring vertices. In this paper, we present an efficient, adaptive, multiresolution method that allows progressive and selective model transmission. It is achieved by reducing the neighboring dependency to a minimum. The new method allows visually important parts of an object to be transmitted to the client at higher priority than the less important parts, and progressively reconstructed there for display. It is even possible to transmit only the visible parts of a model and reconstruct these visible parts at the client. The ability to selectively transmit allows the visualization of very large models across the network with minimal delay. We will present how our method works in a client-server environment. We will also show the data structure of the transmission record and some performance results of the method.

1 Introduction

The popularity of the Internet has brought the development of VRML and Java3D, which enable us to create 3-D virtual environments over the Internet. These 3-D distributed applications, however, increase the demand for efficient transmission of 3-D models. Some distributed VR applications (Falby, Zyda, Pratt, & Mackey, 1993; Schmalstieg & Gervautz, 1996; Singh, Serra, Png, & Ng, 1994) even demand real-time on-request transmission of 3-D models. They generally employ a standard client-server architecture, in which a central server maintains a geometry database of the virtual environment and distributes object models to clients upon request. Because these object models may be complex and are usually large in number, the network bandwidth often becomes the bottleneck of the system.

There may be two approaches to encoding the object models to reduce the amount of information that must be sent through the network. The first approach is to apply a geometry-compression method to reduce the storage size of the models. Most geometry-compression methods consider the geometry information shared by neighboring polygons and reduce the amount of data needed to represent the polygon mesh (Deering, 1995; Gumhold & Strasser, 1998; Taubin, Gueziec, Horn, & Lazarus, 1998). The second approach is to

Presence, *Vol. 10, No. 1, February 2001, 62–74*

encode the object models for progressive transmission. It is to convert the models in such a way that partially transmitted models can be rendered and progressively refined as more information is received. Hence, the client no longer needs to wait for the whole model to be transmitted before rendering, and it can thus provide a more immediate visual feedback to the user. This approach has only recently been attracting attention (Gueziec, Taubin, Lazarus, & Horn, 1998; Hoppe, 1996; Popovic & Hoppe, 1997), and the progressive mesh (Hoppe, 1996) is among the first in this area. In this type of method, an object model is decomposed into a base mesh and a sequence of progressive records. The base mesh represents the minimum-resolution model of the object. A progressive record stores information of a vertex split that may slightly increase the resolution of the base mesh by introducing two triangles into it. Hence, by applying the sequence of progressive records to the base mesh, the model will gradually increase in resolution until it reaches the highest resolution when all the records have been applied. The resolution of the model can be decreased by reversing the above operation. We have recently been developing a distributed walkthrough system based on this approach by transmitting object models in the form of progressive meshes in an on-request manner (Chim, Green, Lau, Leong, & Si, 1998; Chim, Lau, et al., 1998). Initial results show that the system can provide a rapid response to the viewer's motion in an Internet environment.

To further enhance the performance of our distributed walkthrough system, we are currently investigating the possibility of incorporating an adaptive (view-dependent) multiresolution method for model transmission. In a distributed environment, when an object first appears in front of the viewer, different regions of the model should be selectively transmitted according to their visual importance. For example, regions of the model facing the viewer or intersecting with the viewer's line of sight should be transmitted at higher resolution, whereas the rest could be transmitted at lower resolution. For small models, the gain of this method may not be high, but for large or complex models the gain may be significant. However, for an adaptive multiresolution method to be applicable in a distributed environment, it must also support progressive transmission. (Otherwise, we still need to wait for the complete model to be transmitted before rendering begins.) In other words, while the object model is being selectively transmitted from the server according to the visual importance of different parts of the model, it is progressively reconstructed at the client.

In this paper, we present the adaptive multiresolution method that we have developed for progressive model transmission. The new method supports selective transmission of models from the server on request, and progressive reconstruction of them for rendering at the client. The rest of the paper is organized as follows. Section 2 gives a brief survey on existing adaptive multiresolution methods. Section 3 presents an overview of our method. Section 4 discusses the construction of the vertex trees. Section 5 shows how the vertex trees can be selectively transmitted from the server and progressively reconstructed at the client. Section 6 presents some results and evaluates the new method. Finally, section 7 presents the conclusions of the paper and discusses some possible future work.

2 Related Work

An adaptive multiresolution method can optimize the resolution of an object model for rendering by locally adjusting the resolution of it according to some dynamic information, such as the user's position and line of sight, and the velocity speed of the object. We refer to this dynamic information as the *view and animation parameters*. The advantage of dynamically optimizing the model resolution is more obvious for large models. For example, if the user is flying through a landscape modeled as a single surface, he/she can see only a very small part of the landscape most of the time. An adaptive multiresolution method can then be used to adjust the resolution of the landscape model in such a way that the region where the user is looking has a high resolution while the rest has a lower resolution. This ability to adjust the resolution of a model adaptively according to the runtime view and animation parameters implies that the method must be able to operate in real time. Various adaptive multiresolution methods have been proposed, and the typical ones are discussed here.

A few methods have been developed for managing large terrain models (Falby et al., 1993; Lindstrom et al., 1996; Pajarola, 1998). Basically, these methods regularly subdivide a large terrain surface. A hierarchical data structure, usually in the form of a quadtree, is constructed with the leaf nodes representing individual polygons (that is, the highest resolution) and the root node representing the whole terrain surface (the lowest resolution). Each successive higher level of the tree from the leaf nodes represents a four-time decrease in resolution. With this hierarchical data structure, the resolution of a local region can be adjusted simply by choosing the polygons from higher- or lower-level nodes of the tree for rendering. Although this kind of method usually uses simple data structures and is very efficient, it requires the surface to be regularly subdivided. Although this may be fine with a smooth landscape, an excessive amount of polygons may need to be created when the landscape contains many crests and valleys. As such, this kind of method may not be suitable for modeling objects of arbitrary topology. A similar kind of method, but for arbitrary 3-D object models is subdividing the model into regular 3-D cells (Low & Tan, 1997; Luebke & Erikson, 1997; Rossignac & Borrel, 1993; Schaufler & Sturzlinger, 1995). The cells are then hierarchically combined to form an octree. The resolution of a local region can be reduced by merging (or clustering) multiple cells and vertices within the cells. Although the creation of the octree may be time-consuming (Low & Tan, 1997), the runtime performance of these methods is very high. The major limitation of these methods, however, is that the geometry of the model may not be preserved after the simplification.

Methods that preserve the geometry of the object model are mainly based on edge decimation. Xia, Zl-Sana, and Varshney (1997) used a merge tree to store the edge collapses in a hierarchical manner. To prevent mesh folding, during the construction of the hierarchy, the sequence of edge collapses is constrained to be nonoverlapping. As a result, a dependency exists among the nodes in the merge tree. An edge is allowed to split or collapse only if certain neighboring vertices exist. Hoppe (1997) presented the view-dependent progressive mesh similar to Xia et al. (1997) by using a vertex hierarchy, but they differ in that the sequence of edge collapses in Hoppe (1997) is unconstrained and geometrically optimized with minimum dependencies among the collapses. However, because the method requires a model to be transmitted in the form a progressive mesh (Hoppe, 1996), the transmission order must follow a predefined order. The method we propose here, on the other hand, supports dynamic view-dependent transmission in addition to view-dependent refinement. In addition, the hierarchy is constructed from the progressive mesh, and thus the model can be transmitted in this format. A more efficient hierarchical approach designed specifically for terrain models is presented by Hoppe (1998). The method allows real-time walkthrough of a large terrain model. Because neighboring vertices may be forced to split in order to satisfy the preconditions of a vertex split, other invisible parts of the hierarchy may also need to be transmitted to the client.

Floriani, Magillo, and Puppo (1998) proposed a method based on a Directed-Acyclic-Graph (DAG) called multitriangulation (MT). During the simplification or refinement process, a sequence of local operators is applied. Each of them modifies a small region of the mesh, called a *fragment*. The fragments are arranged into a partial order according to their dependencies and stored in the DAG. At runtime, the resolution of each fragment can be changed independently to produce an adaptive multiresolution model. In Cignoni, Puppo, and Scopigno (1997), the MT is represented as a simplicial complex in 3-D space called *hypertriangulation*. The third dimension of it represents the resolution of the fragment, and triangles of different fragments are welded together to form a model. Gueziec et al. (1998) introduced a simpler DAG-based method for progressive model transmission. Surfaces in the model are partitioned during the edge-decimation process, and independent surface patches can be transmitted in the same batch, thus allowing the model to be transmitted progressively to the client. However, because the DAG can be constructed only after the whole model is received, adaptive refinement of different parts of the model is possible only after the client has received the whole model.

All the methods discussed above can adaptively refine the resolution of an object model, and most of them are very efficient. Due to the neighboring dependency constraints, they may require a large portion or even the complete model to be available at the client before rendering can begin. Support for progressive and selective transmission has not been addressed.

3 Method Overview

Our adaptive multiresolution method is based on edge decimation. During the preprocessing stage, we simplify the model by collapsing edges and simultaneously constructing a set of hierarchies to represent the parent-child relationship of the vertices. These hierarchies are referred to as the *vertex trees,* and the root nodes of these trees form the base mesh of the model. The structure of the vertex trees is similar to that in Hoppe (1997) and Xia et al. (1997). To reduce the cost of pointer usage, the vertex trees are linearized to a set of one-dimensional arrays and stored at the server. At runtime, when the client requests an object model, the server transmits the base mesh nodes of the vertex trees first. Other nodes of the vertex trees will then be transmitted from top to bottom progressively; however, we may selectively transmit nodes from different vertex trees according to the view and animation parameters of the client at the time of the transmission. For example, if a region of an object is inside the view frustum of the viewer, we may transmit more nodes from the corresponding vertex trees and less (or even none) from others.

For the server to determine the visible vertices efficiently at any time, a *visible vertex list* (as shown in figure 3) is created to link up all the visible vertices of the vertex trees. This list is updated as the view and animation parameters are changed. A similar visible vertex list is also maintained at the client. At the beginning, the list links up all nodes in the base mesh and these nodes will be transmitted to the client. We then gradually move down the list and transmit new nodes on the list, if they have not been transmitted, until the location in the list reflects a suitable resolution of the model with respect to the current view and animation parameters. As the client receives tree nodes from the server, it reconstructs the vertex trees locally. The visible vertex list is adjusted, and triangles are retrieved from the nodes in this list to form a triangle list for display. This triangle list represents an adaptive multiresolution model of the object.

To compare our method with existing methods such as Cignoni et al. (1997), Floriani et al. (1998), Gueziec et al. (1998), Hoppe (1997, 1998), Xia et al. (1997), our method does not need to perform recursive dependency checking. In Cignoni et al. (1997), Floriani et al. (1998), Gueziec et al. (1998), dependency checking in the DAG is needed when retrieving the triangle fragments. In Hoppe (1997, 1998) and Xia et al. (1997), recursive dependency checking of the hierarchy is needed to determine the dependencies among vertices; hence, other parts of the hierarchy need to be presented at the client. In our method, we reduce the dependency among vertices to a minimum by storing the vertex fan of each vertex at its node to allow only a simple parent-child checking. This gives greater flexibility in changing the resolution of the object surface. In addition, this minimal dependency among nodes allows us to transmit only the visible parts of the vertex trees to the client to reduce the memory cost of the client and the network bandwidth requirement.

Before we discuss in detail how we construct the vertex trees and transmit them from the server to the client, we need to introduce a few notations here. We define the *Edge Fan, EFan*(v), of vertex v as the set of edges adjacent to v ordered in clockwise direction. *The Vertex Fan, VFan*(v), of vertex v is the set of first-ring neighboring vertices of v ordered in clockwise direction. The *Triangle Fan, TFan*(v), of vertex v is the set of triangles adjacent to v ordered in clockwise direction. The *Triangle Ring, TRing*(e) of an edge $e = (v_i, v_j)$ is the set of triangles adjacent to either v_i or v_j; that is, $TRing(e) = TFan(v_i) \cup TFan(v_j)$. Figure 1 shows examples of $EFan(v)$, $VFan(v)$, $TFan(v)$ and $TRing(e)$.

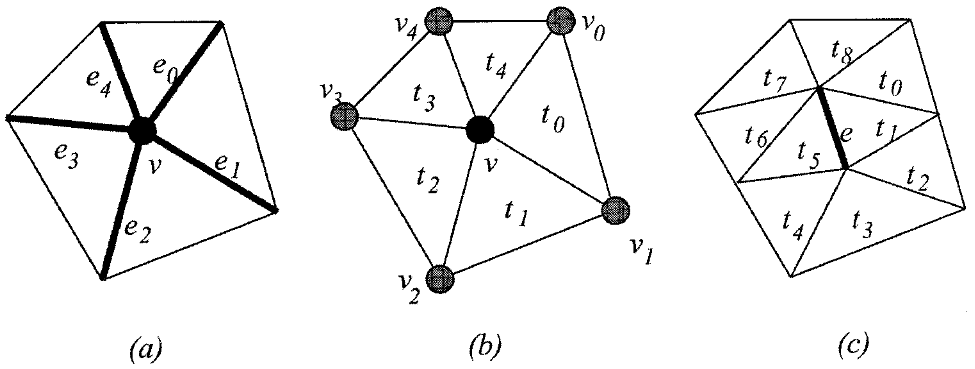


Figure 1. *Examples of (a)* EFan(v) = [e_0, e_1, e_2, e_3, e_4], *(b)* VFan(v) = [v_0, v_1, v_2, v_3, v_4], *and* TFan(v) = [t_0, t_1, t_2, t_3, t_4], *and (c)* TRing(e) = [t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8].

4 Construction of the Vertex Trees

To simplify the model, we apply an edge-decimation method similar to the one described in (Lau, Green, To, & Wong, 1998). We first determine the importance of each vertex in the model based on its local sharpness. (A sharp vertex has a higher importance value than a flat vertex.) An edge importance value for each triangle edge in the model is then determined based on the importance values of its two vertices and the length of the edge. (A short edge will have a relatively lower edge importance than a long edge.) We then assign a node to each vertex and insert all the nodes into a hash table using the vertex importance value as the index. During the decimation process, the node list located at the first entry of the hash table—that is, the list of vertices with the lowest importance values—is first selected. For each node in the list, the edge fan is examined and the edge with the lowest edge-importance value will be chosen to collapse. The vertex with a lower importance value, called the *child vertex,* will be merged to the other, called the *parent vertex.* An edge collapse is valid only if it would not result in mesh folding.

Before an edge is collapsed, the vertex fan of each of the two vertices is determined. As the edge is collapsed, the child vertex is removed from the hash table and inserted into the vertex tree to become the left child node of the parent vertex. The parent node is duplicated to become the right child node. The two vertex fans are then stored in the corresponding child nodes. Each of these vertex fans will later be used to determine the triangle fan of the vertex. The reason for storing the vertex fan instead of the triangle fan is that it is much cheaper in terms of memory cost to store the vertex fan. Because the local geometry is changed after the collapse, the importance values of the adjacent vertices need to be recalculated, and the vertices are then reinserted into the hash table. This decimation process continues until all the collapsible vertices are removed from the hash table. At this point, several vertex trees will have been constructed. The root nodes of these vertex trees form the vertices of the lowest-resolution model and, hence, the base mesh of the object model.

To reconstruct the triangle model from the vertex trees, we need to select appropriate triangles from the triangle fans of nodes on the visible vertex list. Because the triangles from neighboring triangle fans may be at different resolution levels, some of them may overlap each other. With the edge collapse sequence shown in figure 2 as an example, figure 3 shows the corresponding vertex trees and the triangle fans of the vertices. We may observe that *TFan*(v_2) overlaps with *TFan*(v_4). If both of them are visible at the same time, triangles in *TFan*(v_4) are at higher resolution levels and therefore preferred.

To be able to efficiently determine which triangles are at higher resolution during runtime, we introduce two state variables here: the *vertex state* and the *triangle*

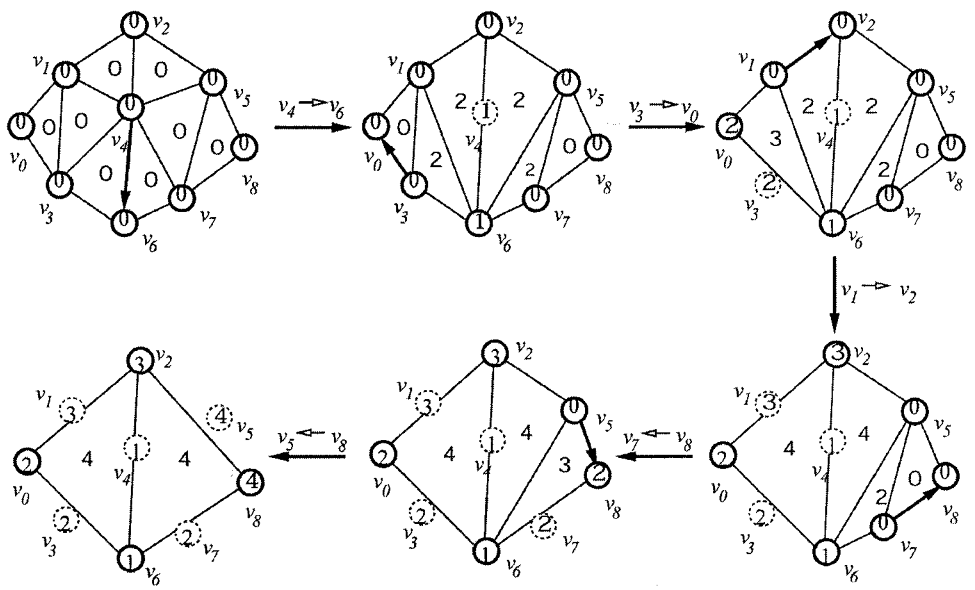


Figure 2. *An edge-collapse sequence and the corresponding change in triangle states and vertex states. (Numbers in bold indicate a change in triangle or vertex state.)*

state. A vertex state is assigned to each vertex in the model to indicate the resolution level of the vertex, whereas a triangle state is assigned to each triangle in the model to indicate the resolution level of the triangle. Let S_v and S_t denote the vertex state and the triangle state, respectively. All the state values are initially set to 0. When an edge $e = (v_{child}, v_{parent})$ is chosen to collapse, we first identify a triangle from $TRing(e)$ with the highest triangle state S_{max} and then update the vertex states of the two vertices to S_{max}. The triangle states of $TRing(e)$ are then incremented by 1. These updates are shown in the following equations. Figure 2 shows an edge-collapsing sequence and the corresponding change in triangle states and vertex states.

$$S_{\max} = \max_{t \in TRing(e)} \{S_t\} \tag{1}$$

$$S_{v_{parent}(e)} = S_{v_{child}(e)} = \begin{cases} 1 & \text{if } S_{\max}(e) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$S_t = S_t + 1, \quad [\ t \in TRing(e) \tag{3}$$

5 Progressive and Selective Model Transmission

When a model is transmitted in the form of a progressive mesh, the base mesh of the model is transmitted first, followed by the sequence of progressive records. The order of these progressive records is predetermined according to the geometric importance of the vertices or triangle edges. With the new adaptive multiresolution method, the order of transmitting the tree nodes can be determined during runtime according to both the geometric importance and the visual importance of the vertices. The visual importance of a vertex is determined from the view and animation parameters.

At runtime, the server is responsible for selecting appropriate nodes for transmission to the client, and the client is responsible for constructing the vertex trees from the information supplied by the server and determining the appropriate triangles for display. We will discuss this in detail in the following subsections. For simplicity, we use figures 2 and 3 as examples in our discussion.

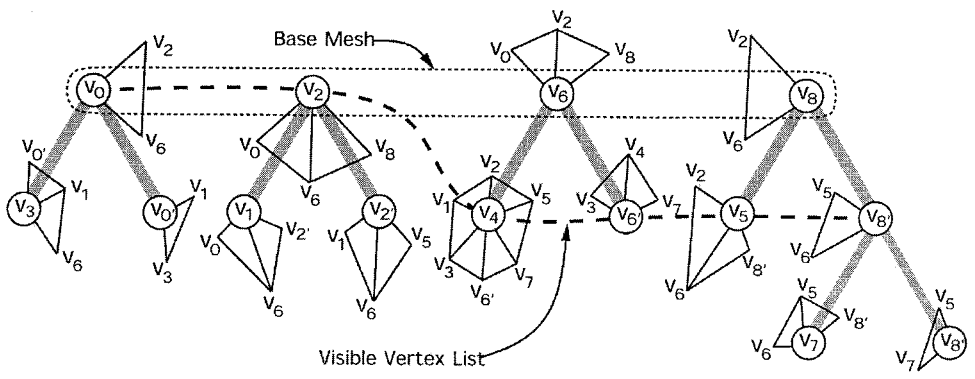


Figure 3. *Triangle fans obtained from the vertex fans stored in the nodes of the vertex trees.*

5.1 The Server Process

At the beginning of transmitting a model, the visible vertex list contains only the base mesh: that is, $\{v_0, v_2, v_6, v_8\}$ in figure 3. The information stored in each node of the base mesh is packaged into a *node record* for transmission to the client. The visible vertex list will then be moved down the trees gradually so that more nodes will be transmitted to the client. Suppose that both v_6 and v_8 are found to be visible and that their local resolutions need to be increased. The visible vertex list will move down one level at nodes v_6 and v_8, and the updated visible vertex list becomes $\{v_0, v_2, v_4, v_{6'}, v_5, v_{8'}\}$. We may find that vertices $\{v_1, v_3, v_7\}$ in $VFan(v_4)$ and vertices $\{v_3, v_7\}$ in $VFan(v_{6'})$ are not currently visible and therefore not available at the client because they are located below the visible vertex list. Hence, we need to trace up the corresponding vertex trees and replace these vertices with their first visible parents: that is, $\{v_1, v_3, v_7\}$ by $\{v_2, v_0, v_{8'}\}$. The task of tracing up the vertex trees for visible vertices is performed by the server. Before a node record is sent, the server checks each vertex of its vertex fan. If an invisible vertex is found, the first transmitted parent of this vertex is identified and its ID is also sent to the client. There are three types of node records for transmission. One is for the nodes in the base mesh called *base nodes*. The other two are for the left and right child nodes called *left nodes* and *right nodes*, respectively. The major difference between these three types of node records is that the right node does not include the vertex coordinate, as it is available from its parent node. In addition, the left node needs to include a vertex ID to identify its parent node. The data structures of these three types of nodes are:

```
// the right child node
class RightNode {
public:
  unsigned short VID;           // Vertex ID
  struct {
    unsigned short closed:1;    // indicate if VFan forms closed loop
    unsigned short VFanLen:7;   // length of vertex fan
  } VFanInfo;
  unsigned short *VFan;         // array for vertex fan
};
```

```
// the base node (inherit from RightNode)
class BaseNode : : public RightNode {
public:
  myFloat Vx, Vy, Xz;          // vertex coordinate (2bytes each)
  unsigned char Vstate;        // vertex state
};
// the left child node (inherit from BaseNode)
class CLeftNode : : public BaseNode {
public:
  unsigned short PID;          // Parent Vertex ID
  unsigned char Vstate;        // vertex state
};
```

5.2 The Client Process

At the client, the vertex trees are being reconstructed as the node records are received. A visible vertex list is maintained to indicate the current visible vertices. This list is constantly being updated as more nodes are received at the client to reflect the change in the model resolution. This value is based on some view and animation parameters, such as object distance from the viewer, object velocity, and current system frame rate. To render the model, the client traverses each node on the list and retrieves triangles from it to form a triangle list for display. This triangle list represents an adaptive multiresolution model of the object. However, the problem here is how to efficiently select suitable triangles from each of the nodes on the list for display. We handle this in the following three steps:

- As mentioned earlier, when triangles from neighboring triangle fans overlap each other, we select those at higher resolution levels for display. To do this, for each triangle in $TFan(v)$ of vertex v, we compare the vertex state S_v of v with the other two vertex states of the triangle. The triangle is inserted to the triangle list only if S_v is the lowest among the vertex states of all three vertices.
- Some triangles in the triangle fans may be degenerate. We would detect and remove them by checking whether there are duplicated vertex IDs in each triangle.
- To reduce the time needed to create the triangle list, we reuse the triangle list produced from the previous frame. When a triangle is inserted to the list, its location in the list is stored at its corresponding node. This allows rapid modification of the triangles in the list. We also note the number of times each vertex appears in the inserted triangles. If a vertex appears only once, it must be a feature vertex if and only if it has a zero vertex state; if it has a nonzero vertex state, it must be a dangling vertex and the corresponding triangle is considered as invalid.

6 Results and Discussions

We have implemented the new method using C++ and Open Inventor. The server module communicates with the client module using TCP/IP. We tested the two modules on two SGI Octane workstations, each with a 195 MHz R10000 CPU and 256MB RAM, using several large triangle models. Figures 4, 5, 6, and 7 demonstrate some results using the terrain, teeth, and Crater Lake models. In these figures, the

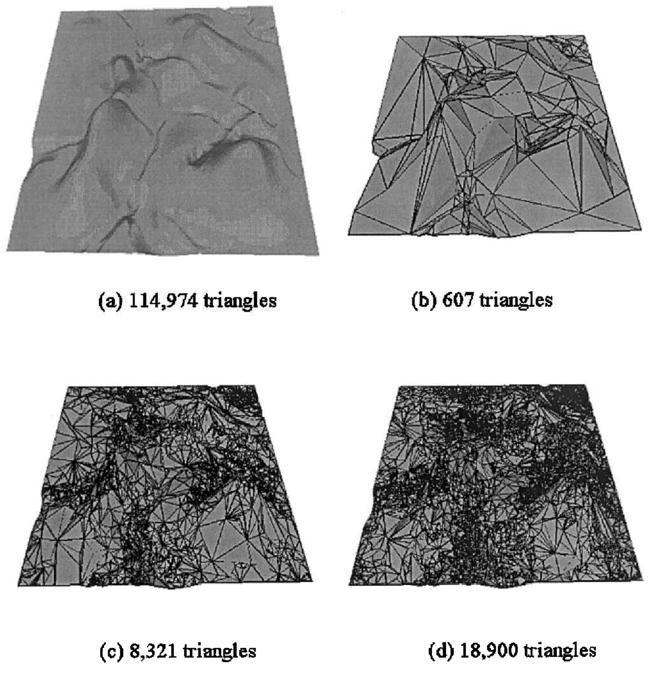


Figure 4. *Adaptive refinement of a terrain model: (a) the original model; (b), (c), and (d) adaptively refined at different resolutions.*

rectangular box represents the view region of the user. Figure 4(a) shows the original terrain model with 114,974 triangles. Figures 4(b), 4(c), and 4(d) show the adaptive refinement of it with 607, 8,321, and 18,900 triangles, respectively. Progressive and selective transmission of the same terrain model is shown in figure 5. A similar set of results using the teeth model is shown in figure 6.

We can also make use of the progressive and selective transmission nature of the new method to transmit only sections of an object that are inside the view region. Figure 7 shows an adaptive refined Crater Lake model with and without view clipping.

When view clipping is used, a node is transmitted, or considered as visible, only if at least one of the vertices from its vertex fan is inside the view region and the normal vector of the node is facing the viewer. (The inside test on each vertex of the vertex fan at runtime can be very expensive. To simplify this test, we precompute the longest edge of the edge fan as the radius of the bound-

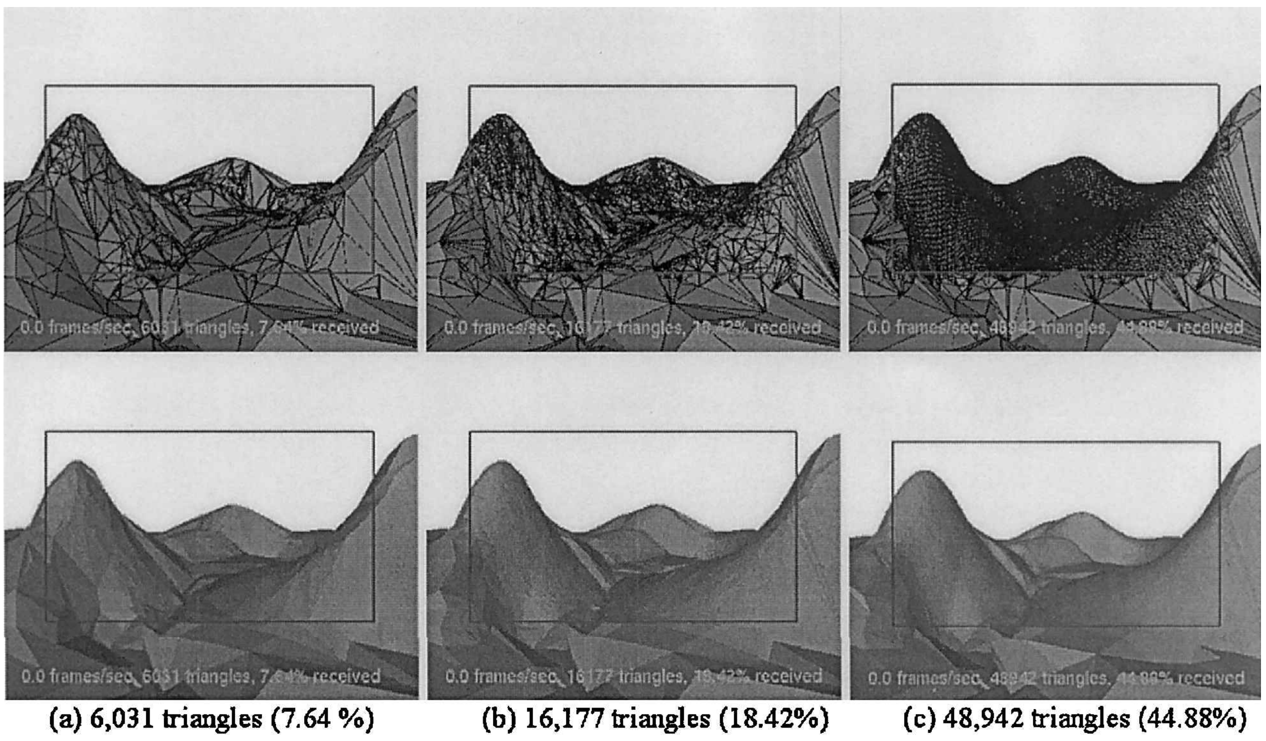
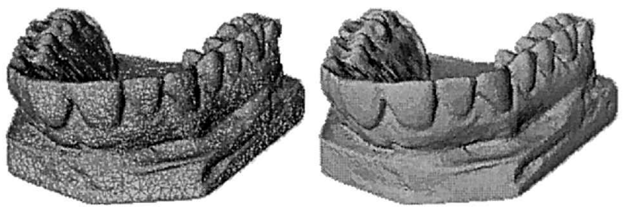

0.0 frames/sec, 6031 triangles, 7.64% received
0.0 frames/sec, 16177 triangles, 18.42% received
0.0 frames/sec, 48942 triangles, 44.88% received
(a) 6,031 triangles (7.64 %)
(b) 16,177 triangles (18.42%)
(c) 48,942 triangles (44.88%)

Figure 5. *Progressive and selective transmission of the terrain model shown in figure 4.*



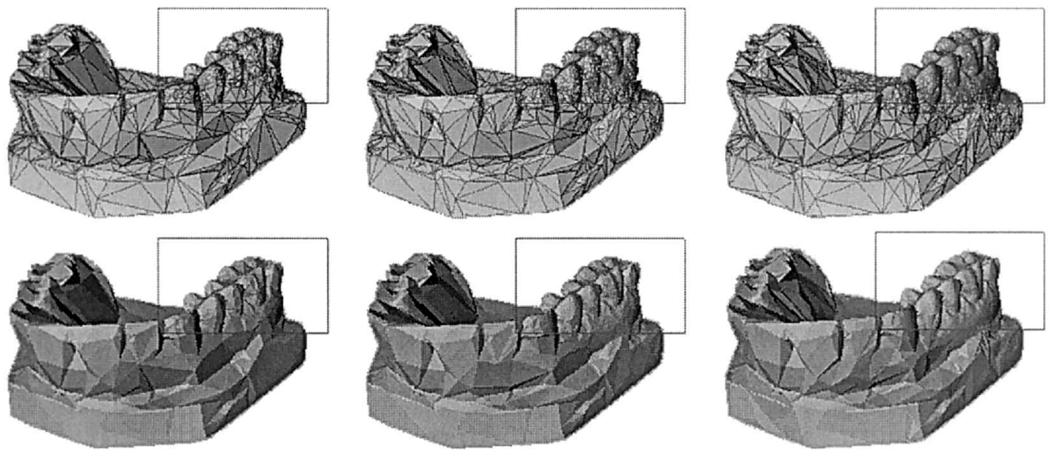


Figure 6. *Adaptive refinement of a teeth model: (a) the original model; (b), (c), and (d) adaptively refined at different resolutions.*

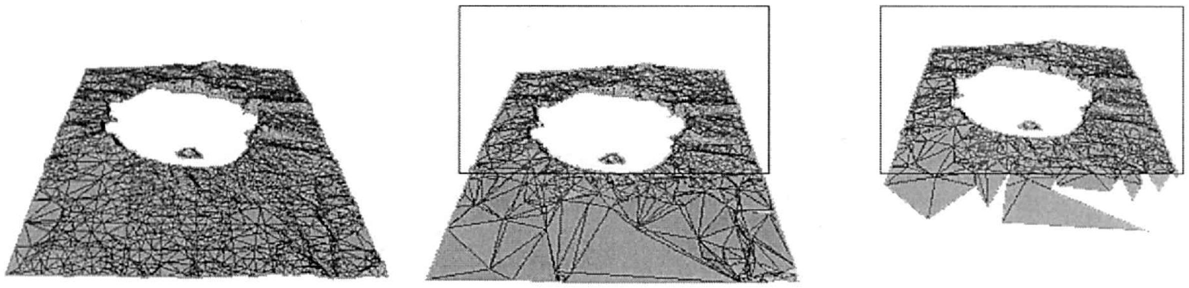


Figure 7. *Adaptive refinement of a crater model with and without view clipping: (a) the original model, (b) without view clipping, and (c) with view clipping.*

ing sphere for the node. During runtime, if the bounding sphere is found to overlap with the view region, the vertex is assumed to be inside.)

Table 1 shows the performance of preprocessing different models. The vertex tree construction time includes the calculation of importance values, edge deci-

Table 1. *Preprocessing Performance of Our Method*

Model	Number of triangles	Vertex tree construction time
Terrain	114,974	60.39 s
Bunny	69,451	35.70 s
Teeth	58,328	31.32 s
Crater Lake	9,620	0.09 s

mation, and the construction of the vertex trees. This construction time is proportional to the total number of triangles in the model.

Table 2 shows the size of the vertex trees in uncompressed representation, the time for progressive transmission of the vertex trees, and the time for reconstructing them at the client. The size of the vertex trees is comparable with the progressive mesh (Hoppe, 1996). Assume that the average size of the vertex fan in each node is 6. As shown in section 5.1, the average size of a base node record is 22 bytes and that of a child node record (average between a right child node and a left child node) is 20 bytes. In our experiments, the average sizes of the two types of records are 24 bytes and 21 bytes, respectively. The difference is mainly due to the byte packing performed by the operating system. We measured the transmission of the vertex trees from the server to the client through a 10Mbits Ethernet during the daytime. Columns 4 and 5 of table 2 show the time needed to transmit the base nodes and the rest of the vertex trees, respectively. Note that the figures shown are the time needed to transmit the complete model. In many walkthrough environments, only part of the model will ever be seen, and only the corresponding part of the vertex trees needs to be transmitted using our method.

If we assume that the bandwidth available to us in the experiments was 2Mbits and that the Internet bandwidth is one-tenth of this (0.2Mbits), the time needed to transmit the vertex trees through the Internet will be roughly ten times longer than the figures shown in the table. Because we can visualize the model as soon as the base mesh is available at the client, this transmission performance can provide a reasonably fast response to the viewer's movement even over the Internet. In addition, after the base mesh has been transmitted, almost all the nodes subsequently transmitted help refine the visible region of the model. We have tested the client process on a SGI O_2 workstation, with a 200 MHz R5000 CPU and 128MB RAM. A walkthrough of the terrain surface shown in figure 4(a) is performed, and the average frame rate has increased from 2.47 fps (without using our method) to 7 fps. Note also that, in our experiments, each node was sent as an individual packet through the network. The transmission time can be further reduced if we transmit the nodes in batch as discussed by Gueziec et al. (1998). We would also expect a reduction in transmission time by integrating a geometric compression technique such as that in Deering (1995) and Taubin et al. (1998). Finally, column 6 of table 2 shows the time for constructing the complete vertex trees at the client side.

In summary, the method we propose here has the following advantages:

- Due to the minimal dependency among neighboring vertices, our method allows multiresolution models to be transmitted selectively and progressively according to the view and animation parameters. This is important in particular for transmitting large models such as a large terrain surface. If a terrain surface is encoded in the form of a progressive mesh (Hoppe, 1996), for example, most of the progressive records being transmitted may be irrelevant to what the viewer is seeing. Not only may the bandwidth be wasted, but the viewer may also have to accept a lower visual quality for a longer period of time while the surface is being transmitted.
- Because only the visible part of the model may need to be transmitted, the new method can save the client's memory space if the viewer visits only small sections of a large model. The invisible sections may never need to be transmitted.
- The Internet is unreliable in transmitting information, and packets may be lost during transmission. When using the progressive mesh method, if a progressive record is lost during the transmission, subsequent records received cannot be used until the

Table 2. *Runtime Performance of Our Method*

Model	Size of vertex trees		Progressive transmission time		Vertex tree reconstruction time
	Base nodes	Child nodes	Base nodes	Child nodes	
Terrain	7.6KB	2.46MB	0.02 s	8.10 s	1.74 s
Bunny	16.3KB	1.38MB	0.01 s	5.36 s	0.97 s
Teeth	36.4KB	1.11MB	0.03 s	4.80 s	0.92 s
Crater Lake	19.2KB	0.16MB	0.00 s	0.61 s	0.10 s

lost record is retransmitted. This causes a roundtrip delay: the client notifies the server of the lost record and then the server retransmits the lost record. In our method, because edge collapses are independent of each other, the loss of a node during transmission will not affect subsequent nodes received.

7 Conclusions and Future Work

In this paper, we present an efficient adaptive multiresolution method that allows progressive and selective model transmission. The priority for transmitting different regions of the object model can be determined according to the view and animation parameters obtained at runtime, such as the user's viewing region and line of sight. The ability to perform adaptive multiresolution modeling with partially transmitted models will be useful in the visualization of large and detailed geometric models, or in the walkthrough of a distributed virtual environment over a low-speed network. We have discussed the mechanism for transmitting the models with our method. We have also presented the preprocessing and runtime performances of the new method.

Based on our adaptive method, we are currently working on a real-time multiresolution modeling technique for textured and animated objects. We are also integrating the new method into our virtual walkthrough system (Chim, Green et al., 1998; Chim, Lau et al., 1998) for transmitting large models. Compression has not been considered in the current implementation, and it would be an advantage to incorporate a geometric compression method to reduce both storage and transmission time.

Acknowledgments

The work described in this paper was partially supported by a CityU grant (Project 7100003).

References

Chim, J., Green, M., Lau, R. W. H., Leong, H. V., & Si, A. (1998). On caching and prefetching of virtual objects in distributed virtual environments. *Proceedings of ACM Multimedia* (pp. 171–180).

Chim, J., Lau, R. W. H., Si, A., Leong, H. V., To, D., Green, M., & Lam, M. (1998). Multi-resolution model transmission in distributed virtual environments. *Proceedings of ACM Symposium on Virtual Reality Software and Technology* (pp. 25–34).

Cignoni, P., Puppo, E., & Scopigno, R. (1997). Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer, 13*(5), 199–217.

Deering, D. (1995). Geometry compression. *Proceedings of ACM SIGGRAPH'95* (pp. 13–20).

Falby, J., Zyda, M., Pratt, D., & Mackey, R. (1993). NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. *Computers & Graphics, 17*(1), 65–69.

Floriani, L., Magillo, P., & Puppo, E. (1998). Efficient implementation of multi-triangulation. *Proceedings of IEEE Visualization* (pp. 43–50).

Gueziec, A., Taubin, G., Lazarus, F., & Horn, W. (1998). Simplical maps for progressive transmission of polygonal surfaces. *Proceedings of Symposium on VRML'98* (pp. 25–31).

Gumhold, S., & Strasser, W. (1998). Real-time compression of triangle mesh connectivity. *Proceedings of ACM SIGGRAPH'98* (pp. 133–140).

Hoppe, H. (1996). Progressive meshes. *Proceedings of ACM SIGGRAPH'96* (pp. 99–108).

———. (1997). View-dependent refinement of progressive meshes. *Proceedings of ACM SIGGRAPH'97* (pp. 189–198).

———. (1998). View-dependent level-of-detail control and its application to terrain rendering. *Proceedings of IEEE Visualization'98* (pp. 35–42).

Lau, R. W. H., Green, M., To, D., & Wong, J. (1998). Real-time continuous multi-resolution method for models of arbitrary topology. *Presence: Teleoperators and Virtual Environments, 7*(1), 22–35.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., & Turner, G. (1996). Real-time continuous level of detail rendering of height fields. *Proceedings of ACM SIGGRAPH'96* (pp. 109–118).

Low, K., & Tan, T. (1997). Model simplification using vertex clustering. *Proceedings of ACM Symposium on Interactive 3D Graphics* (pp. 75–81).

Luebke, D., & Erikson, C. (1997). View-dependent simplification of arbitrary polygonal environments. *Proceedings of ACM SIGGRAPH'97* (pp. 199–208).

Pajarola, R. (1998). Large-scale terrain visualization using the restricted quadtree triangulation. *Proceedings of IEEE Visualization'98* (pp. 19–26).

Popovic, J., & Hoppe, H. (1997). Progressive simplicial complexes. *Proceedings of ACM SIGGRAPH'97* (pp. 209–216).

Rossignac, J., & Borrel, P. (1993). Multi-resolution 3D approximations for Rendering. Modeling in Computer Graphics (pp. 455–465). New York: Springer-Verlag.

Schaufler, G., & Sturzlinger, W. (1995). Generating multiple levels of detail for polygonal geometry models. *Proceedings of Eurographics Workshop on Virtual Environments* (pp. 31–41).

Schmalstieg, D., & Gervautz, M. (1996). Demand-driven geometry transmission for distributed virtual environments. *Proceedings of Eurographics'96* (pp. 421–432).

Singh, G., Serra, L., Png, W., & Ng, H. (1994). BrickNet: A software toolkit for network-based virtual worlds. *Presence: Teleoperators and Virtual Environments, 3*(1), 19–34.

Taubin, G., Gueziec, A., Horn, W., & Lazarus, F. (1998). Progressive forest split compression. *Proceedings of ACM SIGGRAPH'98* (pp. 123–132).

Xia, J., El-Sana, J., & Varshney, A. (1997). Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transaction on Visualization and Computer Graphics, 3*(2), 171–183.