A Progressive View-Dependent Technique for Interactive 3-D Mesh Transmission

Sheng Yang, *Student Member, IEEE*, Chang-Su Kim, *Member, IEEE*, and C.-C. Jay Kuo, *Fellow, IEEE*

***Abstract*—A view-dependent graphics streaming scheme is proposed in this work that facilitates interactive streaming and browsing of three-dimensional (3-D) graphics models. First, a 3-D model is split into several partitions. Second, each partition is simplified and coded independently. Finally, the compressed data is sent in order of relevance to the user's requests to maximize visual quality. Specifically, the server can transmit visible parts in detail, while cutting out invisible parts. Experimental results demonstrate that the proposed algorithm reduces the required transmission bandwidth, and provides an acceptable visual quality even at low bit rates.**

***Index Terms*—Graphics streaming, mesh partitioning, mesh simplification, progressive mesh coding, view-dependent.**

I. Introduction

RECENTLY, virtual reality has attracted a lot of attention, and the demand of three-dimensional (3-D) models has increased significantly. Triangle meshes have been widely used for modeling 3-D objects. Both geometry data and topology data are required to represent a triangle mesh. The geometry data specify vertex properties, for example, locations, colors, and normal vectors while topology data describe the connectivity between vertices. The mesh representation, however, requires a huge amount of storage space in general, and the distribution of 3-D mesh models over communication channels is limited by the available bandwidth. Mesh compression techniques must be used to reduce the storage requirement and the transmission bandwidth.

In this work, we consider a graphics streaming scenario, where graphics models are stored in a server and clients send requests to the server for data retrieval in order to browse one or many models from various viewing angles and distances. One application of this technique is to allow a viewer to browse a fine graphics model over a low bit rate channel with a simple device, say, wireless PDA. Most previous work on graphics compression has paid little attention to this server-client communication issue. In a traditional setting, a complete graphics model is transmitted either in a single or progressive resolution mode, yet no dynamic viewing is taken into account in the system. As a result, a viewer has to suffer a long period of initial download time. In the streaming context, transmission of invisible parts is actually unnecessary so that the initial waiting time can be greatly reduced.

In this research, we propose a view-dependent graphics streaming system based on viewing and network parameters. The system can adaptively transmit graphics models according to user's requests and network constraint. The view-dependent progressive mesh (VDPM) compression algorithm first divides an input mesh into several partitions, and compresses each partition progressively and independently. According to the viewing parameters provided by a client, the server assigns an appropriate resolution to each partition, and then reorganizes and transmits topology and geometry data to maximize visual quality subject to a bandwidth constraint. Experimental results demonstrate that the proposed algorithm can offer desirable visual quality even at low bit rates by allocating the major portion of the available bandwidth to visible parts.

The rest of the paper is organized as follows. The system overview is introduced in Section II. In Section III, we present our techniques for mesh partitioning, simplifying and coding. In Section IV, we describe the details involved in visibility determination and resolution setting. Subsequently, we address how to allocate bandwidth to partitions subject to a network constraint in Section V. Experimental results are reported in Section VI to verify the performance of the proposed algorithm. Finally, concluding remarks and future work are given in Section VII.

II. System Overview

When a viewer browses a 3-D mesh model, the model need not be rendered at the full resolution over its entire surface, since the most urgent data are those that refine the visible regions. Therefore, surface partitioning is needed to allow visible regions being separated from invisible ones. To be more specific, to enable adaptive browsing of 3-D graphics models, these models should be divided into partitions so that each partition can be coded and transmitted independently. Our basic idea is illustrated in Fig. 1.

As shown in Fig. 1(a), the encoded bit stream consists of a base model and several layered partitions. Fig. 1(b) gives an example of view-dependent transmission. The client informs the server of its viewing parameters. Then, the server transmits each partition with an appropriate resolution. Typically, it transmits visible partitions at a higher resolution, while transmits few data for invisible partitions.

It is important for the server to determine the visible regions from the view point of viewers before transmission. This depends on the viewing parameters and the visibility determination algorithm. Here, the relative locations of the viewer and the

Manuscript received January 30, 2002; revised April 3, 2003. This paper was supported in part by the Integrated Media Systems Center, National Science Foundation Engineering Research Center, Cooperative Agreement EEC-9529152. This paper was recommended by Associate Editor H. Shum.

S. Yang and C.-C. J. Kuo are with the Department of Electrical Engineering and the Integrated Media Systems Center, University of Southern California, Los Angeles, CA 90089-2564 USA (e-mail: sheng@costard.usc.edu; cckuo@sipi.usc.edu).

C.-S. Kim is with the Department of Information Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: cskim@ieee.org).

Digital Object Identifier 10.1109/TCSVT.2004.835153

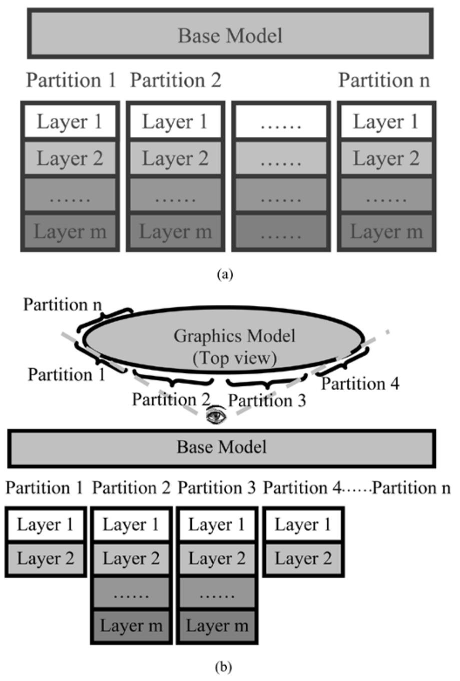


Fig. 1. (a) Data organization of the compressed model. (b) Example of view-dependent transmission.

object and the viewing direction are used to determine the visibility and the proper resolution of each partition. Furthermore, the actual data transmission is also affected by the available network bandwidth.

The proposed graphics streaming system is shown in Fig. 2. It consists of three main modules: 1) mesh representation; 2) visibility determination and resolution setting; and 3) rate control. Each module will be described in detail in Sections III–V.

III. Mesh Representation

A. Background Review

Since there is a large amount of literature on mesh representation, let us first review previous work in this field. Then, we will describe our novel mesh representation scheme which takes mesh partitioning into account.

Since Deering [5] introduced the concept of generalized triangle strip, much effort has been made in the single-resolution mesh compression technique. Taubin and Rossignac [18] developed the topological surgery algorithm, which compresses triangle meshes by using two interlocked trees, called the vertex spanning tree and the triangle spanning tree. Touma and Gotsman [19] proposed the use of the breadth-first traversal method and the parallelogram rule to compress topology and geometry data of triangle meshes, respectively. Also, Gumhold and Straßer [7] developed a fast compression algorithm for topology data, which can be employed in real time applications.

These single-resolution algorithms encode the entire data as a whole. It is however advantageous to progressively compress and transmit the graphics models streaming over IP networks. If the scene is rendered or played after the reception of the entire data, viewers have to wait for a long time. On the contrary, if the data are progressively transmitted, the graphics models can be updated from low to high resolutions continuously at the decoder, alleviating the annoying waiting experience.

Hoppe [8] proposed the notion of progressive meshes based on edge collapse and vertex split operations. The progressive mesh enables continuous transition from the coarsest to the finest resolutions. Cohen *et al.* [3] introduced the simplification envelope method, which generates a hierarchy of level-of-detail approximations from a given polygonal model. Li and Kuo [11] developed a hierarchical 3-D graphics compression scheme to progressively compress an arbitrary triangle mesh into a single bit stream. The receiver can stop at any point within the bit stream to reconstruct the original model with a rate-distortion tradeoff. Khodakovsky *et al.* [10] investigated an alternative progressive coding method by employing the wavelet transform after converting an input mesh into a semi-regular mesh. Alliez and Desbrun [1] proposed a progressive scheme with an excellent compression ratio, which provides the state-of-the-art compression performance. However, these algorithms do not exploit the fact that only the front parts of 3-D models are visible to the viewer and the transmission of invisible parts is actually a waste of the limited bandwidth.

To exploit the dynamic visual importance, several view-dependent rendering algorithms have been proposed in the literature [9], [13], [16], [21]. These view-dependent methods organize a given mesh into a hierarchy and continuously query the hierarchy to generate a set containing only visible primitives. They work well when dealing with mesh data without entropy coding. This setting is likely to happen in an environment where all data are stored locally and a compact description of graphics data is not an essential requirement. They are however not suitable for compressed graphics data streaming. Note that the 3-D mesh model is usually encoded by entropy coders at the last stage to reduce the file size. Then, the coded bits in the resulting bit stream cannot be easily reorganized to take advantage of selective transmission when the viewing parameters are changed dynamically. Some partitioning schemes are proposed in [15], [22] for error resiliency or large model management. In this paper, we develop a new partitioning scheme to facilitate view-dependent graphics data transmission, which is different from the work in [15], [22] for the overall objective and, consequently, the detailed implementation. The proposed mesh compression algorithm is able to transmit visible entropy-coded mesh data according to the dynamic change of viewing parameters.

B. Partitioning

To enable view-dependent compression and transmission of a given graphics model, the proposed algorithm divides a mesh into several partitions, and simplifies each partition independently. Fig. 3 illustrates the mesh partitioning and simplification procedures. Let us assume that the original mesh is divided into three partitions as shown in Fig. 3(a), where partitioning boundaries are drawn with thick lines. The proposed algorithm simplifies the mesh into a base model by merging inner vertices of each partition into a single vertex as shown in Fig. 3(b).

The following definitions are useful in developing the proposed algorithm. A *boundary node* is a vertex that lies on partition boundaries, and an *inner vertex* is a vertex that is not a boundary node. A boundary node is also called a *corner vertex*,

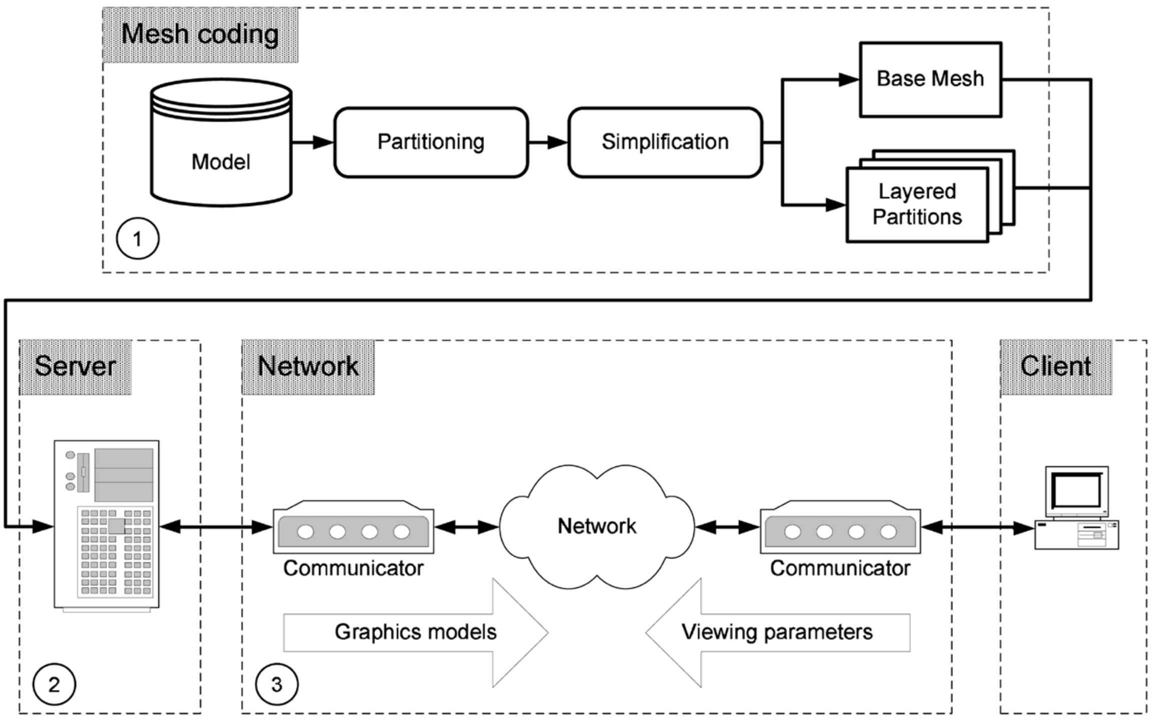


Fig. 2. Overview of the proposed graphics streaming system. (1) mesh representation; (2) visibility determination and resolution setting; and (3) rate control.

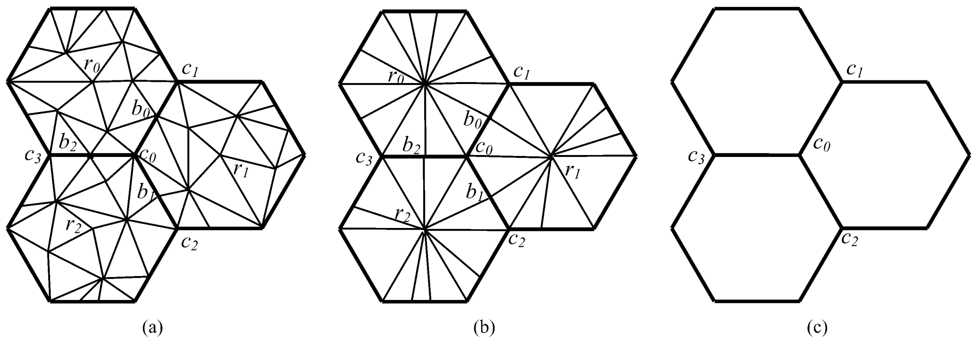


Fig. 3. Mesh partitioning and simplification. (a) Original mesh. (b) Simplified mesh. (c) Final polygonal mesh.

if it is incident on three or more partitions, or if it lies on the physical boundary of the object and is incident on two or more partitions. A boundary node, which is not a corner vertex, is called a *boundary vertex*. A *root vertex* is a vertex to which all inner vertices of a partition are eventually merged during the simplification process. For example, as shown in Fig. 3, b_0, b_1, and b_2 are boundary vertices, c_0, c_1, and c_2 are corner vertices, and r_0, r_1, and r_2 are root vertices, respectively.

A *boundary loop* is a loop connected by a series of boundary nodes. If there is one and only one root vertex within each boundary loop of a base model, we call it a *perfect base model*, as shown in Fig. 3(b). Fig. 4(a) shows an example of an imperfect base model. Note that b_0, b_1, and c_0 form a boundary loop, but there is no root vertex within it. The proposed scheme generates a perfect base model, since it yields a higher coding gain as will be shown in Section III-C. Note that a hole is naturally declared as a partition and the vertices surrounding it form a boundary loop. We can imagine there is a virtual root located at the center of the hole.

To reduce the initial waiting time while maintaining an acceptable visual quality of the base model, there is a tradeoff in

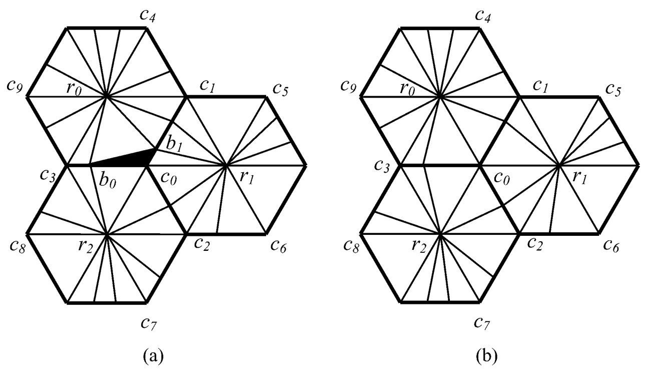


Fig. 4. (a) Imperfect base model. (b) Eliminating the irregular loop in (a).

determining how many partitions a model should be split into. The curves describing how the base mesh size and the distortion vary with the partition number are shown in Fig. 5. The distortion is obtained by computing the mean distance between the base mesh and the original one [2]. The base mesh size linearly increases with the partition number, while the distortion decreases exponentially. Other models also comply with these

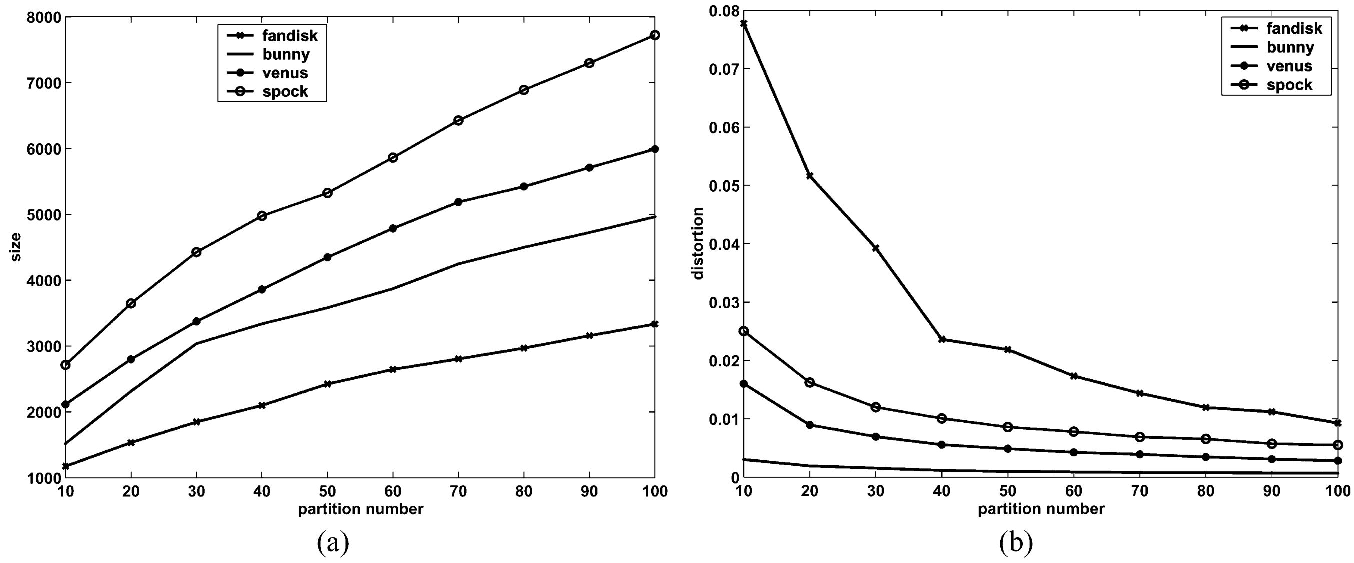


Fig. 5. Relationship between partition number and (a) base mesh size and (b) base mesh distortion.

curves in the general trend. Moreover, if we ignore the geometry data of the base mesh, the base mesh size is dependent of the partition number only, no matter how large the original model is, as described in Section III-D1. According to our experience, it is appropriate to split a given model into 40–60 partitions in most applications.

In our algorithm, root vertices are first selected from a given mesh, and then each partition is expanded from a root vertex. As shown in Fig. 3(b), only root vertices and partitioning boundaries are retained to generate the base model. Therefore, the visual quality of the base model depends on the locations of root vertices. To select root vertices, we adopt the maximum distance method, which is used for the codebook design in vector quantization [20]. The basic notion is to maximize the base mesh visual quality by making the distance between two root vertices as far as possible, as detailed below.

An arbitrary vertex is selected as the first root vertex r_0. Then, the distances between this vertex and all other vertices are compared. The vertex with the maximum distance is chosen as the second root vertex r_1. Similarly, the vertex that has the maximum distance from the vertex set $\mathcal{R} = \{r_0, r_1, \ldots, r_{i-1}\}$ is selected as the new root vertex r_i. The distance from a vertex v to the set of vertices $\mathcal{R}$ is defined as

$$d(v, \mathcal{R}) = \min(||v - r_0||, ||v - r_1||, \ldots, ||v - r_{i-1}||)$$

where $|| \cdot ||$ denotes the Euclidean norm in the 3-D space. This step is repeated until a prespecified number of root vertices have been selected. The Euclidean distance is adopted here due to its simplicity and practicability for our test meshes. However, the adoption of the Euclidean norm may fail to achieve this "equal size partitioning" goal around areas where the mesh undulates very much. For such a mesh, other distance measures such as the geodesic measurement may be considered.

After determining the set of root vertices, the proposed algorithm extends each root vertex to a partition by using bidirectional breadth-first mesh traversal algorithm extended from [12]. To assign each vertex v to the partition whose root vertex

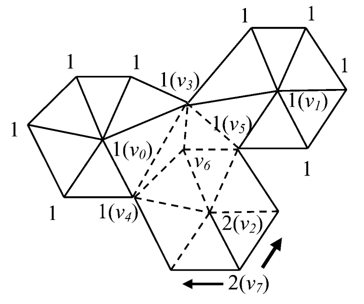


Fig. 6. Bidirectional traversal.

is nearest to v, we maintain a first-in–first-out (FIFO) list of vertices. The principal link of vertex v is defined as the edge, incident on v, from which the traversal starts. First, we select arbitrarily the principal links of root vertices, and push all root vertices into the list in order of their indices. Then, a vertex w is popped from the list to perform the traversal. We check its adjacent vertices in a bidirectional way, starting from the principal link. If an adjacent vertex w' has not been traversed yet, then it is assigned the same partition index as vertex w, and is pushed into the buffer. The principal link of w' is set as edge (w, w'). The process is iteratively performed, until the list becomes empty.

For example, in Fig. 6, let us assume that vertices v_0 and v_1, both of which belong to partition 1, have already been traversed, and that all their adjacent vertices are assigned the same partition index 1. Also, assume that v_2, which belongs to partition 2, is popped from the FIFO list. The traversed links are depicted in solid lines, and the untraversed links in dash lines. If we assign partition index 2 to the untraversed vertex v_6, which is located between v_4 and v_5 in partition 1, the boundary will be of a saw-toothed shape. This is undesirable, since the irregular shape of partition boundary degrades the compression performance. To avoid this undesirable effect, we perform the bidirectional traversal in the following way. We first check the adjacent vertices of v_2 from the principal link (v_2, v_7) in a counterclockwise order until v_5 that belongs to another partition is reached. Then, we repeat the procedure in a clockwise order. Vertex v_6 remains untraversed, until v_3 is popped from the FIFO list later.

TABLE I
STATISTICS OF FOUR TEST BASE MODELS. THE NUMBER OF ROOTS (OR PARTITIONS), CORNER VERTICES, BOUNDARY VERTICES, IRREGULAR LOOPS, THE TOTAL NUMBER OF VERTICES, THE FILE SIZE, AND THE DISTORTION

model	partitions	corner vertices	boundary vertices	irregular loops	total vertices	size (Bytes)	distortion
Fan Disk	50	88	373	56	566	2422	0.021859
Bunny	50	94	711	76	930	3581	0.000982
Venus Head	50	90	1046	55	1240	4347	0.004871
Spock	50	95	1472	58	1675	5323	0.008589

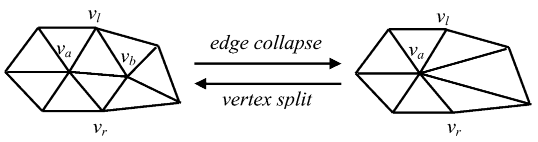


Fig. 7. Illustration of the half edge collapse operation.

After assigning a partition index to each vertex, we detect boundary nodes. Let $\mathcal{P}_i$ be the ith partition. We check each vertex in $\mathcal{P}_0$. A vertex is declared as a boundary node, if its adjacent vertex has the partition index different from 0. Similarly, we check each vertex in $\mathcal{P}_i$, $1 \leq i < N$. A vertex in $\mathcal{P}_i$ is declared as a boundary node, only if its adjacent vertex v has the partition index different from i, and v has not been detected as a boundary node in previous steps.

Finally, the given mesh is partitioned into a Voronoi space in terms of the topological distance (rather than the Euclidean norm). This guarantees that vertices can be averagely assigned to partitions.

C. Simplification

We employ the half edge collapse operation [6] to merge all inner vertices of a partition eventually into the root vertex. The half edge collapse operation is widely used in mesh compression, since the position of a merged vertex need not be encoded. It is worthwhile to point out that the view-dependent compression algorithm is not limited to the use of the edge collapse operation. Any simplification method can be adopted under the view-dependent framework, as long as it can achieve layer simplification within each partition.

Fig. 7 illustrates the half edge collapse operation, which merges vertex v_b to its adjacent vertex v_a. Two adjacent triangles (v_a, v_b, v_l) and (v_a, v_b, v_r) are removed accordingly. Vertices v_a, v_l, v_r, and v_b are called the parent, left, right, and child vertices, respectively. The vertex split is the inverse operation of the half edge collapse, used to refine a mesh. v_a, v_l and v_r are required to uniquely identify an edge collapse operation. To improve the compression performance, it is desirable to record the relative locations of v_l and v_r in the incident vertex list of v_a, instead of their absolute indices. Therefore, v_a cannot be a boundary node due to this concern; otherwise, the partitions cannot be simplified and coded independently.

Accordingly, we perform successive edge collapse operations within a partition until all inner vertices are removed, subject to the constraint that only inner vertices can be merged to inner or root vertices. We first compute the quadric error metrics (QEM) [6] for each edge. Then, edge (v_1, v_2) is selected to be decimated, if (v_1, v_2) has the minimum error.

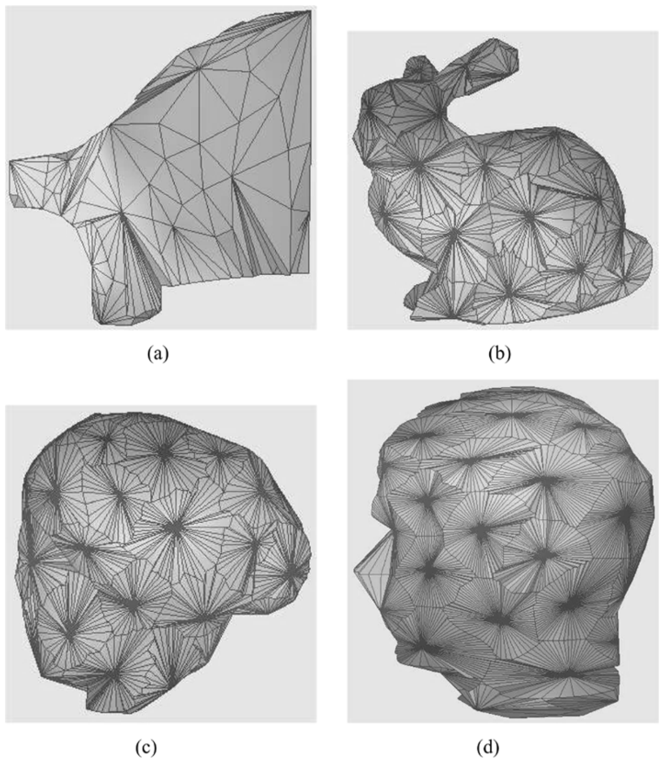


Fig. 8. Four base models.

The proposed algorithm encodes the base model, assuming that the simplified mesh is a perfect base model. However, even after all inner vertices have been merged into root vertices, there can be irregular boundary loops as shown in Fig. 4(a). In this example, triangle (b_0, b_1, c_0) is an irregular boundary loop, since there is no root vertex within it. In our approach, these irregular loops are removed using a postprocessing technique. That is, we detect irregular boundary loops by checking the three vertices of each triangle in the simplified mesh. If all three vertices are boundary nodes, they form an irregular loop. Then, we perform one edge collapse operation to eliminate one of the three boundary nodes. But the child vertex is not restricted to be merged to the root vertex. The operation is permitted as long as the irregular loop can be removed. The irregular loop in Fig. 4(a) can be eliminated by merging b_1 into c_0, as shown in Fig. 4(b).

Some statistics of four test base models are reported in Table I. They include the number of partitions (or roots), corner vertices, boundary vertices, irregular loops, the total number of vertices, the file size and the distortion. The four base models are depicted in Fig. 8.

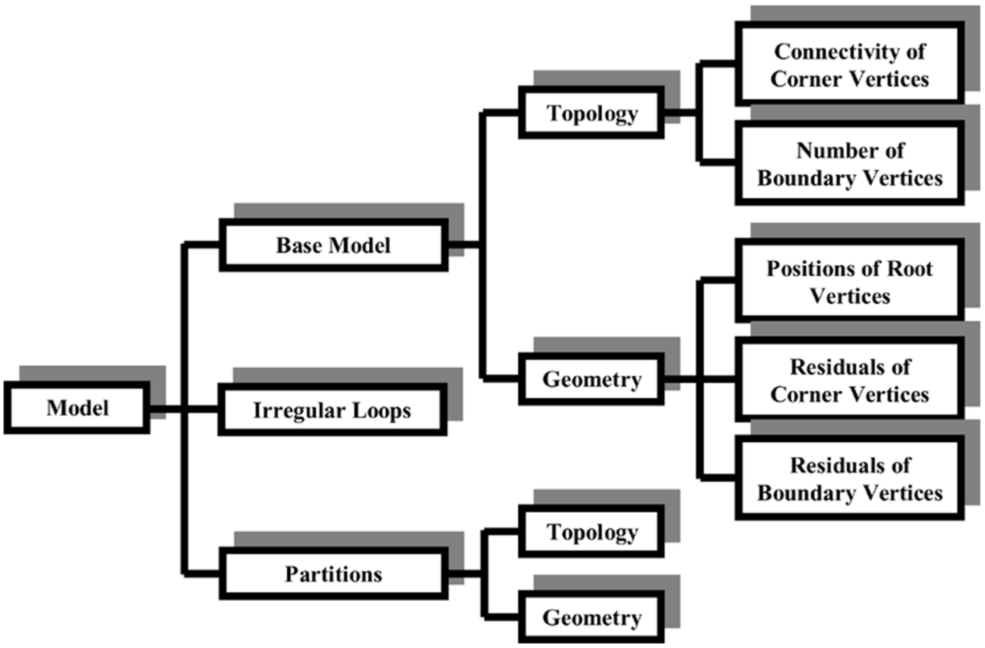

Model
Base Model
Irregular Loops
Partitions
Topology
Geometry
Connectivity of Corner Vertices
Number of Boundary Vertices
Positions of Root Vertices
Residuals of Corner Vertices
Residuals of Boundary Vertices
Topology
Geometry

Fig. 9. Data structure of a graphics model.

D. Progressive Coding

1) Coding of the Base Model: The previous section described how to generate a perfect base model from a given mesh. The perfect base model has two properties, which can be exploited to effectively encode its topology data. First, a boundary vertex is always connected to two root vertices and two boundary nodes, as shown in Fig. 3(b). Thus, to preserve the topology, it is sufficient to record the number of boundary vertices between every two corner vertices. Second, since each boundary loop contains a single root vertex, we can remove root vertices without losing any topology information.

Fig. 3(c) shows the final polygonal mesh, which is obtained by removing the boundary and root vertices from the perfect base model in Fig. 3(b). The topology of the perfect base model can be losslessly reconstructed from that of the final polygonal mesh and the number of boundary vertices between every two corner vertices. This is why we claim that the size of the base mesh topology is only dependent of the partition number in Section III-B. Finally, the topology of the final polygonal mesh is encoded by the breadth-first mesh traversal algorithm [12].

After encoding the topology of the base model, we quantize and encode the geometry. We take vertex positions as illustration, which are encoded in the order of the root, corner, and boundary vertices. The positions of root vertices are entropy-encoded by a QM coder [17]. It is expected that a corner vertex c_i is located near the center of its adjacent root vertices. Thus, we predict its position $P(c_i)$ via

$$\hat{P}(c_i) = \frac{1}{k}\sum_{j=0}^{k-1} P(r_{ij})$$

where r_{ij} denotes an adjacent root vertex, and k is the number of adjacent root vertices. Then, the prediction error, $P(c_i) - \hat{P}(c_i)$, is encoded by the QM coder. Similarly, the position of a boundary vertex is predicted from those of two corner vertices, and the prediction error is encoded. Let us assume that there are l boundary vertices b_i $(0 \le i < l)$ between two corner vertices c_0 and c_1. $c_0, b_0, b_1, \ldots, b_{l-1}, c_1$ sequentially lie on a boundary segment. Then, the position of b_i is estimated by linear interpolation, given by

$$\hat{P}(b_i) = P(c_0) + \frac{i+1}{l+1}[P(c_1) - P(c_0)], \qquad i = 0, 1, \ldots, l-1.$$

2) Coding of Partitions: Before encoding each partition, the irregular loops must be recorded to preserve the topology data. Since an irregular loop is eliminated by an edge collapse operation, it can be uniquely recorded by the parent, left, and right vertices that determine this edge collapse operation. Thereafter, each partition is encoded independently by using the conventional progressive mesh coding techniques. We adopt the compressed progressive mesh (CPM) method [14] to represent each partition into N layers

$$M_0 \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots \longrightarrow M_{N-1}$$

where M_0 is the coarsest mesh partition, already transmitted as parts of the base model, and M_{N-1} is the finest mesh partition. In this work, the index i in M_i is called the resolution of the partition.

The data structure of a graphics model is illustrated in Fig. 9. The order in which the data are coded and transmitted is stated as follows: connectivity of corner vertices, positions of root vertices, residual of corner vertices, number of boundary vertices, residual of boundary vertices, irregular loops, topology of partitions, and geometry of partitions.

IV. Visibility Determination and Resolution Setting

A. Visibility Determination

In this section, we describe a method to determine the visibility of partitions. This is the basis of view-dependent transmission. The method uses three criteria: viewing frustum, partition orientation, and occlusion.

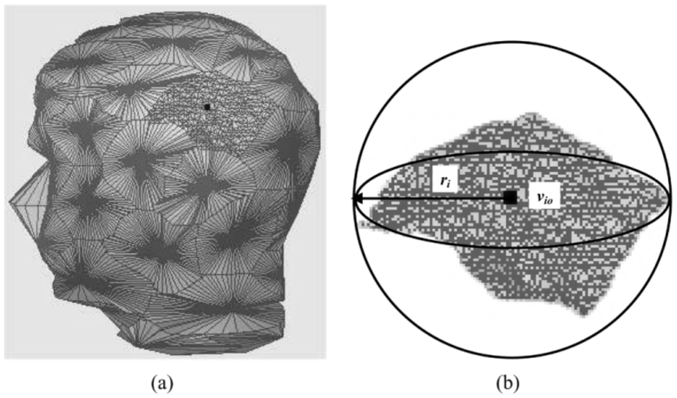


Fig. 10. Use of a sphere to represent a partition. (a) Partitioned Spock model. (b) Sphere.

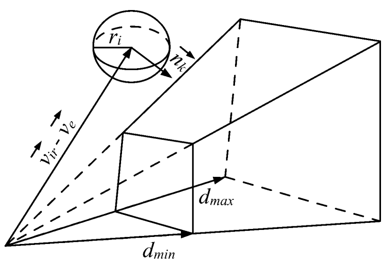


Fig. 11. Viewing frustum test.

1) Viewing Frustum: The viewing frustum defines a volume, and the viewer can only perceive objects located in it. The viewing frustum is enclosed by a near plane, a far plane, and four side planes intersected at the view point. The purpose of this criterion is to cut out the partitions which are located out of the viewing frustum.

The test can be done by checking whether the intersection between the viewing frustum and the partition exists. To simplify the process, we use a sphere to enclose vertices within one partition. Let us take the Spock model as an example. The partitioned Spock model is drawn in Fig. 10(a). Note that only the partition of interest is denoted by P_i, which is rendered at the highest resolution. The root vertex of P_i is represented by a black dot and the vertices within P_i is surrounded by a sphere as shown in (b) with its center located at root vertex v_{io}. The radius r_i can be computed via

$$r_i = \max_{v_{ij}\in P_i}\{|v_{ij} - v_{io}|\}$$

where v_{ij} is any inner vertex within P_i.

The viewing frustum is shown in Fig. 11. The normal vectors of four side planes are denoted by $\vec{n}_k$, $k = 1,\ldots,4$. The distance from the view point to the near plane and far plane are denoted by $d_{\min}$ and $d_{\max}$, respectively. Then, the sphere lies outside of the frustum if and only if

$$(\vec{v}_{ir} - \vec{v}_e)\cdot\vec{n}_k < -r_i$$

where $\cdot$ denotes the inner product, and

$$|\vec{v}_{ir} - \vec{v}_e| - r_i < d_{\min} \text{ or } |\vec{v}_{ir} - \vec{v}_e| + r_i > d_{\max}.$$

By using the above criterion, some experimental results are shown in Fig. 12, where (a) is the image reconstructed by the partitions located in the viewing frustum and (b) is the image of the whole model. Comparing these two images, we can see that the details of partitions located out of the viewing frustum are skipped.

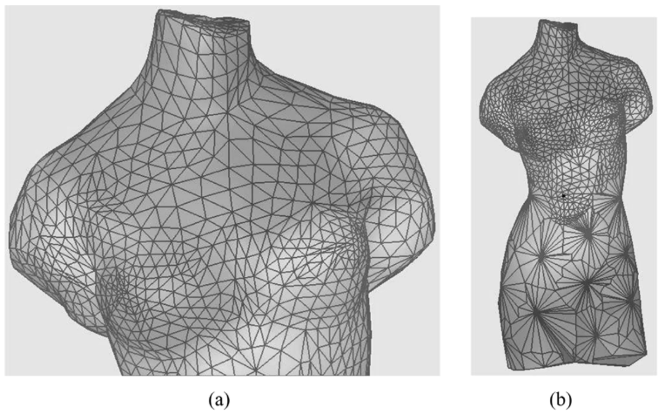


Fig. 12. Visibility determination based on the viewing frustum criterion. (a) Partial image. (b) Whole image.

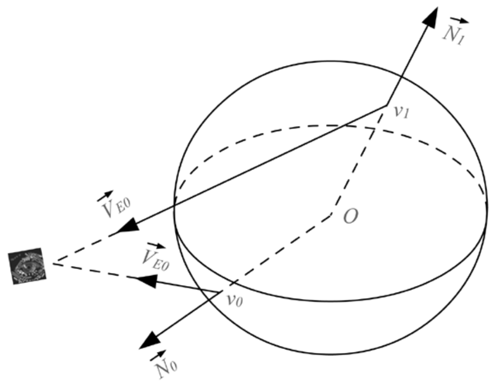


Fig. 13. Determining the visibility of a vertex.

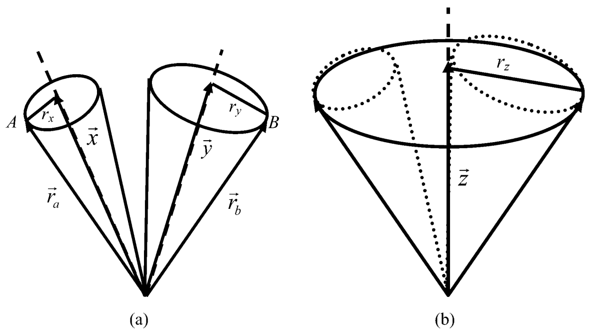


Fig. 14. Merging of two Gauss maps. (a) Two Gauss maps. (b) Their mergence.

2) Partition Orientation: The purpose of the partition orientation test is to cut out partitions facing away from the viewer. The way to determine whether a vertex v is facing toward the viewer or not is illustrated in Fig. 13. Let $\vec{V}_E$ be the vector pointing from v to the viewer, and $\vec{N}$ be the normal vector of v. The angle θ between $\vec{V}_E$ and $\vec{N}$ is computed via

$$\theta = \arccos\left(\frac{\vec{V}_E\cdot\vec{N}}{|\vec{V}_E|\cdot|\vec{N}|}\right).$$

If $\theta \le (\pi/2)$, v_0 is visible. Otherwise, it is invisible.

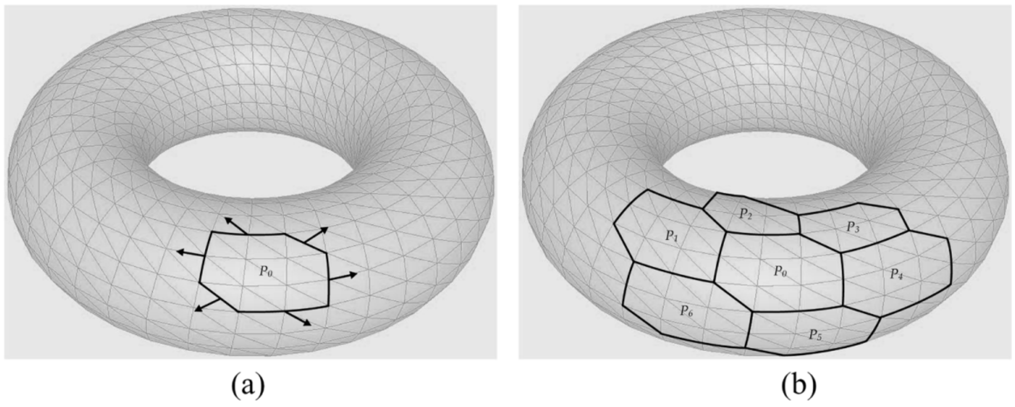


Fig. 15. Occlusion check.

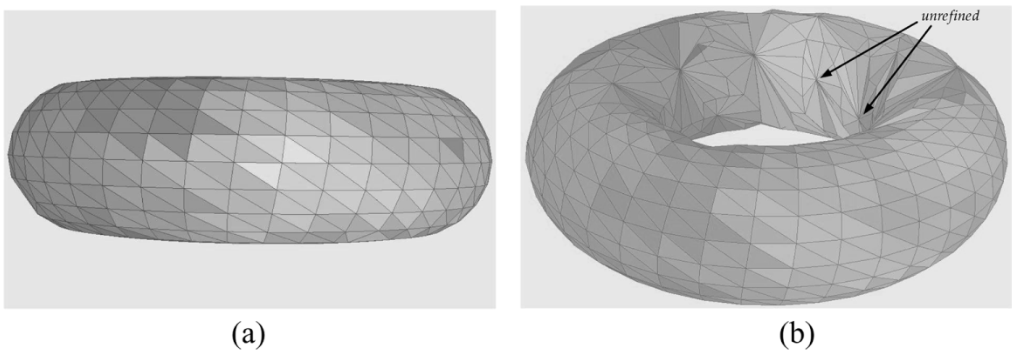


Fig. 16. Example of the occlusion effect.

To find out the overall normal vector (or the orientation) of vertices in a partition, the Gauss map [21] provides an effective method. By using the Gauss map, the unit normal vectors are mapped to the corresponding points on the surface of a unit sphere, and all vectors are represented by a cone. In our approach, we obtain the Gauss map for each partition, and employ the axis of the cone as the representative normal vector for that partition.

In mesh simplification, each partition is implicitly organized as a hierarchical tree, where the root vertex is the root of the tree, and the edge collapse operation determines the parent-child relationship. Thus, we can obtain the Gauss map of the partition in a bottom-up approach. We start with the trivial Gauss maps of leaf nodes or vertices, whose radius are 0, and iteratively obtain the Gauss map of the parent node by merging the Gauss maps of child nodes. Fig. 14 illustrates the merging of two Gauss maps. Let $\vec{x}$, $\vec{y}$ and r_x, r_y denote the axes and the radii of two child cones before merging, respectively. For the derivation of $\vec{z}$ and r_z, we refer to [23].

Consequently, the angle between the representative normal vector of the partition and the viewing vector can be calculated as

$$\theta = \arccos\left(\frac{\vec{V}_E \cdot \vec{z}}{|\vec{V}_E| \cdot |\vec{z}|}\right).$$

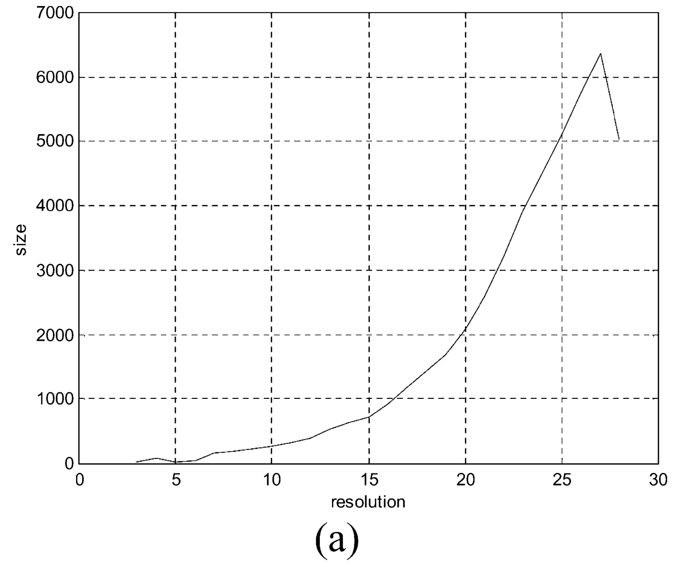
If $\theta - (\pi/2) \leq \arcsin(r_z)$, the partition is visible. Otherwise, it is invisible.

3) Occlusion Effect: In some cases, a viewer cannot observe a partition even if it is located in the viewing frustum and facing toward the viewer. This is due to the self-occlusion property of concave objects. When rendering, the occlusion effect can be easily detected with the Z-buffer. However, it is difficult for the server to adopt a similar approach in the current context. We develop a new method to check occlusion as described below.

We first determine the set, denoted by U, that contains all partitions that are located in the viewing frustum and facing toward the viewer. Then, the partition in U that is the closest to the viewer is taken out, denoted by P_0, as shown in Fig. 15(a). P_0 is visible and we store it in another set V. We check the partitions adjacent to P_0. If they belong to U, we store them in V as shown in Fig. 15(b). Repeat the traversing process, until the visible region containing P_0 has been found. If there are partitions remaining in U, we repeat the same procedure subject to the constraint that only when the partition is not occluded by known visible regions, will it be stored in V. Eventually, all visible regions can be found. An example is shown in Fig. 16, where (a) is the image rendered from the viewer's perspective. We can see from (b) that the occluded parts are kept at the coarse resolution.

B. Resolution Setting

An edge of unit length, which is perpendicular to the normal vector $\vec{N}$, is seen as the edge of length $\cos\theta$ by the viewer, where θ is the angle between the normal vector and viewing vector. Thus, the resolution of the surface of an object should be proportional to $\cos\theta$. The distance between the viewer and the object also plays an important role in resolution setting. The closer a viewer is to the object, the more details he can see, and vice versa. Let us describe how to determine the resolutions of partitions in Sections IV-B1 and 2.



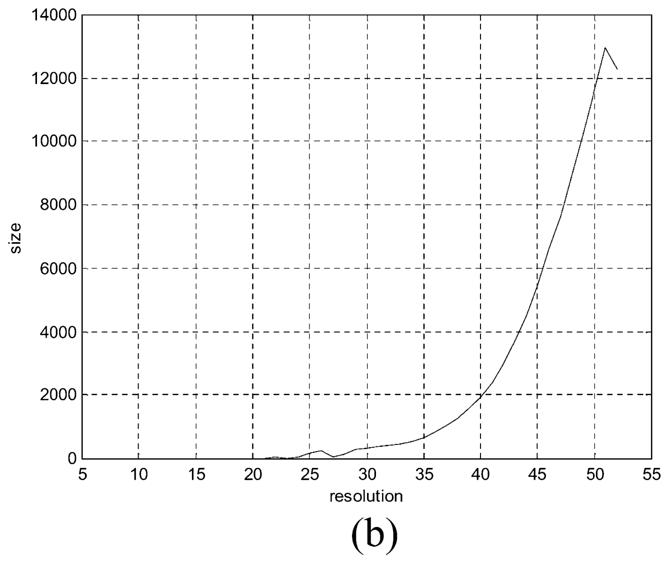


Fig. 17. Amount of data needed at each resolution for: (a) the Spock model and (b) the Bunny model.

1) Viewing Angle: Generally speaking, human eyes are more sensitive to light beams coming into eyeball at a smaller angle with respect to the line of sight. Therefore, the overall normal vector of partition obtained in Section IV-A2 is helpful in expressing the relationship between the resolution and the viewing angle.

We determine the resolution r_θ of a partition by using the viewing vector $\vec{V}_E$ and the Gauss map $\mathcal{G}$ of the partition via

$$\begin{aligned}\theta_{\min} &= \min_{\vec{N}\in\mathcal{G}}\left\{\arccos\left(\frac{\vec{V}_E\cdot\vec{N}}{|\vec{V}_E|\cdot|\vec{N}|}\right)\right\}\\ r_\theta &= r_{\max}\cdot\cos(\theta_{\min})\end{aligned} \quad (1)$$

where $r_{\max}$ is the highest resolution of the partition. Angle $\theta_{\min}$ is the minimum angle between the viewing vector and any possible vector in the Gauss map. Thus, an edge of unit length in the partition is perceived by the viewer as the edge whose length is shorter than $\cos(\theta_{\min})$. Therefore, the resolution of the partition is set to be proportional to $\cos(\theta_{\min})$.

Furthermore, silhouette plays an important role in the perception of details. However, by (1), the center partitions have higher resolutions than side ones. This will inevitably blur the silhouette. To overcome this drawback, we add the complementary condition

$$\theta = \arccos\left(\frac{\vec{V}_E\cdot\vec{N}}{|\vec{V}_E|\cdot|\vec{N}|}\right)$$

if there exists $\vec{N}$ belonging to $\mathcal{G}$, which makes $\theta = (\pi/2)$, then we have $r_\theta = r_{\max}$.

2) Viewing Distance: To find out the relationship between the distance and the resolution, we first study the amount of data needed for each resolution, which is shown in Fig. 17. The data-fitting curve approaches a parabolic function. Furthermore, the larger the model is, the better the parabolic approximation is. Thus, the following formula is used to describe the curve

$$s = (s_{\max} - s_{\min})\cdot\left[\frac{r_d - r_{\min}}{r_{\max} - r_{\min}}\right]^2 + s_{\min} \quad (2)$$

where s is the data size at resolution r_d, $r_{\min}$ the lowest resolution, $r_{\max}$ the highest resolution, $s_{\min}$ the data size at resolution $r_{\min}$, and $s_{\max}$ the data size at resolution $r_{\max}$. The approximating results are shown in Fig. 18.

Let the distances from the viewer to the near and far planes of the viewing frustum be $d_{\min}$ and $d_{\max}$, respectively, as shown in Fig. 11. If the distance between the viewer and the model is equal to $d_{\max}$, it is rendered at the lowest resolution. If the distance is equal to $d_{\min}$, it is rendered at the highest resolution. Given a constant bandwidth and a user's moving speed, this distance should be linearly related to data needed, i.e.,

$$\begin{aligned}\frac{d - d_{\min}}{d_{\max} - d_{\min}} &= \frac{s_{\max} - s}{s_{\max}}\\ &\Rightarrow s = s_{\max} - s_{\max}\cdot\frac{d - d_{\min}}{d_{\max} - d_{\min}}.\end{aligned} \quad (3)$$

By substituting (3) into (2), we have

$$r_d = (r_{\max} - r_{\min})\cdot\sqrt{\frac{s_{\max}\cdot\frac{d_{\max}-d}{d_{\max}-d_{\min}} - s_{\min}}{s_{\max} - s_{\min}}} + r_{\min}. \quad (4)$$

To summarize, by incorporating (1), the final resolution setting formula can be obtained by

$$r = r_d \cdot r_\theta. \quad (5)$$

V. Rate Control

The resolution setting in (5) implicitly assumes that there is no constraint on transmission bandwidth. But, the bandwidth is usually limited by networks in a time-varying way. We develop a rate control algorithm to adapt to different network situations. It is assumed that data are transmitted over a channel with variable bandwidth $X(t)$, and the server is informed of viewer's position every $\triangle t$ without delay.

Fig. 19 illustrates a scenario where a viewer moves around the object. At time t_1, the viewer is located at V_1 and able to discern the region denoted by arc $\widehat{A_1B_1}$. The server determines the resolution of visible partitions according to (5). Parts of the data

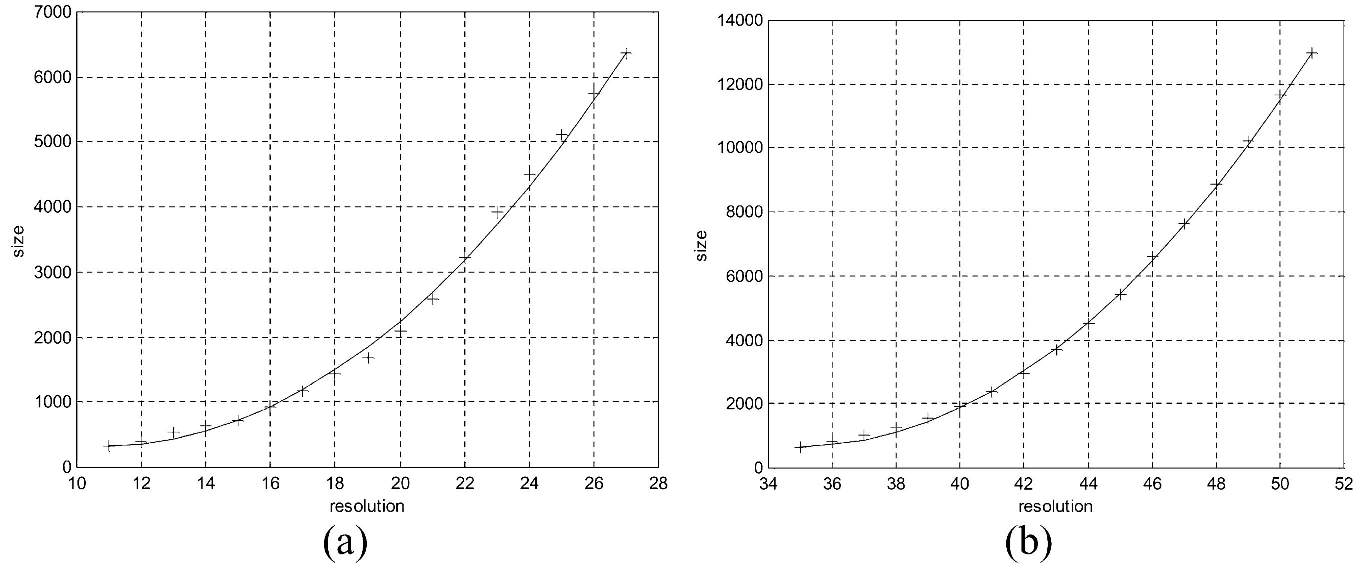


Fig. 18. Parabolic curve fitting for: (a) the Spock model and (b) the Bunny model.

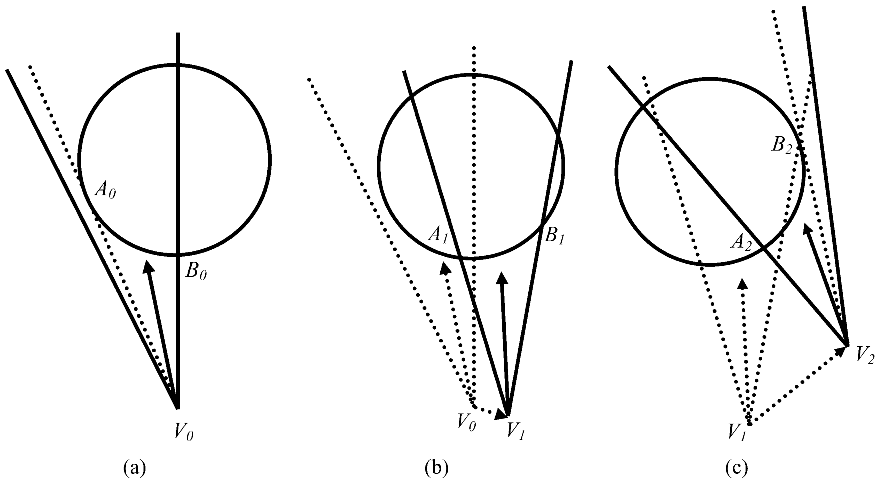


Fig. 19. Viewer moving around the model. (a) $t = t_1 - \triangle t$. (b) $t = t_1$. (c) $t = t_1 + \triangle t$.

in visible region may have already been transmitted previously. The incremental data amount is denoted by $D_R(t_1, t_1 + \triangle t)$. The maximum data amount that can be transmitted during interval $(t_1, t_1 + \triangle t)$ can be estimated via

$$D_N(t_1, t_1 + \triangle t) = X(t_1) \cdot \triangle t.$$

If $D_R(t_1, t_1 + \triangle t) > D_N(t_1, t_1 + \triangle t)$, the networks cannot support the requested resolutions. In such a case, the server iteratively searches the partition with the highest resolution within the visible area $\widehat{A_1 B_1}$ and decreases its resolution level by 1, until the incremental data amount satisfies the network constraint.

On the other hand, if $D_R(t_1, t_1 + \triangle t) \leq D_N(t_1, t_1 + \triangle t)$, the server repeatedly increases the partition with the lowest resolution within area $\widehat{A_1 B_1}$ by 1, until all extra bandwidth is consumed. Moreover, when the bandwidth is not fully consumed even if all visible partitions are transmitted at the highest resolution, the server predicts the viewer's new position in the next interval and finds out which partitions will probably come into view. There are two parameters associated with viewer's movement, position and viewing direction.

As shown in Fig. 19, locations of the viewer at time instances $t_1 - \triangle t$ and t_1 are denoted by V_0 and V_1, respectively. Then, the location at time $t_1 + \triangle t$ can be predicted as

$$V_2' = a_1 V_1 + a_0 V_0$$

where the weighting parameters a_1 and a_0 can be determined by the least square method based on previous N observations. The viewing direction can also be predicted using a similar approach.

VI. Experimental Results

The proposed system is tested in two scenarios: static viewing and dynamic viewing. In the static viewing case, the view point does not change so that the server only need to set the resolution once. In the dynamic viewing case, since the viewing parameters vary continuously, the server has to update the resolution adaptively.

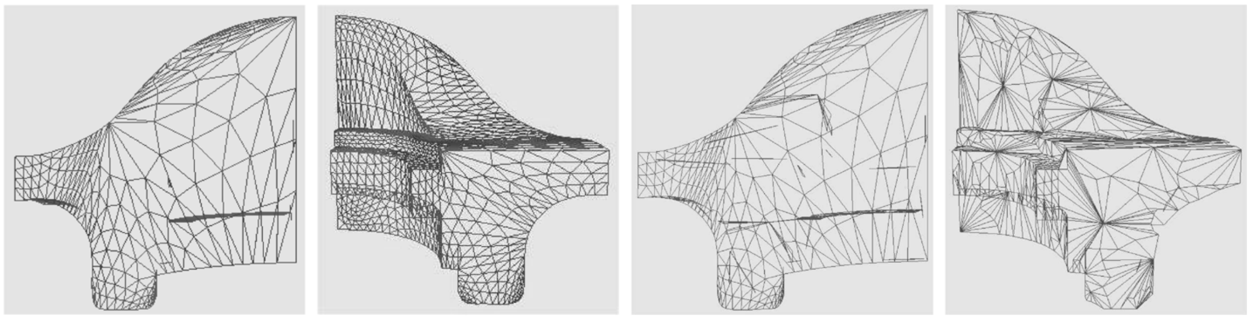


Fig. 20. Compression of the Fan Disk model. From left to right: CPM, front-facing image; CPM, back-facing image; the proposed algorithm, front-facing image; the proposed algorithm, back-facing image.

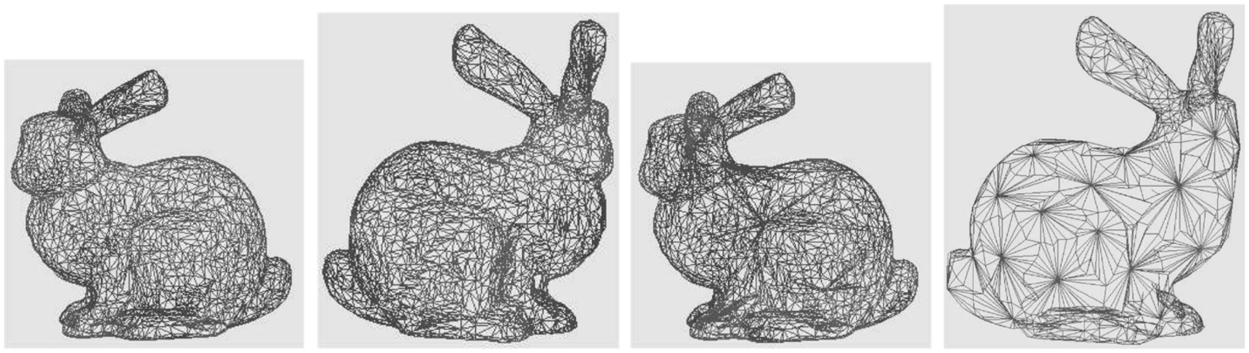


Fig. 21. Compression of the Bunny model. From left to right: CPM, front-facing image; CPM, back-facing image; the proposed algorithm, front-facing image; the proposed algorithm, back-facing image.

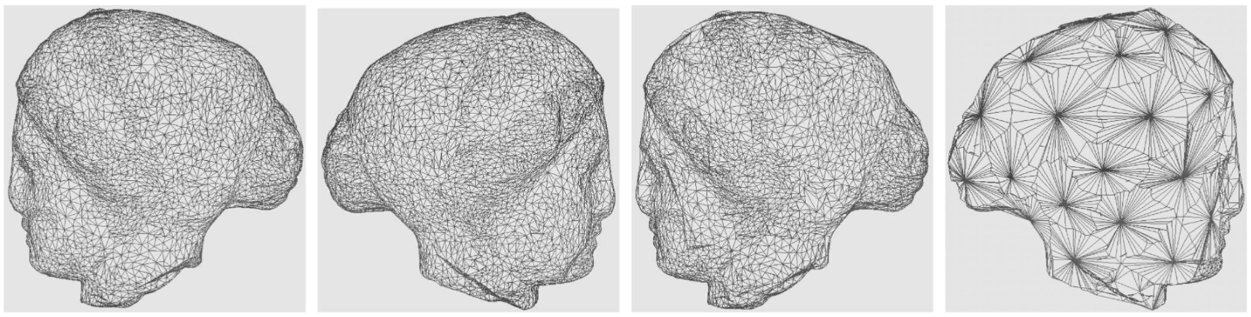


Fig. 22. Compression of the Venus Head model. From left to right: CPM, front-facing image; CPM, back-facing image; the proposed algorithm, front-facing image; the proposed algorithm, back-facing image.

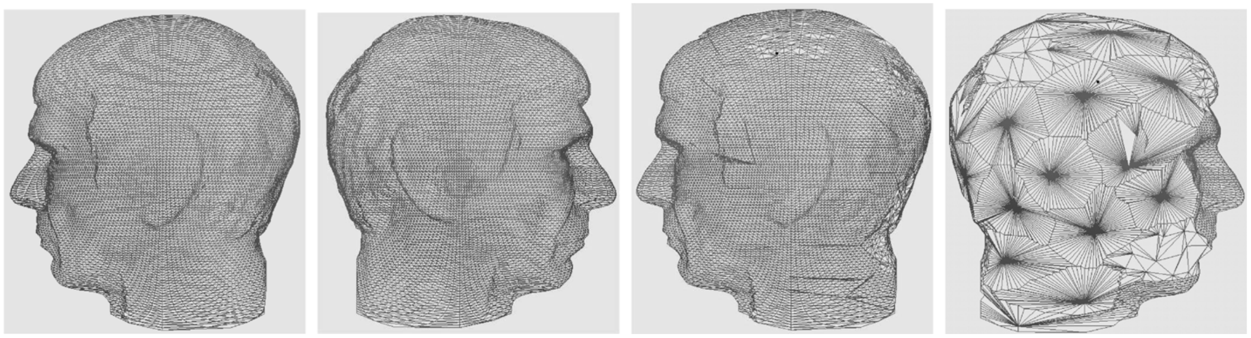


Fig. 23. Compression of the Spock model. From left to right: CPM, front-facing image; CPM, back-facing image; the proposed algorithm, front-facing image; the proposed algorithm, back-facing image.

A. Static Viewing

We compare the performance of VDPM on Fan Disk, Bunny, Venus Head, and Spock mesh models with that of CPM. Figs. 20–23 show reconstructed images. Note that the two algorithms use the same simplification operations and arithmetic coders. The only difference is the introduction of the partition

TABLE II
COMPRESSED FILE SIZES (IN BYTES). "TRANS. DATA" = SIZE OF THE TRANSMITTED DATA FOR FIGS. 20–23

model	#. of	#. of visible	CPM	proposed algorithm		comparison	
	vertices	vertices	total size (A)	total size (B)	trans. data (C)	B/A	C/A
Fan Disk	1698	945	6967	7626	4273	1.095	0.613
Bunny	4919	2745	19803	20835	11958	1.052	0.604
Venus Head	8268	4190	30808	32131	16402	1.043	0.532
Spock	16386	9202	50902	52583	30458	1.033	0.598

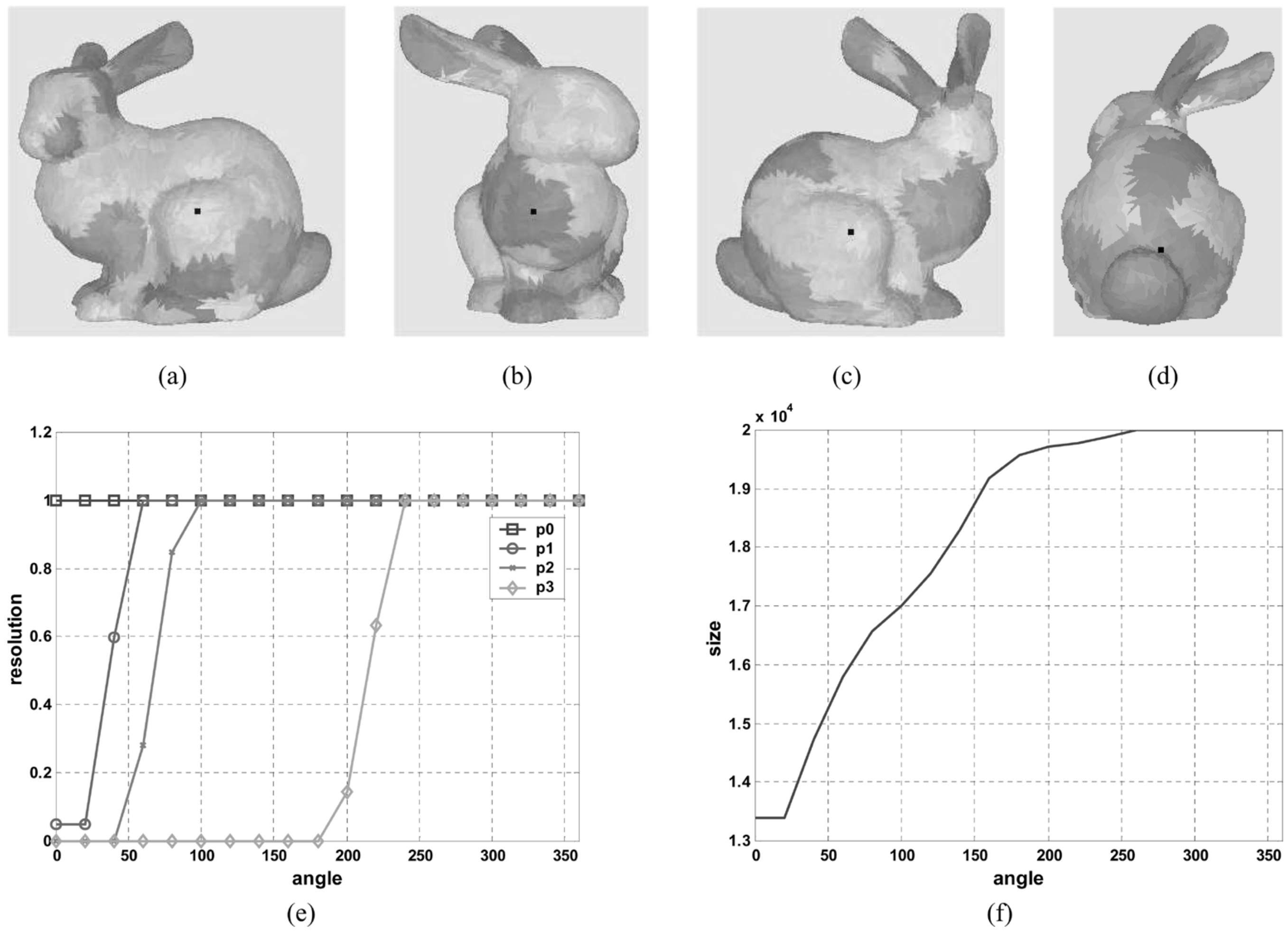


Fig. 24. Moving around the Bunny model: (a) 0°. (b) 90°. (c) 180°. (d) 270°. (e) Resolution curves of four partitions. (f) Transmitted data size.

concept in our algorithm. As a result, the simplification strategy is totally changed.

For geometry encoding, 12 bits are used to represent each coordinate of vertices for the original mesh. The full resolution is assigned to CPM. For VDPM, each partition is assigned the resolution adaptively, based on the viewing angle by (5). The proposed algorithm provides comparable quality for the front-facing images, while reconstructing invisible back-facing parts at the lowest resolution.

Table II summarizes coding results. VDPM requires about 3%–10% more bits to refine the models at the finest resolution. This is because VDPM independently encodes each partition to enable view-dependent transmission. Thus, the correlation among partitions cannot be as efficiently exploited as in the uniform progressive coder that has no restriction on independency. However, notice that VDPM requires about 40% less bits to yield a comparable visual quality for front-facing images.

An extra benefit of our method is the reduction of memory usage. Since each partition can be simplified and encoded independently, our method demands a much smaller memory size for encoding than CPM. The same argument applies to the decoding process. In the rendering process, by assuming that the required memory is proportional to the number of vertices to be

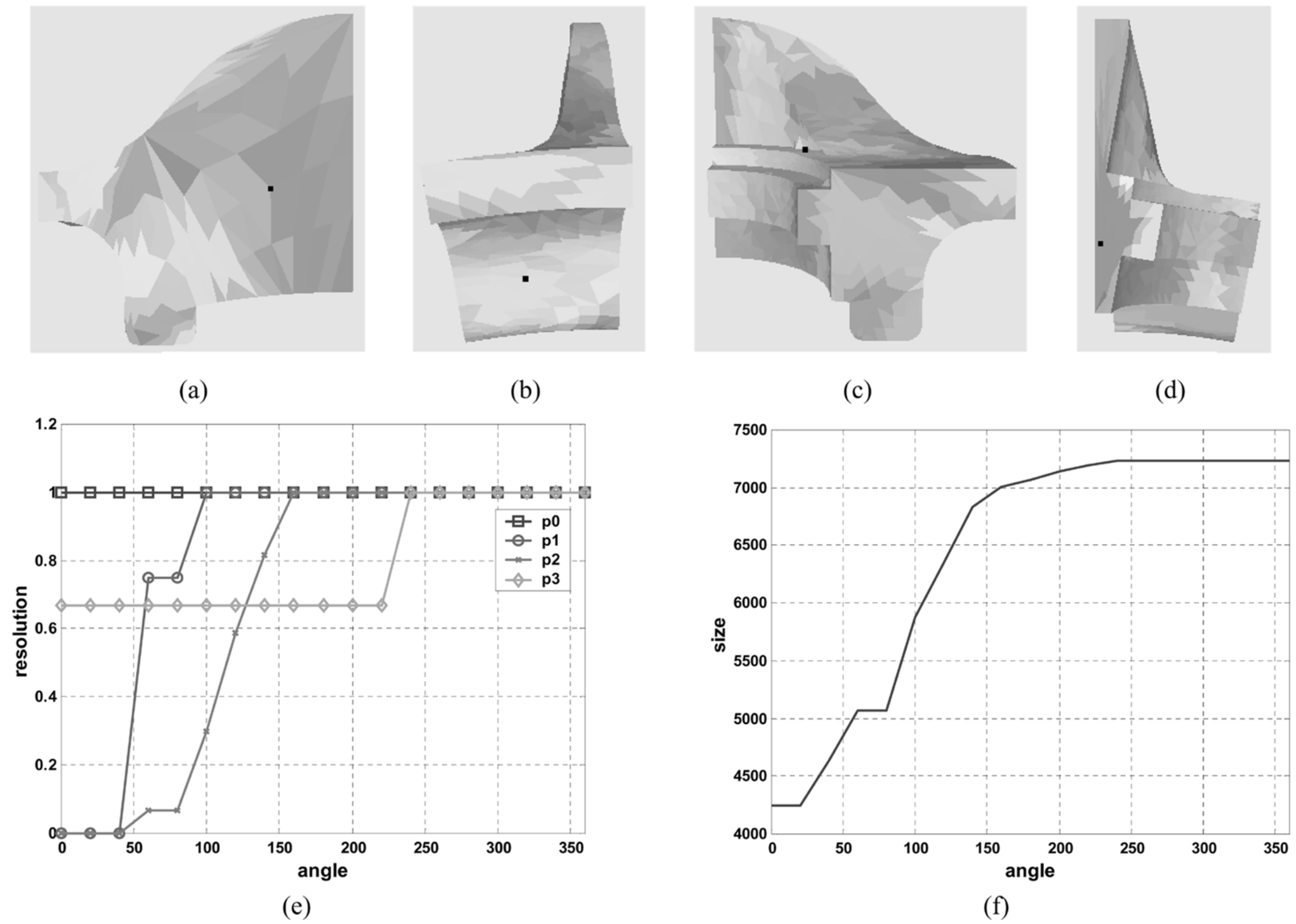


Fig. 25. Moving around the Fan Disk model: (a) 0°. (b) 90°. (c) 180°, (d) 270°. (e) Resolution curves of four partitions. (f) Transmitted data size.

rendered, we see from Table II that our method only uses around one half memory of CPM.

B. Dynamic Viewing

Viewer's movement can be categorized into three modes: 1) moving straight toward the object; 2) moving around the object in a circle; and 3) moving randomly. The third case can be viewed as a superposition of the first two movements. Due to space limitation, we only present the simulation results for the second case.

First, we demonstrate that the partitions are transmitted in order of relevance to the user's viewing parameters. In this test, we assume that a viewer moves around the object at a constant velocity, and the refinement data obtained by (5) are transmitted during the initial waiting time along with the base model. Fig. 24(a)–(d) shows rendered images of the Bunny model at angles 0°, 90°, 180°, and 270°, respectively. Fig. 24(e) shows the resolutions of four partitions $\mathcal{P}_0 - \mathcal{P}_3$ in terms of angles. The root vertices of $\mathcal{P}_0 - \mathcal{P}_3$ are drawn as black dots in Fig. 24(a)–(d), respectively. At angle 0°, $\mathcal{P}_0$ is located approximately at the center of the front-facing parts. Thus, it is transmitted at the full resolution. On the contrary, at angle 0°, $\mathcal{P}_2$ is invisible, and its data are not transmitted at all. It is observed that $\mathcal{P}_2$ starts to be visible and its resolution starts to increase, when the user moves about 90°. Fig. 24(f) shows the number of bytes to be transmitted in this case. At first, we require about 10 000 bytes to reconstruct visible parts. Then, as the user is moving, we need to transmit only the incremental data that are necessary to reconstruct newly visible parts. If the incremental data rate is smaller than the channel bandwidth, the viewer can interactively browse the object in real time after the initial waiting time. Fig. 25(a)–(f) shows similar results for the Fan Disk model. These results demonstrate that the proposed algorithm can effectively reduce the required bandwidth by transmitting only visible parts.

Next, we compare the performance of VDPM and CPM under different bandwidth constraint on the Spock model. It is assumed that a viewer moves around the Spock model at a constant velocity, namely, 10°/s, and the base model, together with the visible parts from the viewer's first view point, have already been transmitted during the initial waiting time. The same amount of data are transmitted for CPM to fairly compare the performance of the two algorithms in the streaming scenario.

Transmission of refined details is conducted under three conditions: at a constant bit rate (CBR) of 2.0, 4.0, and 6.0 kb/s. Fig. 26 show rendered images when the model is transmitted at

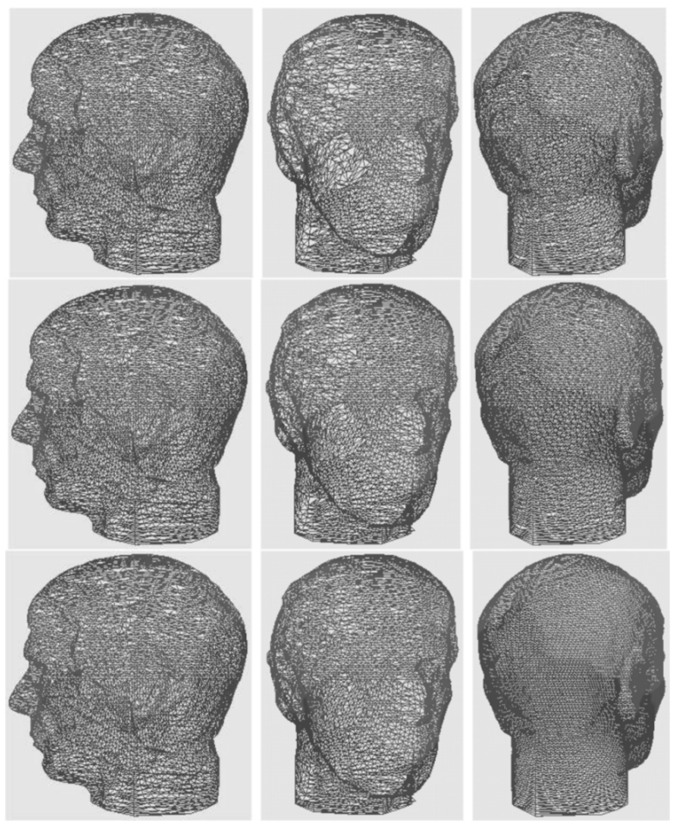


Fig. 26. Moving around the Spock model with VDPM, where the first row is transmitted at a rate of 2.0 kb/s, the second row at 4.0 kb/s and the third row at 6.0 kb/s.

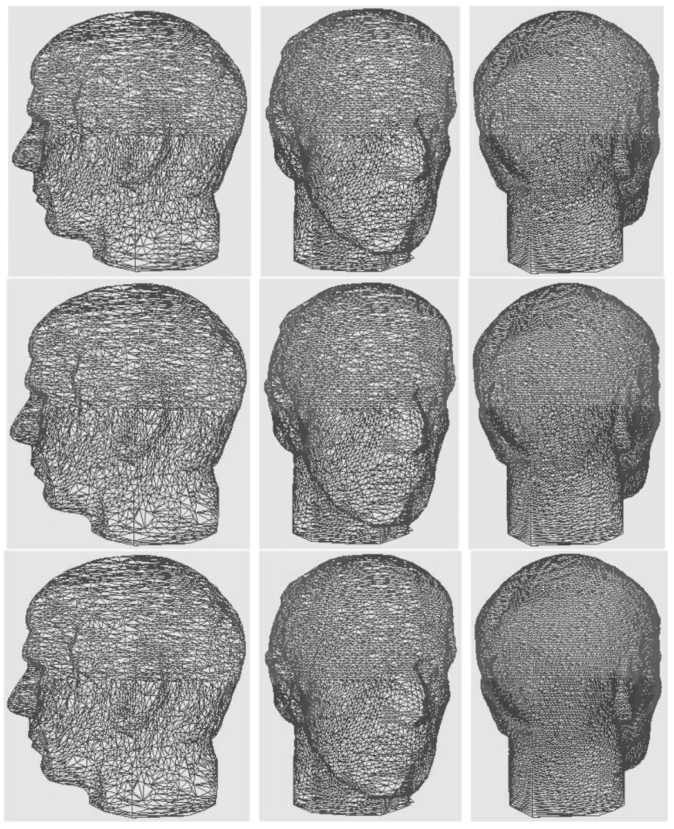


Fig. 27. Moving around the Spock model with CPM, where the first row is transmitted at a rate of 2.0 kb/s, the second row at 4.0 kb/s and the third row at 6.0 kb/s.

these three rates. The three images on each row correspond to the model rotated at angles 0°, 120°, and 240°, respectively. Fig. 27 show images of CPM for comparison. VDPM always provides better visual quality than CPM in the beginning, since it utilizes the network resource on visible parts only. However, when the bandwidth is not enough or the user moves too fast, it is observed that there is significant visual quality degradation in newly visible parts, as shown in the 2.0 kb/s case. The degradation in CPM is not as severe since it refines models uniformly. This drawback can be overcome by buffering a larger area at a lower resolution. The network-dependent strategy deserves further study.

To control and measure the overall effect on the visual quality after cutting out invisible parts, errors need be calculated and compared with uniform mesh compression algorithms. We use the following two measures.

- The global error. This error is used to compare the global difference of models, no matter whether the vertex is visible or not.
- The visible error. This error is used to compare the difference only in visible areas.

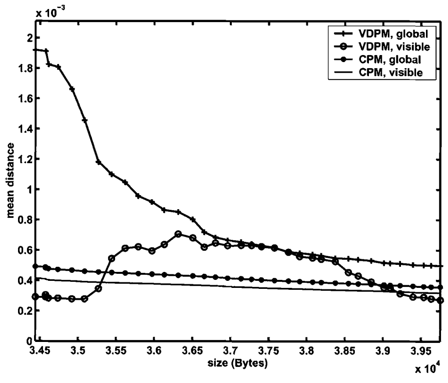
Both of them are obtained by computing the mean distance between the simplified mesh and the original one [2]. The error curves are shown in Fig. 28, where the x-axis represents the data transmitted and the y-axis is the error measure.

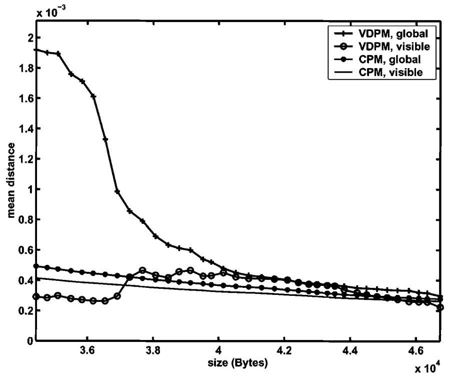
Although the global error of our algorithm is always the largest, our algorithm can provide superior visual quality in visible areas at the beginning with respect to CPM. After the viewer moves around 150°, CPM has better quality since the back-facing parts come into view gradually and it can take advantage of previously transmitted data, as analyzed above. This demonstrates that our algorithm is more efficient than non view-dependent method when the user does not need to view the whole model within a short period of time.

VII. Conclusion and Future Work

A view-dependent progressive mesh coding and streaming algorithm was proposed in this research. To enable view-dependent progressive transmission, the proposed algorithm divided a mesh model into several partitions, and encoded each partition independently. It was shown by simulation results that the proposed algorithm can reduce the required bandwidth by transmitting only visible parts while cutting out invisible parts and have better performance than non-view-dependent methods when the user does not need to view the whole model within a short period of time.

There are several interesting topics worthy of further investigation. First, it is important to develop an interactive protocol, which enables real time graphics streaming and enhances interaction between server and client. Second, we would like to consider error-resilient issues, so we can find ways to combat channel noise arising in wireless channels.


(a)


(b)

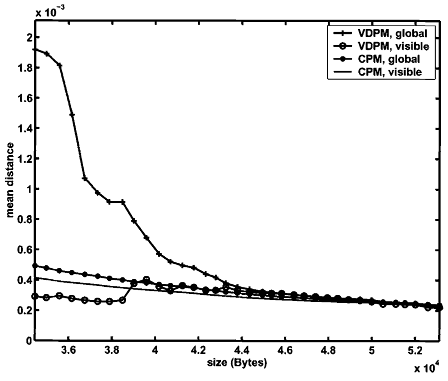

(c)

Fig. 28. Distortion curves at a transmission rate of: (a) 2.0 kb/s, (b) 4.0 kb/s, and (c) 6.0 kb/s.

Third, the network-dependent strategy needs further study. For example, how to adjust the resolution setting method according to different network bandwidth constraint.

References

[1] P. Alliez and M. Desbrun, "Progressive compression for lossless transmission of triangle meshes," in *Proc. 28th Annu. Conf. Computer Graphics Interactive Techniques*, 2001, pp. 195–202.

[2] P. Cignoni, C. Rocchini, and R. Scopigno, *Metro: Measuring Error on Simplified Surfaces*. Malden, MA: Blackwell, 1998, vol. 17, pp. 167–174.

[3] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwat, F. Brooks, and W. Wright, "Simplification envelope," in *Proc. Computer Graphics, Annu. Conf. Series, ACM SIGGRAPH*, Aug. 1995, pp. 119–128.

[4] D. Cohen-Or, D. Levin, and O. Remez, "Progressive compression of arbitrary triangular meshes," in *Proc. Visualization*, 1999, pp. 67–72.

[5] M. Deering, "Geometry compression," in *Proc. Comp. Graphics, Annu. Conf. Series, ACM SIGGRAPH*, Aug. 1995, pp. 13–20.

[6] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques*, 1997, pp. 209–216.

[7] S. Gumhold and W. Straßer, "Real time compression of triangle mesh connectivity," in *Proc. Computer Graphics, Annu. Conf. Series, ACM SIGGRAPH*, Aug. 1998, pp. 133–140.

[8] H. Hoppe, "Progressive meshes," in *Proc. SIGGRAPH-96*, Aug. 1996, pp. 99–108.

[9] ——, "View-dependent refinement of progressive meshes," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques*, 1997, pp. 189–198.

[10] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in *Proc. SIGGRAPH-2000*, July 2000, pp. 271–278.

[11] J. Li and C.-C. J. Kuo, "Progressive coding of 3-D graphics models," in *Proc. Multimedia Computing and Systems*, 1997, pp. 135–142.

[12] ——, "A dual graph approach to 3-D triangular mesh compression," in *Proc. IEEE Int. Conf. Image Processing*, Chicago, IL, 1998, pp. 891–894.

[13] D. Luebke and C. Erikson, "View-dependent simplification of arbitrary polygonal environments," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques*, 1997, pp. 199–208.

[14] R. Pajarola and J. Rossignac, "Compressed progressive meshes," *IEEE Trans. Visual. Comput. Graphics*, vol. 6, pp. 79–93, Feb. 2000.

[15] C. Prince, "Progressive meshes for large models of arbitrary topology," M. S. thesis, Computer Sci. and Eng. Dept., Univ. Washington, Seattle, 2000.

[16] R. Southern, S. Perkins, B. Steyn, A. Muller, and P. M. Blake, "A stateless client for progressive view-dependent transmission," in *Proc. Web3-D Symp., ACM*, 2001, pp. 43–50.

[17] K. Sayood, *Introduction to Data Compression*. San Mateo, CA: Morgan Kaufmann, 1996.

[18] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," IBM Watson Research Center, Tech. Rep. RC-20 340, 1996.

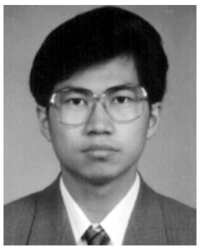
[19] C. Touma and C. Gotsman, "Triangle mesh compression," *Proc. Graphics Interface*, pp. 26–34, 1998.

[20] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1992.

[21] J. C. Xia and A. Varshney, "Dynamic view-dependent simplification for polygonal models," in *Proc. Visualization*, 1996, pp. 327–334.

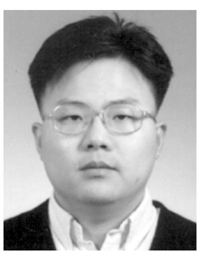
[22] Z. Yan, S. Kumar, and C.-C. J. Kuo, "Error-resilient coding of 3-D graphics models via adaptive mesh segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 860–873, July 2001.

[23] S. Yang, C.-S. Kim, and C.-C. J. Kuo, "View-dependent progressive mesh coding for graphics streaming," in *Proc. SPIE ITCOM*, 2001, pp. 154–165.



Sheng Yang (S'04) received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California (USC), Los Angeles, in 2004.

Since June 2004, he has worked as a Visiting Scholar with the Signal and Image Processing Institute, USC. His research interests include 3-D graphic compression and transmission.



Chang-Su Kim (S'95–M'01) was born in Seoul, Korea, in 1971. He received the B.S. and M.S. degrees in control and instrumentation engineering in 1994 and 1996, respectively, and the Ph.D. degree in electrical engineering in 2000, all from Seoul National University (SNU).

From 2000 to 2001, he was a Visiting Scholar with the Signal and Image Processing Institute, University of Southern California, Los Angeles, and a Consultant for InterVideo Inc., Los Angeles. From 2001 to 2003, he was a Postdoctoral Researcher with the School of Electrical Engineering, SNU. In August 2003, he joined the Department of Information Engineering, the Chinese University of Hong Kong, Hong Kong, as an Assistant Professor. His research topics include video and 3-D graphics processing and multimedia communications.

C.-C. Jay Kuo (S'83–M'86–SM'92–F'99) received the B.S. degree from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1980 and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1985 and 1987, respectively, all in electrical engineering.

He was Computational and Applied Mathematics (CAM) Research Assistant Professor in the Department of Mathematics at the University of California, Los Angeles, from October 1987 to December 1988. Since January 1989, he has been with the Department of Electrical Engineering-Systems and the Signal and Image Processing Institute at the University of Southern California, where he currently has a joint appointment as Professor of Electrical Engineering and Mathematics. His research interests are in the areas of digital signal and image processing, audio and video coding, multimedia communication technologies and delivery protocols, and embedded system design. He has guided about 50 students to their Ph.D. degrees and supervised ten postdoctoral research fellows. He is a coauthor of six books and more than 600 technical publications in international conferences and journals.

Dr. Kuo is a Fellow of SPIE and a member of ACM. He is Editor-in-Chief for the *Journal of Visual Communication and Image Representation*, Associate Editor for IEEE TRANSACTIONS on SPEECH AND AUDIO PROCESSING and Editor for the *Journal of Information Science and Engineering* and the EURASIP *Journal of Applied Signal Processing*. He is also on the Editorial Board of the IEEE Signal Processing Magazine. He served as Associate Editor for IEEE TRANSACTIONS ON IMAGE PROCESSING in 1995–1998 and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 1995–1997. He received the National Science Foundation Young Investigator Award (NYI) and Presidential Faculty Fellow (PFF) Award in 1992 and 1993, respectively.