In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pp. 89-95, Dec. 1999.

A Method for Progressive and Selective Transmission of Multi-Resolution Models

Danny S.P. To Rynson W.H. Lau
Department of Computer Science
City University of Hong Kong, Hong Kong
{dannyto, rynson}@cs.cityu.edu.hk

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

Abstract

Although there are many adaptive (or view-dependent) multi-resolution methods developed, support for progressive transmission and reconstruction has not been addressed. A major reason for this is that most of these methods require large portion of the hierarchical data structure to be available at the client before rendering starts, due to the neighboring dependency constraints. In this paper, we present an efficient multi-resolution method that allows progressive and selective transmission of multi-resolution models. This 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. We will present the new method and discuss how it 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 3D virtual environments over the Internet. These 3D distributed applications, however, increase the demand for efficient transmission of 3D models. Some distributed VR applications [7, 22, 23] even demand for real-time on-request transmission of 3D 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 requested. 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 for encoding the object models in order to reduce the amount of information needed to be sent through the network. The first approach is by applying either a geometry compression method or a level of details (LoD) 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 [6, 9, 24]. LoD methods consider the fact that the details of an object become less visible as the object moves away from the viewer and thus fewer polygons may be used to represent a distant object. These methods can potentially reduce the amount of data needed to be transmitted by sending models of appropriate resolutions to the clients [22]. There have been a lot of LoD methods proposed [3, 10, 20]. In general, this approach requires the complete model to be sent to the client before the model can be used.

The second approach is to encode the object models for progressive transmission. This means that the models are encoded 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 can thus provide a more immediate visual feedback to the user. This approach has only recently been attracting a lot of attention [8, 11, 19], and the progressive mesh [11] is among the first in this area. In this 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 [1, 2]. 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) multi-resolution 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 importances. For example, regions of the model facing the viewer or intersecting with the viewer's line of sight should be transmitted at higher resolution while the rest could be transmitted at lower resolution. However, for an adaptive multi-resolution 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 the visual importance of different parts of the model, it is progressively reconstructed at the client.

In this paper, we present the adaptive multi-resolution method that we have developed for 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 multi-resolution 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 a conclusion of the paper and discusses some possible future work.

2 Related Work

An adaptive multi-resolution 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 moving speed of the object. We refer to this dynamic information as the *view and animation parameters*. The advantage of dynamically optimizing the resolution of the model is more obvious for large models. For example, if the user is flying through a landscape modeled as a single surface, he/she can only see a very small part of the landscape most of the time. An adaptive multi-resolution 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 at has a high resolution while the rest has lower. This ability to adjust the resolution of a model adaptively according to the run-time view and animation parameters implies that the method must be able to operate in real-time. Various adaptive multi-resolution methods have been proposed and the typical ones are discussed here.

A few methods have been developed for managing large terrain models [7, 15, 18]. 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, i.e., the highest resolution, and the root node representing the whole terrain surface, i.e., 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 methods usually uses simple data structures and is very efficient, it requires the surface to be regularly subdivided. While this may be fine with smooth landscape, excessive amount of polygons may need to be created when the landscape contains a lot of crests and valleys. As such, this kind of methods may not be suitable for modeling objects of arbitrary topology. A similar kind of methods but for arbitrary 3D object models is by subdividing the model into regular 3D cells [16, 17, 21]. 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 [16], the run-time 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 et al. [25] uses 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 non-overlapping. 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 in [12] presents the view dependent progressive mesh similar to [25] by using a vertex hierarchy. However, they differ in that the sequence of edge collapses in [12] is unconstrained and geometrically optimized with minimum dependencies among the collapses. In addition, the hierarchy is constructed from the progressive mesh [11] and thus the model can be transmitted in this format. A more efficient hierarchical approach designed specifically for terrain models is presented in [13]. The method allows real-time walkthrough of a large terrain model. Since 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 et al. [5] proposes a method based on a Directed-Acyclic-Graph (DAG) called multi-triangulation (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 run-time, the resolution of each fragment can be changed independently to produce an adaptive multi-resolution model. In [4], the MT is represented as a simplicial complex in 3D 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. Guéziec et al. [8] introduces 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, since the DAG can only be constructed after the whole model is received, adaptive refinement of different parts of the model is only possible after the whole model is received by the client.

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 large portion or even 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 multi-resolution method is based on edge decimation. During the preprocessing stage, we simplify the model by collapsing edges and at the same time 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 [12, 25]. To reduce the cost of pointer usage, the vertex trees are linearized to set of one-dimensional arrays and stored at the server. At run-time, when the client requests for an object model, the server will transmit 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 user, 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 the list down and transmit new nodes on the list, if they have not been transmitted, until the location of 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 multi-resolution model of the object.

To compare our method with existing methods such as [4, 5, 8, 12, 13, 25], our method does not need to perform the recursive dependency checking. In [4, 5, 8], dependency checking in the DAG is needed when retrieving the triangle fragments. In [12, 13, 25], recursive dependency checking of the hierarchy is needed to determine the dependency among vertices and 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. We do this by storing the vertex fan of each vertex at its node to allow only a simple parent-child checking. This gives a greater flexibility in changing the resolution of the object surface. In addition, this minimal dependency between nodes allows us to transmit only the visible parts of the vertex trees to the client to reduce network bandwidth and storage space.

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 1^{st}-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, i.e., $TRing(e) = TFan(v_i) \cup TFan(v_j)$. Figure 1 shows examples of $EFan(v)$, $VFan(v)$, $TFan(v)$ and $TRing(e)$.

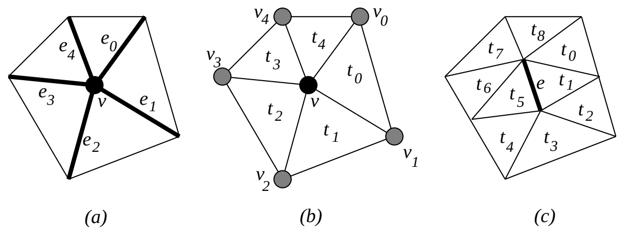


Figure 1: Examples of (a) $EFan(v) = \{e_0, \cdots, e_4\}$, (b) $VFan(v) = \{v_0, \cdots, v_4\}$ and $TFan(v) = \{t_0, \cdots, t_4\}$, and (c) $TRing(e) = \{t_0, \cdots, t_8\}$.

4 Vertex Tree Construction

To simplify the model, we apply an edge decimation method similar to the one that we proposed recently [14]. 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. 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. In the decimation process, the node list located at the first entry of the hash table, i.e., 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 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. This vertex fan 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. Since the triangles from neighboring triangle fans may be at different resolution levels, some of them may overlap each others. Using the edge collapsing 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. In order to be able to efficiently determine which triangles are at higher resolution during run-time, we introduce two state variables here, the *vertex state* and the *triangle state*. A vertex state is assigned to each vertex in the model to indicate the resolution level of the vertex while 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 zero. When an edge $e = (v_{child}, v_{parent})$ is chosen to collapse, we first identify a triangle t from $TRing(e)$ with the highest triangle state $S_{t,max}$ and then set the vertex states of the two vertices as follows:

$$S_{v_{parent}} = S_{v_{child}} = \begin{cases} 1 & \text{if } S_{t,max} = 0 \\ S_{t,max} & \text{otherwise} \end{cases}$$

The triangle states of $TRing(e)$ are then incremented by 1. Figure 2 shows an edge collapsing sequence and the corresponding change in triangle states and vertex states.

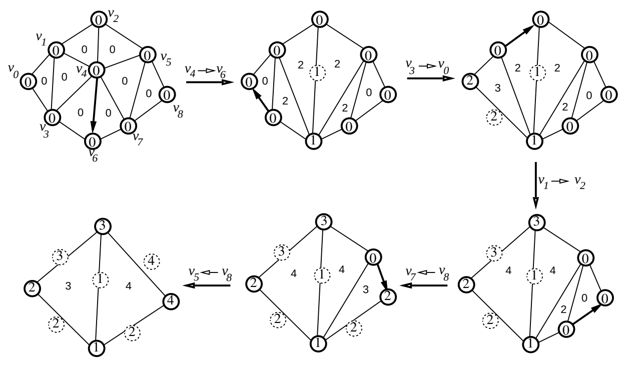


Figure 2: An edge collapse sequence and the corresponding change in triangle states and vertex states.

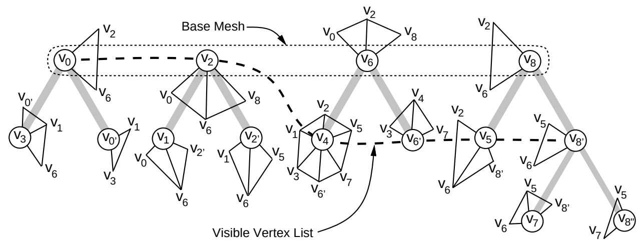


Figure 3: Triangle fans obtained from the vertex fans stored in the nodes of the vertex trees.

5 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/edges. With the new adaptive multi-resolution method, the order of transmitting the tree nodes can be determined during run-time 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 run-time, the server is responsible for selecting appropriate nodes for transmission to the client, while 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 an example in our discussion.

5.1 The Server Process

At the beginning of transmitting a model, the visible vertex list contains only the base mesh, i.e., $\{v_0, v_2, v_6, v_8\}$ of 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 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 replacing these vertices with their first visible parents, i.e., $\{v_1, v_3, v_7\}$ 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 node*. The other two are for the left and right child nodes called *left node* and *right node*, 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 shown as follows:

```
// the right child node
class RightNode {
public:
  unsigned short VID;          // Vertex ID
  struct {
     unsigned short close:1;   // 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 coord (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, which is determined by calculating a triangle budget value. This value is based on some view and animation parameters, such as object distance from the viewer, object moving 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 multi-resolution 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 others, 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 degenerated. 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 re-use the triangle list produced from the previous frame. When a triangle is inserted to the list, location of it in the list is stored at its corresponding node. This allows rapid modification of the triangles in the list. We also note down 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 non-zero 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 195MHz R10000 CPU and 256MB RAM, using several large polygonal models. Figures 4, 5, 6 and 7 demonstrate some results using the Terrain, Teeth, and Crater Lake models. In these figures, the 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 which 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 run-time can be very expensive. To simplify this testing, we precompute the longest edge of the edge fan as the radius of the bounding sphere for the node. During run-time, if the bounding sphere is found to overlap with the view region, the inside test is true.)

Table 1 shows the performance of preprocessing different models. The vertex tree construction time includes the calculation of importance values, edge decimation, 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 [11]. 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 are 24 bytes and 21 bytes, respectively. The difference is mainly due to the byte packing performed by the operating system. We experimented the transmission of the vertex trees from the server to the client through a 10Mbits Ethernet during the day time. Columns 4 and 5 of the table show the time needed to transmit the base nodes and the rest of the vertex trees, respectively. Note that

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

Table 1: Preprocessing performance of our method.

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 need 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, i.e., 0.2Mbits, the time needed to transmit the vertex trees through the Internet will be roughly 10 times longer than the figures shown in the table. Since 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 is 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 200MHz R5000 CPU and 128MB RAM. A walkthrough of the terrain shown in Figure 4(a) is performed and the average frame rate has increased from 2.47fps to 7fps. Note also that in our experiments, each node was sent as an individual package through the network. The transmission time can be further reduced if we transmit the nodes in batch as discussed in [8]. We would also expect a reduction in transmission time by integrating a geometric compression technique such as [6, 24]. Finally, column 6 of Table 2 shows the time for constructing the complete vertex trees at the client side.

As a summary, the method we propose here has the following advantages:

- Due to the minimal dependency between vertices, our method allows multi-resolution 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 [11], for example, most of the progressive records being transmitted may be irrelevant to what the viewer is seeing. Not only the bandwidth may be wasted, but also that the viewer may have to accept a lower visual quality for a longer period of time during the transmission of the surface model.

- Since 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 only visits small sections of a large model. The invisible sections may never need to be transmitted.

- The Internet is unreliable in transmitting information, and packages may be lost during the transmission. When using the progressive mesh method, if a progressive record is lost during the transmission, subsequent records received cannot be used until the lost record is retransmitted. This causes a round-trip delay, i.e., the client notifies the server of the lost

Model	Vertex trees Size		Progressive Transmission Time		Vertex trees Reconstruction Time
	Base Nodes	Child Nodes	Base Nodes	Child Nodes	
Terrain	7.6KB	2,459KB	0.02s	8.1s	1,74s
Bunny	16.3KB	1,377KB	0.01s	5.36s	0.97s
Teeth	36.4KB	1,113KB	0.03s	4.80s	0.92s
Crater Lake	19.2KB	163.7KB	0.00s	0.61s	0.10s

Table 2: Run-time performance of our method.

record and then the server retransmits the lost record. In our method, since edge collapses are independent of each others, the lost of a node during transmission will not affect subsequent nodes received.

7 Conclusions and Future Work

In this paper, we have presented a framework that allows progressive and selective transmission of adaptive multi-resolution models according to the view and animation parameters such as the user's view region and line of sight. The mechanism for transmitting the models has been discussed. The ability of performing adaptive multi-resolution 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 also presented the pre-processing and run-time performances of the new method.

Based on our adaptive method, we are currently working on a real-time multi-resolution modeling technique for textured and animated objects. We are also integrating the new method into our virtual walkthrough system [1, 2] 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.

Acknowledgements

This work is supported in part by the CityU Direct Allocation Grant, number 7100003-540.

References

[1] J. Chim, M. Green, R.W.H. Lau, H.V. Leong, A. Si, and A. Si. On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments. In *ACM Multimedia*, September 1998.

[2] J. Chim, R.W.H. Lau, A. Si, H.V. Leong, D. To, M. Green, and M.L. Lam. Multi-Resolution Model Transmission in Distributed Virtual Environments. In *ACM Symposium on Virtual Reality Software and Technology*, November 1998.

[3] P. Cignoni, C. Montani, and R. Scopigno. A Comparison of Mesh Simplification Algorithms. *Computers & Graphics*, **22**(1):37–54, 1998.

[4] P. Cignoni, E. Puppo, and R. Scopigno. Representation and Visualization of Terrain Surfaces at Variable Resolution. *The Visual Computer*, **13**:199–217, 1997.

[5] L. D. Floriani, P. Magillo, and E. Puppo. Efficient Implementation of Multi-Triangulation. In *Proceedings of IEEE Visualization '98*, October 1998.

[6] M. Deering. Geometry Compression. In *ACM Computer Graphics (SIGGRAPH'95)*, pages 13–20, August 1995.

[7] J. Falby, M. Zyda, D. Pratt, and R. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, **17**(1):65–69, 1993.

[8] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial Maps for Progressive Tansmission of Polygonal Surfaces. In *Proceedings of Symposium on VRML 98*, pages 25–31, 1998.

[9] S. Gumhold and Straβer. Real time Compression of Triangle Mesh Connectivity. In *ACM Computer Graphics (SIGGRAPH'98)*, pages 133–140, July 1998.

[10] P. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms. In *ACM SIGGRAPH'97 Course Notes*, August 1997. Available at http://www.cs.cmu.edu/∼ph.

[11] H. Hoppe. Progressive Meshes. In *ACM Computer Graphics (SIGGRAPH'96)*, pages 99–108, August 1996.

[12] H. Hoppe. View-Depndent Refinement of Progressive Meshes. In *ACM Computer Graphics (SIGGRAPH'97)*, pages 189–198, August 1997.

[13] H Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *Proceedings of IEEE Visualization '98*, pages 35–42, October 1998.

[14] R.W.H. Lau, M. Green, D. To, and J. Wong. Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology. *Presence: Teleoperators and Virtual Environments*, pages 22–35, February 1998.

[15] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. A. Turner. Real-Time Continuous Level of Detail Rendering of Height Fields. In *ACM Computer Graphics (SIGGRAPH'96)*, pages 109–118, August 1996.

[16] K. L. Low and T. S. Tan. Model Simplification using Vertex Clustering. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 75–81, April 1997.

[17] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *ACM Computer Graphics (SIGGRAPH '97)*, pages 199–208, August 1997.

[18] R. Pajarola. Large Scale Terrain Visualization using the Restricted Quadtree Triangulation. In *Proceedings of IEEE Visualization '98*, October 1998.

[19] J. Popović and H. Hoppe. Progressive Simplicial Complexes. In *ACM Computer Graphics (SIGGRAPH'97)*, pages 209–216, August 1997.

[20] E. Puppo and R. Scopigno. Simplification, LOD and Multiresolution Principles and Applications. In *Eurographics '97 Tutorial Notes*, page 104. Eurographics, 1997.

[21] J. Rossignac and P. Borrel. Multi-Resolution 3D Approximations for Rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, 1993.

[22] D. Schmalstieg and M. Gervautz. Demand-Driven Geometry Transmission for Distributed Virtual Environments. In *Proceedings of Eurographics '96*, pages 421–432, 1996.

[23] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence: Teleoperators and Virtual Environments*, 3(1):19–34, 1994.

[24] G. Taubin, A. Guéiec, W. Horn, and Lazarus. Progressive Forest Split Compression. In *ACM Computer Graphics (SIGGRAPH'98)*, pages 123–132, July 1998.

[25] J.C. Xia, J. El-Sana, and A. Varshney. Adaptive Real-Time Level-of-detail-based Rendering for Polygonal Models. *IEEE Transaction on Visualization and Computer Graphics*, **3**:171–183, 1997.

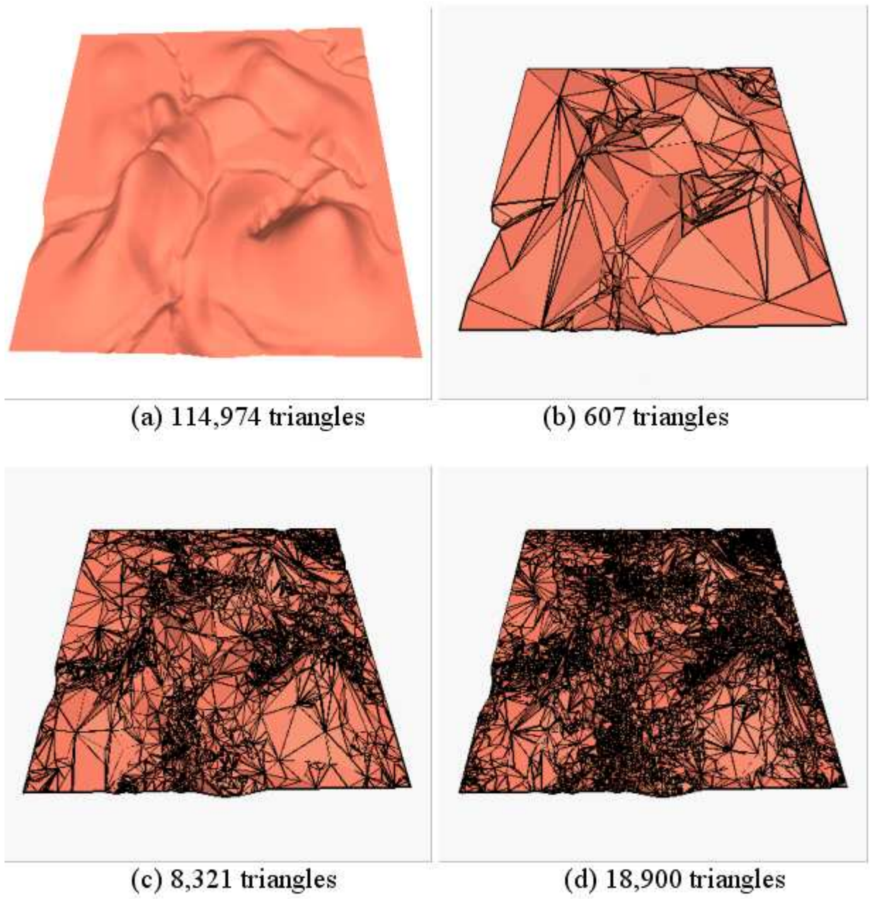
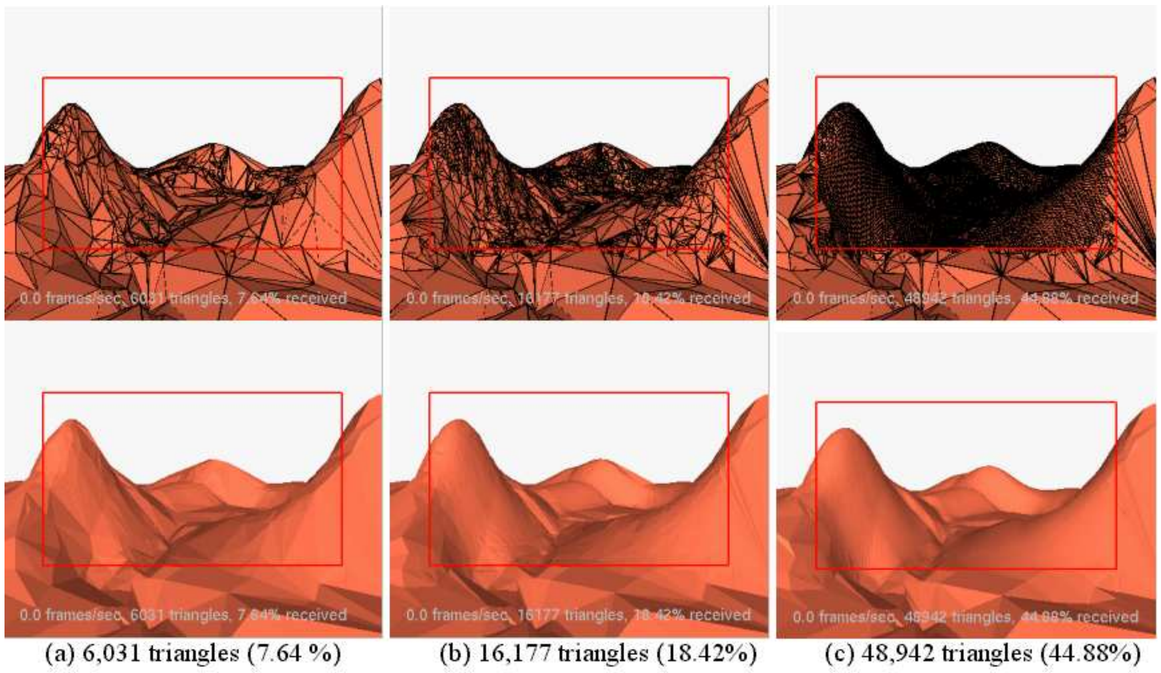


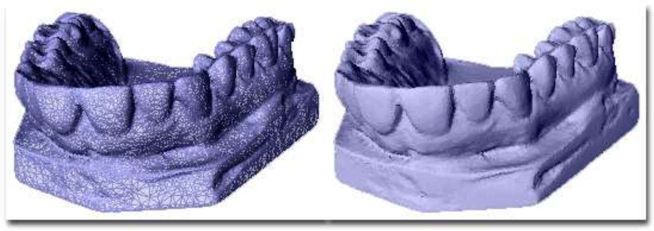
Figure 4: Adaptive refinement of a terrain model: (a) the original model, (b), (c) and (d) adaptively refined at different resolutions.



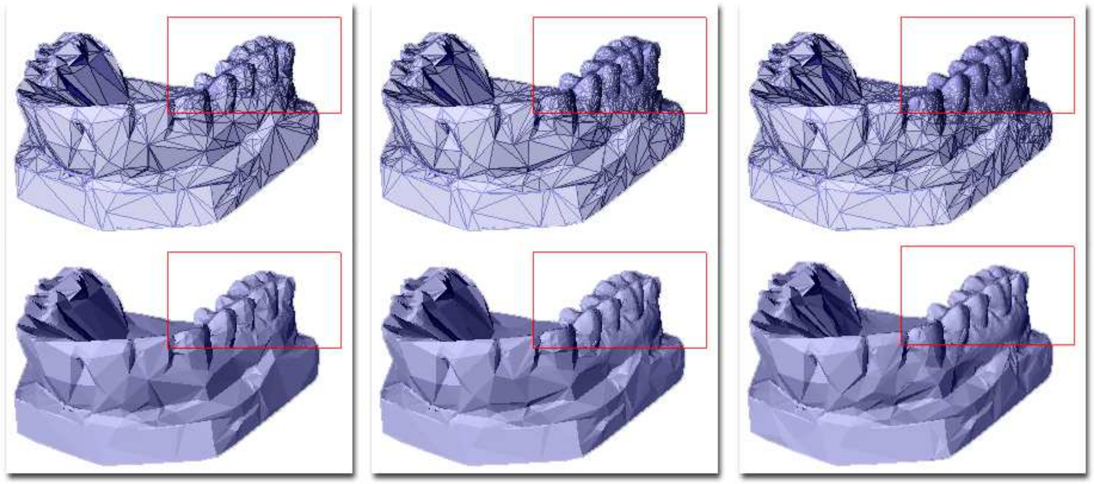
```
0.0 frames/sec, 6031 triangles, 7.64% received
0.0 frames/sec, 16177 triangles, 18.42% received
0.0 frames/sec, 48942 triangles, 44.88% received
0.0 frames/sec, 6031 triangles, 7.64% received
0.0 frames/sec, 16177 triangles, 18.42% received
0.0 frames/sec, 48942 triangles, 44.88% received
```

(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.

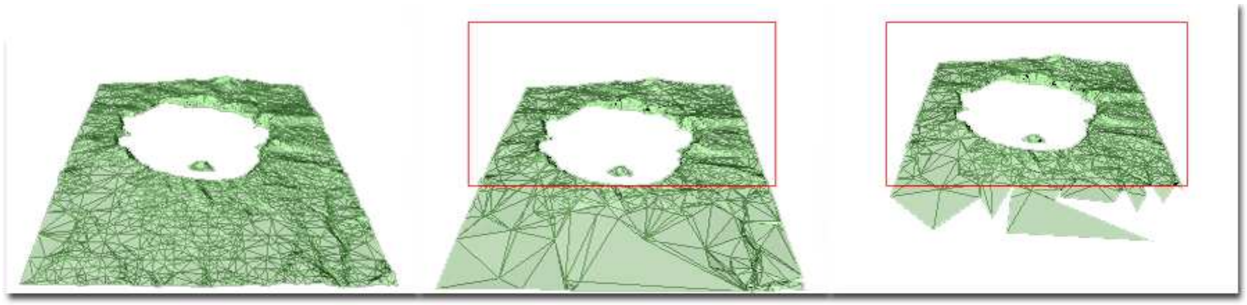


(a) 58,328 triangles



(b) 5,137 triangles (c) 8,750 triangles (d) 14,689 triangles

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



(a) 9,620 triangles (b) 4,353 triangles (c) 2,768 triangles

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.