Progressive Coding of 3-D Graphic Models

JIANKUN LI AND C.-C. JAY KUO, SENIOR MEMBER, IEEE

Based on state-of-the-art graphic-simplification techniques and progressive image-coding schemes, we propose a new hierarchical three-dimensional graphic-compression scheme in this research. This scheme progressively compresses an arbitrary polygonal mesh into a single bitstream. Along the encoding process, every output bit contributes to the reduction of coding distortion, and the contribution of bits decreases according to their order of position in the bitstream. At the receiver end, the decoder can stop at any point while giving a reconstruction of the original model with the best rate-distortion tradeoff. A series of models of continuous varying resolution can thus be constructed from the single bitstream. This property, which is referred to as the embedding property since the coding of a coarser model is embedded in the coding of a finer model, can be widely used in robust error control, progressive transmission and display, level-of-detail control, etc. It is demonstrated by experiments that an acceptable quality level can be achieved at a compression ratio of 20 to 1 for several test graphic models.

Keywords*—*Embedded codecs, graphic coding, graphic simplification, progressive coding.

I. INTRODUCTION

Since the boom of the three-dimensional (3-D) laser scanning system and the virtual-reality modeling language for graphic description, 3-D graphic models have become more accessible to general end users. In most existing commercial hardware and software, the polygonal mesh is one of the popular tools for object description. A complex object with many fine details can be faithfully represented with thousands or even millions of polygons. Even high-end computers have difficulty in manipulating and visualizing such kinds of models, since both the memory requirement and the rendering speed are directly proportional to the total number of polygons in a model [9]. Similarly, transmission of 3-D graphic models over the network is a well-known bottleneck for collaborative design due to the large amount of data. Future applications of 3-D graphic models could be potentially limited due to the lack of efficient representation. Another important consideration in the generation of 3-D graphic models is to allow a multiresolution representation of the object. That is, the information is coded in the order of importance. The most important information is encoded first, while finer details are gradually added at a later stage. As a result, the decoder can construct a model of different resolutions, from the coarsest approximation to the finest replica, depending on the application requirement. This property, which is often referred to as the embedding property, finds wide applications in robust error control, progressive transmission/display, level-of-detail control, etc.

Generally speaking, there are two types of data in a 3-D graphic model, i.e., structure data and attribute data. Structure data specify the connectivity information among vertices and characterize the topology of the model, while attribute data describe information of each individual vertex such as the position, color, surface normal, and other application-specific information. Most 3-D graphic files specify the structure data by a list of polygons, each of which is in turn specified by a list of vertex indexes. To recover the model properly, structure data have to be coded losslessly. Consequently, it is difficult to achieve a high compression ratio in representing these data. This is the major obstacle for 3-D graphic coding, especially at low bit rates. In contrast, attribute data bear a very strong local correlation. They can be effectively compressed with lossy compression methods while keeping the error under a certain tolerable level.

In comparison with research in audio, image, and video coding, the amount of work in 3-D graphic coding is relatively small. This might be explained by the fact that the 3-D graphic model has been gaining popularity only recently. Another reason is that audio, image, and video data come as one-dimensional (1-D), two-dimensional (2-D), and 3-D regular array, respectively. Such structures require no coding by themselves. However, 3-D graphic data are not defined on a regular grid. Both the geometric structure and data values have to be coded. This requirement imposes a very challenging task beyond existing audio-, image-, and video-coding techniques.

Three-dimensional graphic model simplification is a closely related topic in which multiple simplification

Manuscript received July 21, 1997; revised December 11, 1997. The Guest Editor coordinating the review of this paper and approving it for publication was T. Chen. This work was supported in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, in part by the Annenberg Center for Communication, University of Southern California, and in part by the California Trade and Commerce Agency.

The authors are with the Integrated Media Systems Center and the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-2564 USA (e-mail: jiankunl@sipi.usc.edu; cckuo@sipi.usc.edu).

Publisher Item Identifier S 0018-9219(98)03518-X.

steps are applied to an object to obtain a sequence of approximations with a lower complexity. Some simplification methods preserve the topology of the polygonal mesh to maintain a faithful description. Others ignore topological constraints in order to achieve a high data deduction. In both cases, the sequence of simplification steps is carefully chosen according to a certain rule to maximize the fidelity of approximations. Successive levels of simplification details are stored, and the most appropriate level (sometimes all levels) are used for a given application [6].

Previous work on 3-D graphic simplification is reviewed below. Schröeder [19] proposed a decimation algorithm that significantly reduces the number of polygons required to represent an object by repeatedly removing nonessential vertices. Deering [3] introduced the concept of a generalized triangle strip, which allows a compact representation of a planar graph by using a linear data structure and a set of stack operators. Sweldens [22] constructed a lifting scheme to compactly represent scalar functions defined on a sphere. Lounsbery [13] and Eck [4] established a wavelet transform defined on an arbitrary 3-D domain to compress a graphic model at several detail levels. Taubin and Rossignac [23] represented a triangulated mesh by using two interlocked trees. Their method compressed the connectivity information into, roughly, an average of two bits per triangle and used multistage vector quantization to code the vertex position. Hoppe [10], [15] constructed a progressive mesh by recursively applying the edge-collapse operation to an arbitrary mesh and encoded the vertex position via Huffman coding.

In all previous work, the coding procedures of structure data and attribute data are kept separately, as are the coding results. Furthermore, attribute data are usually coded by the traditional run-length coding with a single resolution level. Therefore, the embedding property is not fully supported. In this work, we propose a new 3-D graphic coding scheme that progressively compresses an arbitrary polygonal mesh into one single bitstream. Along the encoding process, we carefully allocate bits to the coding of structure or attribute data. Notice that both codings contribute to the quality of the decompressed model, and the quality is determined by the coarser information from both parts. Repeatedly coding one part without checking the other will result in a model with either too many vertices of insufficient precision or too few vertices of overly accurate precision. A quality-control rule is devised in our scheme to determine switching between these two coding tasks to guarantee the quality balance. Every output bit contributes to the distortion reduction, and the contribution depends on its location in the bitstream. Bits at later locations are less important. The decoder can stop at any point while giving a reconstruction of the original model with the optimal rate-distortion tradeoff. A series of models of continuous varying resolutions can thus be constructed from the single bitstream.

This paper is organized as follows. A brief review of graphic simplification and embedded coding techniques is given in Section II, where the concept of successive quantization and entropy coding is also explained. The new, progressive 3-D graphic coder is described in detail in Section III. It contains three basic steps. For a given arbitrary polygonal mesh, a hierarchical representation consisting of structure and attribute data is first built. Then, both structure and attribute data are encoded separately in a progressive manner. Last, the coding bitstreams of structure and attribute data are multiplexed into a single bitstream according to a rate-distortion criterion. In Section IV, experimental results are provided to demonstrate the performance of the proposed graphic-coding scheme. Concluding remarks and future extensions are given in Section V.

II. Background

A. Graphic Simplification

The polygon is a primitive of an object represented by a polygonal mesh. The cost of storing or transmitting a polygonal mesh of n primitives is $O(n)$. The cost of rendering such a model is also $O(n)$ [1], [8]. The primary goal of graphic simplification is to reduce the number of primitives required to represent a physical or an abstract object faithfully. Several different algorithms have been proposed to serve this purpose. They can be roughly classified into three categories, i.e., surface retiling, vertex decimation, and vertex clustering, as detailed below.

With surface retiling, one triangulates polygonal surfaces with a new set of vertices to replace the original. The new set usually consists of fewer vertices than the original one while preserving its topology. Turk [25] suggested randomly adding points on the polygonal surface and then redistributing these points by exerting a repelling force while removing old vertices gradually. Hoppe and DeRose [11] defined an energy function that was minimized to determine positions of new vertices. Hinker and Hansen [9] merged coplanar and near coplanar polygons into larger complex polygons and retriangulated them into fewer simple polygons. Kalvin and Taylor [12] developed a similar algorithm, which allows additional control of approximation error. Cohen *et al.* [2] and Varshney *et al.* [26] surrounded the original polygonal surface with two envelopes and then generated a simplified surface within the volume enclosed by these two envelopes.

By using vertex decimation, Schröeder [19] proposed to make multiple passes over an existing polygonal mesh and use local geometry and topology information to remove vertices that meet a distance or an angle criterion. The hole left by the vertex-removal process is patched by a local triangulating process. Soucy [21] described a more sophisticated algorithm following the same idea.

The idea behind vertex clustering is that a detailed part in a model is represented by a set of spatially close vertices. A clustering algorithm is used to generate a hierarchy of clusters and approximations of different resolutions [18]. Rossignac and Borrel [16] divided the bounding box of

the original model into a grid. Within each cell, vertices are clustered together into a single vertex, and the model surface is updated accordingly. He *et al.* [7] adopted a signal-processing approach to sample the input object and used low-pass filtering to remove high frequencies of the object. Edge collapse is the most commonly used clustering technique. It removes an edge and contracts two end points into a new vertex. To determine the position of the new vertex, André [6] required the volume to be preserved after contraction. Hoppe [10], [15] used edge collapse to form the progressive representation of triangulated geometric models. Garland [5] developed a method called pair contraction, which is capable of contracting an arbitrary pair of vertices.

All of the above methods can generate multiple level-of-detail representations of the original model. Surface retiling achieves this by choosing several new sets consisting of a different number of vertices. However, it cannot guarantee that the coarse representation is a subset of the finer representation. It has been shown that, with a clever use of vertex decimation or clustering techniques, one can generate a progressive representation by building a hierarchical structure along the simplification process. Furthermore, in some cases, the progressive representation is designed in such a way that the original models can be represented by a sequence of approximations of continuous varying resolutions.

B. Embedded Coding

A typical image-coding scheme consists of three steps: transform, quantization, and entropy coding. For example, in the image-compression standard of the Joint Photographic Experts Group (JPEG), the discrete cosine transform (DCT) is first applied to an image block of 8×8 pixels. The resulting DCT coefficients are then quantized with a quantization table and mapped into an index set. Last, the index set is encoded by an entropy-performed coder. Decoding is simply in reverse order. The entire process is applied in a coefficient-by-coefficient approach. That is, a coefficient is completely coded before the coding of the next coefficient. In the embedded coding scheme, however, each coefficient is successively quantized into a certain number of bits. Their most significant bits are grouped together to form one bit layer and encoded first. Then the layer of the second most significant bits are encoded, and so on. By adding more layers of bits, the approximation becomes more precise. Such a coding order is consistent with the importance of each bit so that the coder can provide the best possible reconstruction of the original image within a given bit budget.

Embedded coding has been intensively studied in the context of image compression for the last several years. Some examples of embedded coders are given below. The JPEG still-image compression standard [14] defines a progressive compression mode that enables progressive transmission and display of compressed images. Shapiro [20] proposed the embedded zero-tree wavelet algorithm, which encodes the bit layer effectively by considering a particular data structure called the zero tree. Following the basic framework of Shapiro *et al.* [24] developed the layered zero coder, which adopts an efficient arithmetic coder to gain more coding efficiency. Said and Pearlman [17] presented another modification based on the concept of set partitioning in hierarchical trees.

Embedded coding can be applied in either the spatial domain or the transform domain. In the proposed algorithm detailed in Section III, we will perform embedded coding in the spatial domain. The two basic building blocks of the embedded coder are successive quantization and entropy coding. Details of these two steps are given below.

1) Successive Quantization: Given a sequence of input coefficients V_j, $j = 0, \cdots, n$, the first step of successive quantization is to scale all coefficients to the interval $[-1, 1]$. This can be done by simply dividing every coefficient V_j by the maximum magnitude of all coefficients.

Hereafter, we assume that every V_j is within the interval $[-1, 1]$ and can therefore be represented by $0.a_{j,0}a_{j,1}a_{j,2}\ldots$ for its magnitude plus bit s_j for its sign. We define function $sid(j) = k$, where $a_{j,k}$ is the leading nonzero bit of V_j. We call $a_{j,0}, a_{j,1}, \cdots, a_{j,k}$ the identification bits of V_j, and all bits that follows the refinement bits of V_j.

In the ith stage of the successive quantization, we use $a_{j,0}, a_{j,1}, \cdots, a_{j,i}$ to approximate the value of coefficient V_j. However, instead of using $0.a_{j,0}a_{j,1}\cdots a_{j,i}$ directly, we use $0.a_{j,0}a_{j,1}\cdots a_{j,i}1$ as the approximation, i.e., the average value rather than the lower bound of the quantization bin is adopted. Fig. 1 shows results for the first three successive quantization steps for nonnegative coefficients. For each quantization bin in the figure, the corresponding bit sequence is to its right, while a vertical bar shows the approximation value. The only exception to this approximation rule is the lowest quantization bin, where the coefficient is approximated by zero, as shown by the shadow in the figure. Note that a typical set of transformation coefficients consists of a large portion of coefficients with small magnitude. Their average is closer to zero.

In actual implementation, there is no need to scale the coefficients. Instead, we choose a set of quantization steps to generate $a_{j,i}$. The initial quantization step T_0 is one-half the maximum magnitude of all coefficients. Then, the quantization step size at stages $i+1$ and i are related via $T_{i+1} = 0.5 \times T_i$. We call a coefficient significant if its magnitude is greater than the current quantization step. Otherwise, it is insignificant. At the beginning, all coefficients are insignificant. For each coefficient V_j at quantization stage i, we have the following.

1) If currently insignificant, we apply

$$\begin{array}{llll} V_j > T_i, & a_{j,i} = 1, & s_j = 1, & V_j = V_j - 1.5 \cdot T_i \\ V_j < -T_i, & a_{j,i} = 1, & s_j = 0, & V_j = V_j + 1.5 \cdot T_i \\ \text{Otherwise}, & a_{j,i} = 0. & & \end{array}$$

In the first and second cases shown above, the coefficient becomes significant at this quantization step and $sid(j) = i$.

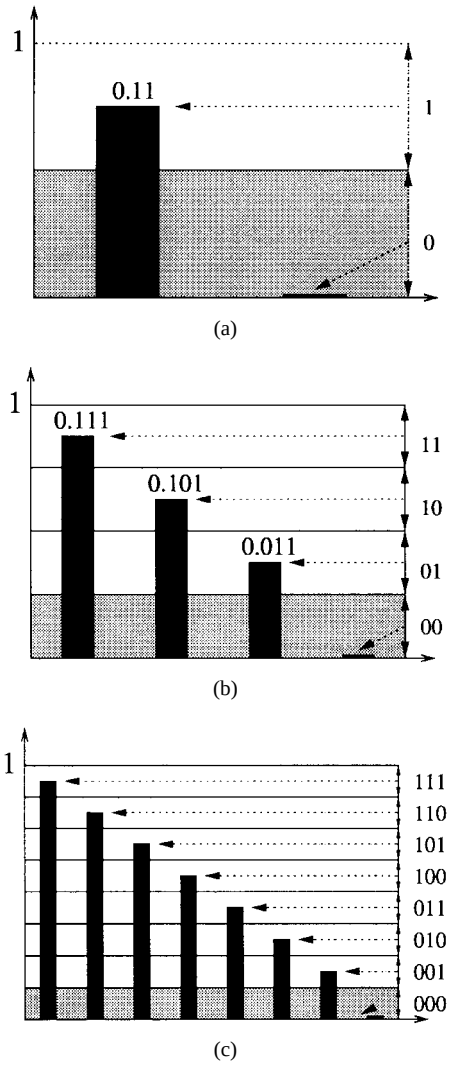


Fig. 1. Illustration of successive quantization. (a) After stage 0. (b) After stage 1. (c) After stage 2.

2) If currently significant, we apply the following refinement rules:

$$V_j \geq 0, \quad a_{j,i} = 1, \quad V_j = V_j - T_i$$
$$V_j < 0, \quad a_{j,i} = 0, \quad V_j = V_j + T_i.$$

2) Context Arithmetic Coding: For each identification bit $a_{j,i}$, $j = 1, \cdots, n$, $i = 0, 1, \cdots, sid(j)$, we use a context arithmetic coder to encode it. As mentioned above, a large portion of coefficients are with a very small magnitude, so that only n identification bits, i.e., $a_{j,\mathrm{sid}(j)}$, $j = 0, \cdots, n$, have a value "1," while the rest are all "0." They can be compressed efficiently if we can find a good probability distribution model for these bits. The context arithmetic coding provides a good tool to accomplish such a task. A context arithmetic coder is a set of independent arithmetic coders. Each coder is indexed by a number called the context. In the process of coding, a binary symbol along with a context is fed to the encoder. For each arithmetic coder, it encodes the binary symbol using its own built-in probability estimation. It counts the numbers of 0's and 1's under the same context and uses those occurrences to estimate the probability for the next incoming symbol. Each arithmetic coder dynamically estimates its own probability model as it encodes. We refer to [14, ch. 12–14] for more detailed discussion. The challenge that remains is the assignment of appropriate context so that each individual arithmetic coder can build a proper probability estimation for incoming symbols. This is application dependent. We present our method of context assignment in Section III-C.

For the refinement bit $a_{j,i}$, $j = 1, \cdots, n$, $i > sid(j)$, we simply take its value without further compression. Since it can be 0 or 1 with an equal probability, it cannot be compressed easily.

III. Progressive Graphic-Coding Algorithm

In this section, we describe the progressive 3-D graphic-coding algorithm in detail. First, a procedure to construct a hierarchical mesh is presented in Section III-A. Then, we examine the coding of the structure and the attribute data in Sections III-B and III-C, respectively. Last, in Section III-D, the coded bitstreams of the two types of data are multiplexed into a single bitstream by using a rate-distortion model.

A. Construction of Hierarchical Mesh

Even though the developed compression scheme can be applied to an arbitrary polygonal mesh in principle, we will focus on a mesh composed by triangles for simplicity. There are many graphic-simplification techniques, as reviewed in Section II. We adopt the vertex decimation method described in [19]. There are two reasons for this choice. First, it provides a nice hierarchical structure in terms of neighborhood areas for the progressive compression of structure data, as described in the remainder of this section. Second, the average of the attribute data of neighboring vertices provides a good prediction of the central vertex. Thus, the compression of attribute data can be done more effectively.

Multiple passes are made over all vertices in the mesh. Each vertex and its local topology are checked for the eligibility of removal. According to its local topology, the vertex is classified into three different types.

Simple	A simple vertex is surrounded by a complete cycle of triangles, and each edge connected to the vertex is shared by two triangles [see Fig. 2(a)].
Complex	If one of its connecting edges is not shared by two triangles, or is not forming a complete cycle of triangles, then the vertex is complex [see Fig. 2(b)].
Boundary	A boundary vertex is on the boundary of a mesh, i.e., within a semicycle of triangles [see Fig. 2(c)].

Each vertex may be assigned to one of three possible categories. To preserve the topology, complex vertices are not deleted from the mesh. Only simple and boundary

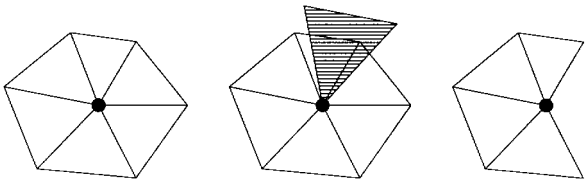


Fig. 2. Classification of vertices. (a) Simple. (b) Complex. (c) Boundary.

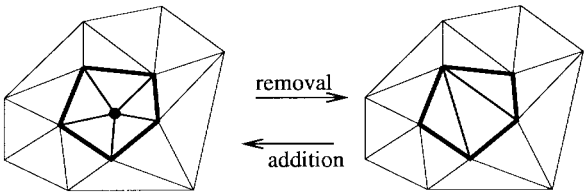


Fig. 3. Removal and addition of a vertex.

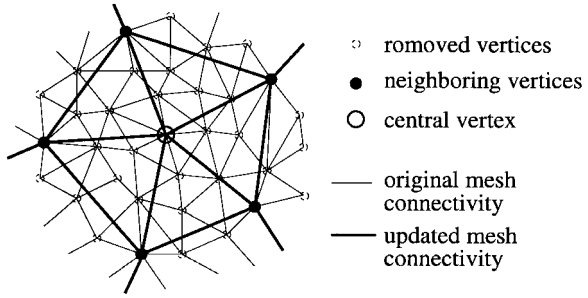


Fig. 4. Illustration of graphic simplification in the neighborhood of a vertex.

vertices are candidates for removal. The removal of a vertex and all triangles depending on that vertex results in a hole in the mesh. This hole is patched by local retriangulation, as shown in Fig. 3. The delete-patch operation is repeated until the removal of any vertex causes a topological violation. The resulting mesh serves as the base mesh, which is the coarsest approximation of the original. It is very simple in comparison with the original mesh. For example, a high-quality sphere mesh may have thousands of vertices and triangles. However, its base mesh is the simplest 3-D mesh, i.e., tetrahedron. The original model can be recovered by adding removed vertices back to the base mesh in reverse order.

For every removed vertex, it is always associated with a list of neighboring vertices. By *neighboring*, we mean the neighborhood of a vertex with respect to the currently updated mesh rather than the original mesh (see Fig. 4). For the first vertex removed from the original mesh, its neighboring vertices are truly its nearest vertices. However, every following removed vertex may have some neighboring vertices that are not its nearest vertices, since some of those vertices may have been removed beforehand and the local topology has been changed. A hierarchical representation of the connectivity of the mesh is then constructed by this neighborhood structure.

We adopt a very simple yet effective criterion to determine the sequence of vertex removal. That is, a vertex is removed if its removal results in the least distortion between the simplified and the original models. Nonessential vertices, such as those vertices coplaned with their neighborhood, are removed first. Vertices that are visually important tend to be significantly different from their neighborhoods and will be preserved in the early stage and removed later. Since vertices are added back to the base mesh in reverse order, visually prominent vertices are put back first, followed by regular vertices with medium distortion and, last, nonessential vertices. This approach ensures that the compressed model converges to the original at a maximum rate. A variety of distortion measurements can be used. We choose the Euclidean distance between the position of removed vertex and the average position of its neighboring vertices. This measurement can be computed easily and reflects well the visual importance of vertices according to our experimental results.

By coding the base mesh and each vertex addition step, we can completely record the original mesh. The base mesh can be coded with a regular graphic format. As to the coding of vertex addition, we consider two different types of data. One type is structure data, which specify where and how the mesh topology is locally updated. The other type is attribute data, which describe the individual information of the new vertex, such as its position, normal, and color. The coding of these two data is detailed in the following two subsections.

B. Coding of Structure Data

We explain the procedure to encode the local topology update with a simple example, as shown in Fig. 3, where the addition (or removal) of one vertex in a local region is illustrated. The first step is to delete all edges inside the region. Then, a new vertex is added somewhere inside the region and connected to all neighboring vertices. This local region is called the neighborhood of the central vertex. This vertex addition (or removal) process can be encoded by recording a list of boundary vertices or composing triangles in the updated mesh. However, these approaches require a lot of coding bits.

In our algorithm, we consider a different approach by implementing a pattern lookup table, where the region is specified by one local and one global index. The local index determines the pattern of the neighborhood topology, while the global index locates the neighborhood in the mesh. This approach can be justified as follows. There is a fixed number of topological patterns associated with a neighborhood. For example, a neighborhood with six boundary edges (i.e., hexagon) has only four partitioning patterns, as shown in Fig. 5. In Table 1, we show the relationship between the maximum number of allowed patterns and the number of boundary edges of a neighborhood up to ten edges. Patterns that differ by a rotational factor are counted in the same entry. Though the number of patterns grows very rapidly as the number of edges (NOE) of a neighborhood increases, the number of edges for most neighborhoods is found to be between five and seven via extensive analysis of the distributions of NOE for many models. Only a small

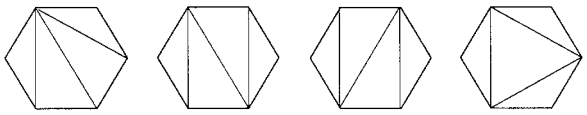


Fig. 5. Patterns of a partitioned hexagon.

Table 1 The Number of Neighborhood Patterns Associated with Polygons Parameterized by Edge Numbers

edges	3	4	5	6	7	8	9	10
patterns	1	1	1	4	6	19	49	150

fraction of neighbors have an NOE larger than ten. Since there are in total 231 patterns with an NOE less than or equal to ten, an 8-bit index set can be used to represent these patterns and stored in both the encoder and the decoder. We impose a constraint in the vertex removal process, namely, only vertices with an NOE less than 11 are eligible for removal. Therefore, every neighborhood thus generated can be uniquely encoded with the 8-bit index set.

For each partitioning pattern in the table, we mark one triangle in the pattern. To encode a neighborhood, we first find its pattern in the lookup table. Then, the index of the marked triangle in the triangle list of the updated mesh is used as the global index to locate the pattern in the updated mesh. Starting from each edge of this triangle, we can find one neighborhood according to the partitioning pattern. We also encode one of three edges that can produce the desired area. The triangle index can be represented compactly. At the early stage of coding, the updated mesh has a smaller number of triangles and requires a smaller number of bits to encode a triangle id. For example, if the total number of triangles is between 32 and 64, only 6 bits are needed to encode the index. Note that both the encoder and the decoder have the full knowledge of the structure of the updated mesh. When the number of triangles becomes larger and requires more bits, both the encoder and the decoder will increase the number of coding bits accordingly.

Experimental results of the proposed coding method applied to the Spock model are listed in Table 3, where $\#v$ is the number of vertices of the mesh at different resolutions and BPS is the averaged number of bits to encode a neighborhood. Our method requires around $\log_2(\#v)+6.0$ bits, which is comparable to the generalized triangle strip [3] and the progressive mesh method [10].

C. Coding of Attribute Data

One essential attribute datum is the vertex position. Other commonly used attribute data include the normal and the color of the vertex. Regardless of the type, all attribute data can be coded with the same framework. Generally speaking, attribute data bear a strong correlation among their neighborhood. Due to the irregularity of the mesh structure, it is very difficult to perform the transform such as DCT or the wavelet transform for energy compaction. Instead, we predict the attribute of a vertex by simply averaging the same attribute of its neighboring vertices. Thus, attribute

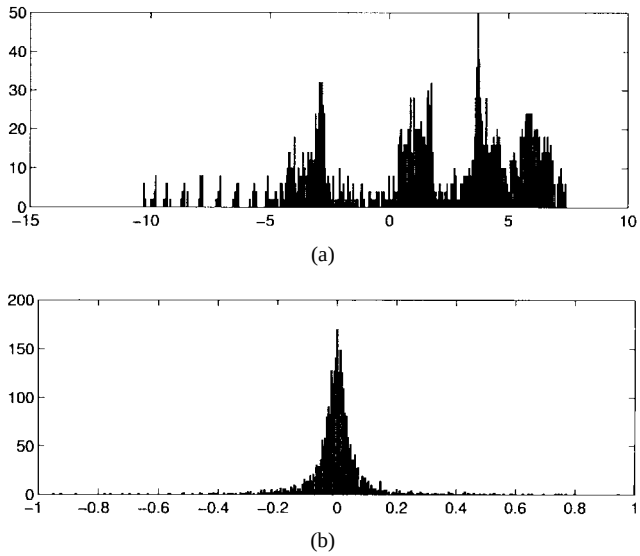


Fig. 6. Comparison of histograms of (a) original data and (b) prediction residual.

values of each vertex have prediction residues. Residues of the same attribute are then arranged into a 1-D array in the same order as the sequence of vertex addition and encoded by the embedded coding method as described in Section II-B.

Histograms of original data and prediction residues are compared in Fig. 6. The x coordinates of all vertices from the dinosaur model are plotted in Fig. 6(a). They are in the interval of [-10, 10] with an irregular distribution pattern. The corresponding prediction residues are plotted in Fig. 6(b). They are highly concentrated in the interval of [-0.5, 0.5], and the distribution fits well with the Laplacian model. As mentioned in Section II-B, we have to consider both identification and refinement bits in the coding process, which have quite different bit-consumption characteristics.

For identification bits, each is encoded by the context arithmetic coder. The context is generated according to the significant identification of its neighboring vertices. For each neighboring vertex, we use 1 bit to represent whether or not its attribute datum of the same kind is significant. We simply concatenate all such bits to create a binary representation of the context. The rationale behind this approach is that attribute data with a similar neighboring circumstance most likely have a similar distribution. Since the probability of "0" is much larger than that of "1," an arithmetic coder might take a fraction of one bit to code an identification symbol 0, while it might take several bits to code a 1. On the average, it still takes much less than 1 bit to code one identification symbol. As far as the refinement is concerned, it has a fixed half-half probability, so that it costs exactly 1 bit. If a residue V_j is very large and $sid(j)=0$, it costs several bits to encode $a_{j,0}=1$. Then, at each following layer, it costs exactly 1 bit to encode each refinement symbol. If V_j is relatively small and $sid(j)$ is large, then each of $a_{j,0}, a_{j,1}, \cdots, a_{j,\mathrm{sid}(j)-1}$, which is zero, only costs a fraction of one bit and a small amount in total. The exponential histogram of prediction residues

guarantees a much better coding performance than the direct binary representation of the original data. The average bit consumption per attribute (BPA) datum versus the number n layers for the Spock model is given in Table 3. For each n, the residue is encoded to the precision of 2^{-n} with respect to the magnitude of the original threshold. It requires n bits to achieve the same performance for a regular binary representation. At beginning layers, BPA is almost the same as n. As the coding proceeds to finer layers, BPA becomes smaller in comparison with n since more residues of small magnitude are encoded.

D. Integration

In the last two subsections, we present two coding procedures. One is the coding of structure data and the other is the coding of attribute data. Each coding scheme produces its own bitstream. In this subsection, we examine the multiplexing of these two bitstreams.

The purpose of multiplexing is not only mixing two bitstreams together. The bitstream of structure data provides information about the mesh structure, while that of attribute data provides information about the model geometry. The quality of the reconstructed model depends on both the number of vertices and the precision of attribute data. The more a bitstream is decoded, the more similar the decoded model is to the original model. Decoding of either bitstream contributes to the reduction of compression distortion. The contribution of bits decreases according to their order in the corresponding bitstream. It is desirable that this property is preserved in the final bitstream as well. This problem is similar to the merging of two arrays, which have been sorted in order according to a certain measurement individually, into one array by the same measurement. In this case, the measurement is "contribute to the distortion reduction." A rate-distortion model is built for each bitstream to study the average distortion reduction per bit so that these two bitstreams can be multiplexed in the proper order.

In the coding of structure data, a vertex removed later is likely to have a larger residue. Rigorously speaking, this is not absolutely correct since the vertex removal is performed locally and the update can change the mesh structure and the residue of neighboring vertices. The altered residue is possibly smaller than that of the vertex just removed. However, the distortions reduced by a vertex addition are in a decreasing order in the global scope. Fig. 7 shows such a decreasing curve calculated based on the dinosaur model. The local fluctuation is due to the effect of local update.

In the coding of attribute data, the quantization residue of each vertex is uniformly distributed in the interval $[-T_{i+1}, T_{i+1}]$ at stage i. Consequently, the distortion reduction at a certain vertex is random. It is not correlated to that of the preceding or the following vertices. Such a rate-distortion curve is shown in Fig. 8.

According to these features, we propose a multiplexing scheme as follows. We first find the prediction residue of every vertex to be removed and determine the maximum magnitude T of all residues. Vertices are added back in a layer-by-layer fashion. Two set of thresholds S_i and T_i are chosen to control the vertex addition and coding of attribute data, respectively. Thresholds T_i are defined as

$$T_0 = \frac{T}{2} \quad \text{and} \quad T_{i+1} = \frac{T_i}{2} \quad \text{for } i = 1, 2, 3, \cdots$$

where T is the largest magnitude of the prediction residues, while S_i is a monotonically decreasing sequence

$$\propto > S_0 > S1 > \cdots.$$

At layer i, all vertices with a prediction residue in the interval $[S_{i-1}, S_i]$ are added back to the mesh in order. For each newly added vertex, its attribute data are immediately coded progressively up to the quantization layer i, which is controlled by threshold T_i. Then, the attribute date of all existing vertices, i.e., old vertices introduced in the previous layers and new vertices introduced in the layer i, are further quantized and encoded up to threshold T_{i+1}. This finishes the coding of information at layer i. The same procedure can be repeated for all layers. After all vertices are added back, only the coding of attribute data is conducted.

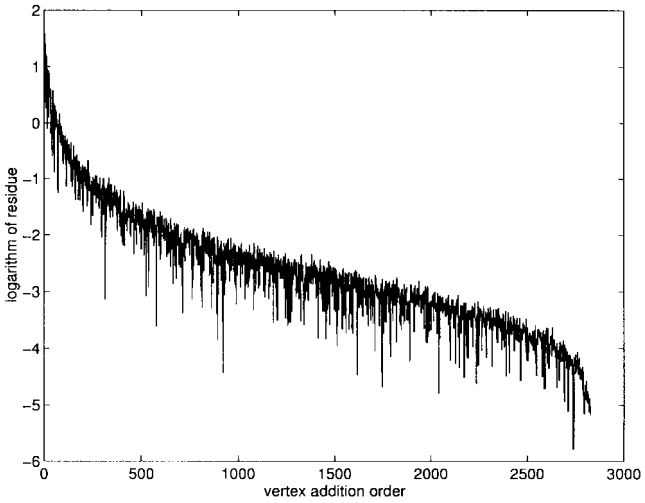


Fig. 7. The rate-distortion curve for structure data.

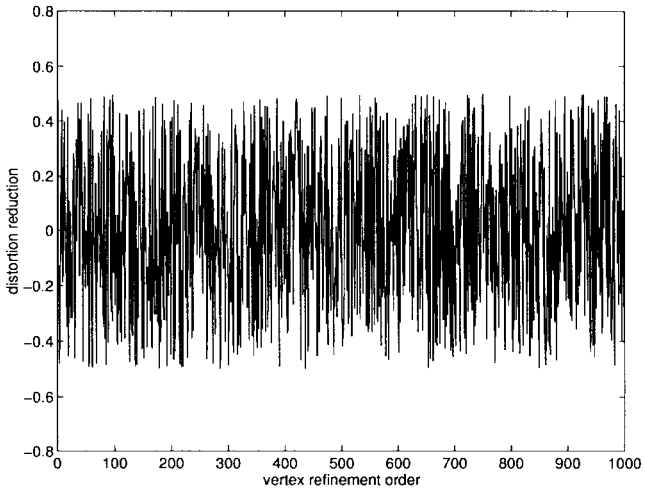


Fig. 8. The rate-distortion curve for attribute data.

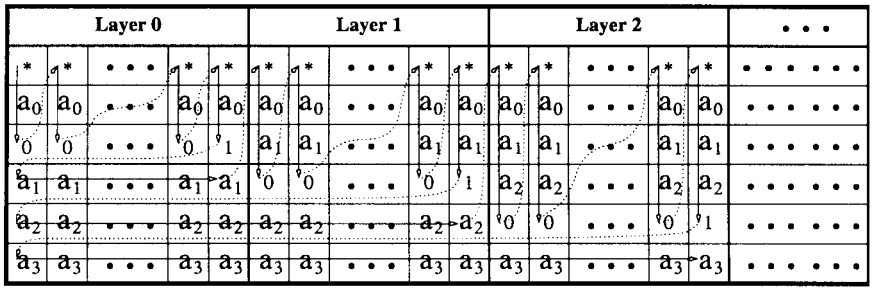


Fig. 9. The multiplexing procedure.

To synchronize the encoder and the decoder, extra bits have to be inserted in the final bitstream to indicate the switch between coding of structure and attribute data. More specifically, after each vertex addition, one extra bit is added to the final bitstream to indicate the type of the next coding process. If no switching is required, the codec adds a new vertex to the updated mesh. Otherwise, the codec starts to code the residues of all existing vertices according to threshold T_{i+1}, then moves to the coding layer $i+1$ and adds a new vertex.

Consider an example with a model of 10 000 vertices and the attribute data having been refined to the twelfth layer. A total of 10 000 extra bits are introduced. Twelve of them have a binary value of "1" to signal switching, and the other 9988 bits have a binary value of "0" to indicate continuing. The entropy of this extra bit sequence is

$$-\frac{12}{10\,000}\log_2\frac{12}{10\,000}-\frac{9988}{10\,000}\log_2\frac{9988}{10\,000}=0.0134.$$

Thus, it requires $0.0134\times 10\,000=134$ bits in total. This amount of bits is negligible compared with the size of the final bitstream.

Fig. 9 illustrates such a multiplexing process. Each column represents the complete information of a vertex addition. The star stands for structure data. For simplicity, we assume only one attribute datum for each vertex, and the index j of $a_{j,i}$ is omitted. The sign bit s_j is also omitted since it is right after the $a_{j,\mathrm{sid}(j)}$ and thus not in a fixed position. The explicit 0 and 1 are synchronization bits. The arrow indicates the actual coding order, where structure data are encoded by the scheme detailed in Section III-B and each $a_{j,i}$ is encoded by the scheme detailed in Section II-B, respectively.

The choice of S_i is decided in such a way that expected distortion reduction of the last vertex introduced at layer i (with prediction residue S_i) is equal to the average distortion reduction of attribute data coded by threshold T_i. Even though there is no close-form solution for S_i, it can be estimated based on a few assumptions. A very common case is that the attribute data include only the vertex position. We choose S_i to be the maximum magnitude of its prediction residue r_x, r_y, and r_z. Without loss of generality, let $r_x = S_i$ and r_y and r_z be uniformly distributed in the interval $[-S_i, S_i]$. Before the vertex addition, the residue is approximated by zero so that the average distortion is

$$\begin{aligned} d_1 &= E\big[r_x^2+r_y^2+r_z^2\big] \\ &= S_i^2+2\times\frac{1}{2S_i}\int_{-S_i}^{S_i}\lambda^2\,d\lambda \\ &= \frac{5}{3}S_i^2. \end{aligned}$$

After the vertex addition, the average distortion is

$$\begin{aligned} d_2 &= E\big[(r_x-\tilde r_x)^2+(r_y-\tilde r_y)^2+(r_y-\tilde r_y)^2\big] \\ &= \left(S_i-\frac{3}{4}S_i\right)^2+2 \\ &\quad\times\frac{1}{2S_i}\left[\int_{-S_i}^{-\frac{S_i}{2}}\left(\lambda+\frac{3}{4}S_i\right)^2 d\lambda+\int_{-\frac{S_i}{2}}^{\frac{S_i}{2}}\lambda^2\,d\lambda\right. \\ &\qquad\left.+\int_{\frac{S_i}{2}}^{S_i}\left(\lambda-\frac{3}{4}S_i\right)^2 d\lambda\right] \\ &= \frac{1}{6}S_i^2. \end{aligned}$$

Here, we assume that attribute data are quantized by threshold $S_i/2$ and encoded by using the coding rule described in Section II-B. By vertex addition, we mean both the coding of its neighborhood and the coding of its attribute data up to the quantization layer i. Thus, the distortion reduction is

$$d_s = d_1 - d_2 = \frac{3}{2}S_i^2.$$

If a vertex addition requires K bits on the average, the distortion reduction per bit (DRPB) for structure data is

$$\mathrm{DRPB}_s = \frac{3}{2}\cdot\frac{S_i^2}{K}.$$

For attribute data refinement quantized by threshold T_i with the uniform distribution assumption, the average distortions before and after refinement, respectively, are

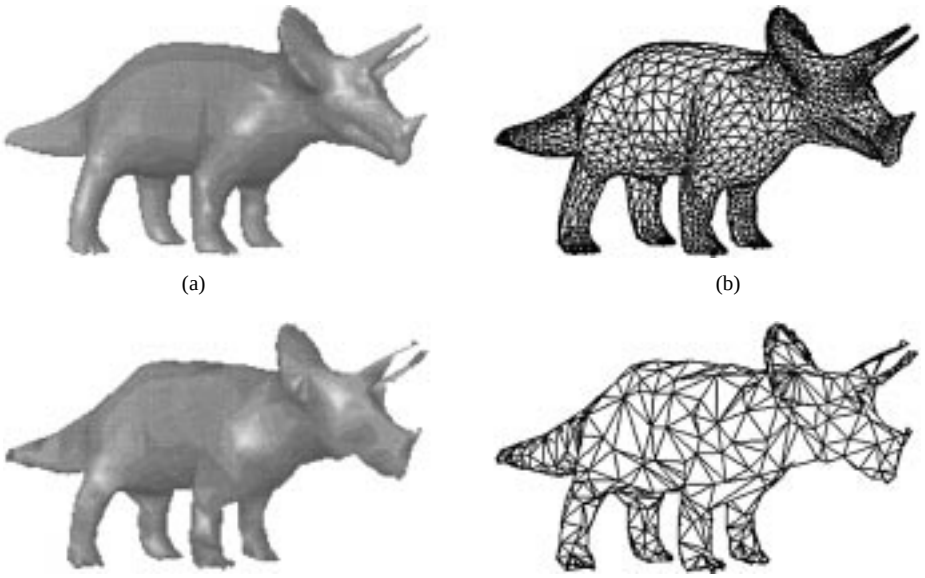
$$d_1 = \frac{T_{i-1}^2}{12} = \frac{T_i^2}{3},\quad d_2 = \frac{T_i^2}{12}.$$

Thus, the distortion reduction can be calculated as

$$d_a = d_1 - d_2 = \frac{T_i^2}{4}.$$

Each refinement requires 1 bit, so that the distortion reduction per bit (DRPB) for attribute data is

$$\mathrm{DRPB}_a = \frac{T_i^2}{4}.$$



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

Fig. 10. Compression of the dinosaur. (a) Original mesh. (b) Original wire frame. (c) 100 : 1 compressed mesh. (d) 100 : 1 compressed wire frame.

By requiring $\mathrm{DRPB}_s = \mathrm{DRPB}_a$, we have

$$S_i = \sqrt{\frac{K}{6}} T_i.$$

Typically, K is between 20 and 28, so that we choose $S_i \approx 2T_i$.

IV. Experimental Results

We have tested our algorithm on four typical 3-D graphic models. They are dinosaur, tube, Spock, and bunny. All models contain six different attribute data: three for the vertex position and three for the vertex normal. The tube model also contains 2-D texture coordinates as part of its attribute data.

To give a rough idea of the performance of the proposed algorithm, we show the original dinosaur model and its wire-frame structure in Fig. 10(a) and (b), respectively. The wire frame of the 100 : 1 compressed model is shown in Fig. 10(d). For this particular case, the compressed model contains around one-tenth the vertices and triangles of the original model. By comparing Fig. 10(b) and (d), it is clear that the compressed model has a much simpler wire-frame structure. Also, the vertex position has a coarser representation, where each coordinate is represented with four quantization layers (in contrast with the 32-bit full resolution). With such a mesh structure and the precision of vertex positions, we are able to render a 2-D image with some distortion, as shown in Fig. 10(c).

More experimental results are listed in Table 2, where $\#v$ is the number of vertices in the model, $\#t$ is the number of triangles, CR is the compression ratio, and SNR is the signal-to-noise ratio. The CR is defined to be the ratio of the original file size to the compressed file size. It is worthwhile to point out that a graphic model is usually stored as an ASCII file. To be properly indented, it contains a large amount of white-space characters such as spaces and tabs, which in fact convey no information. Different indentation styles may require different storage space. Furthermore, a floating or an integer number represented in the ASCII format usually demands more than 32 bits, as required in its binary format. It is desirable to count the original file size with its maximum compactness so that the experimental results are independent of any particular file format. In our calculation, we assume that each attribute datum (floating number) as well as each vertex index (integer) requires 32 bits (or 4 bytes) to estimate the original file size. For instance, the Spock model has 16 386 vertices and 32 768 triangles. Since each vertex has six attribute data and each triangle is specified by three vertex indexes, the original file size in terms of bytes can be calculated as follows:

$$4 \times (16386 \times 6 + 32\,768 \times 3) = 786\,480 \text{ (bytes)}.$$

Table 2 Comparison of the Compression Performance for Different Graphic Models

object	original		compressed		CR	SNR
	#v	#t	#v	#t		
dinosaur	2832	5647	1091	2178	20	41.39
tube	6292	12584	2627	5254	20	44.81
spock	16386	32768	5182	10369	20	45.93
bunny	34835	69473	5688	11217	40	45.03

Table 3 Compression Results of the Spock Model

CR	n	#v	#t	SNR	BPS	BPA
1000	4	150	298	23.50	15.7	4.3
700	4	216	428	25.70	16.5	4.2
400	5	333	662	28.49	17.1	5.0
100	6	1220	2436	35.33	18.7	5.6
50	7	2213	4422	39.84	19.8	6.0
40	7	2734	5464	40.79	20.2	6.2
30	7	3822	7640	41.82	20.6	6.4
20	8	5182	10369	45.93	21.1	6.6
10	9	9708	19412	52.81	22.1	7.1

This estimated file size is about one-third of its actual ASCII file size. The SNR measures the similarity between the original and the reconstructed 3-D graphic models. It is defined as follows:

$$\text{SNR} = 10 \log \frac{\Sigma_0^n ||\vec{V}_i - \vec{V}_c||^2}{\Sigma_0^n ||\vec{V}_i - \vec{\mathcal{V}}_i||^2}$$

where n is the total number of vertices, $\vec{V}_i$ is the original position of the ith vertex, $\vec{V}_c$ is the average position of n vertices (i.e., the centroid of the graphic model), and $\vec{\mathcal{V}}_i$ is the reconstructed position of the ith vertex. Note that a compressed graphic model may not include all n vertices in the original model. In this case, $\vec{\mathcal{V}}_i$ can be obtained via interpolation based on existing vertices in the compressed model.

In the context of image compression, a compressed image with the peak SNR equal to 35 dB or above is considered to be of very good quality. Usually in such a case, the decompressed image appears indistinguishable from the original one to inexperienced eyes. In the context of graphic compression, the quality of a compressed graphic model is influenced by many factors. One obvious factor is the size of rendering area or, equivalently, the viewing distance. If the model is located far away from the viewer, the base mesh may be sufficient to render the object with acceptable quality. On the other hand, if we zoom into a particular small region, even the original model might appear too coarse by itself. Another factor is the viewing angle. A model should be observed from any viewing angle. The same amount of distortion might be negligible from one viewing point but intolerable from another. Thus, a more accurate approximation is required to ensure an overall satisfying performance. According to our experience, a high-quality rendered image with resolution 512×512 pixels can be obtained from a compressed graphic model with SNR in the range of 40–45 dB.

The tradeoff curves between CR and SNR for the four test graphic models are shown in Fig. 11. Since the proposed coding scheme is progressive, we are able to decode only one embedded bitstream to obtain all results given in each curve. As more bits are decoded, the reconstructed model becomes more accurate. Correspondingly, SNR gradually increases while CR decreases. Different compression performances are achieved for different models due to the different degrees of redundancy existing in the models. Based on the figure, we conclude that the bunny model has the highest degree of redundancy, while the dinosaur model has the lowest degree of redundancy. This is consistent with our visual observation of these models. The bunny model is approximately a round shape without many visually important details in the body. Its total number of vertices is 34 835. The dinosaur is more complicated due to the four legs, two ears, two horns, and some special features around the mouth region. However, its total number of vertices is only 2832. For models with a high degree of redundancy, the proposed algorithm removes the redundancy at an early stage of the compression process. The "nonessential" information will be added back for a higher bit budget.

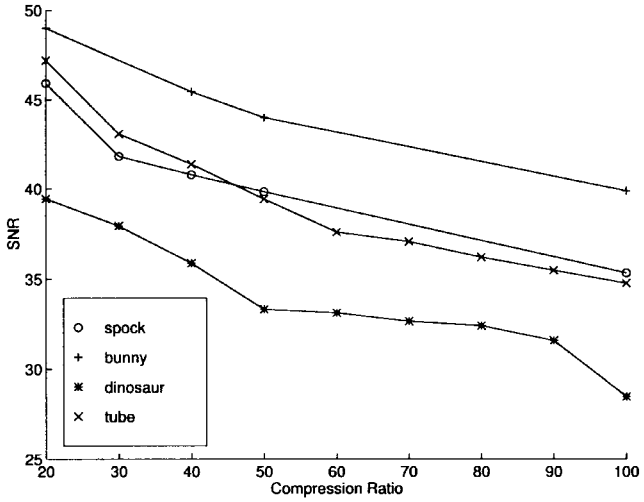


Fig. 11. Rate-distortion performances for embedding coding of graphic models.

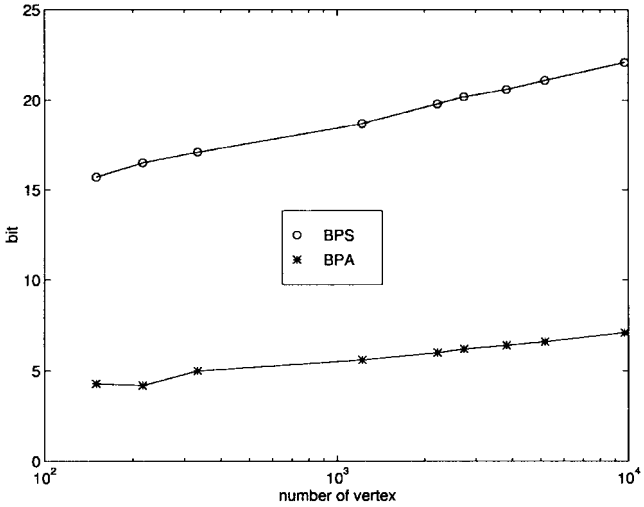


Fig. 12. Plot of bit consumptions as functions of the vertex number for the Spock model.

To illustrate the multiplexing effect between the coding of structure and attribute data, we plot the bit-consumption curve as a function of the vertex number for the Spock model in Fig. 12. The BPS curve denotes the average number of bits spent in the coding of each neighborhood, and the BPA curve denotes the average number of bits spent in the coding of each attribute datum. Both BPS and BPA grow smoothly as the number of vertices increase. This indicates that bits are allocated proportionally in the coding of structure and attribute data as the total bit budget increases. When there are more triangles in the reconstructed model, more bits are required to encode a triangle index so that BPS grows. Similarly, BPA increases since attribute data should be better represented by more quantization layers when the number of vertices increases. It can be shown that both BPS and BPA grow logarithmically with the level of details of the reconstructed model, which ensures the superior performance of our algorithm.

Last, to demonstrate the visual effect of rendered images from the progressively coded graphic models, we show

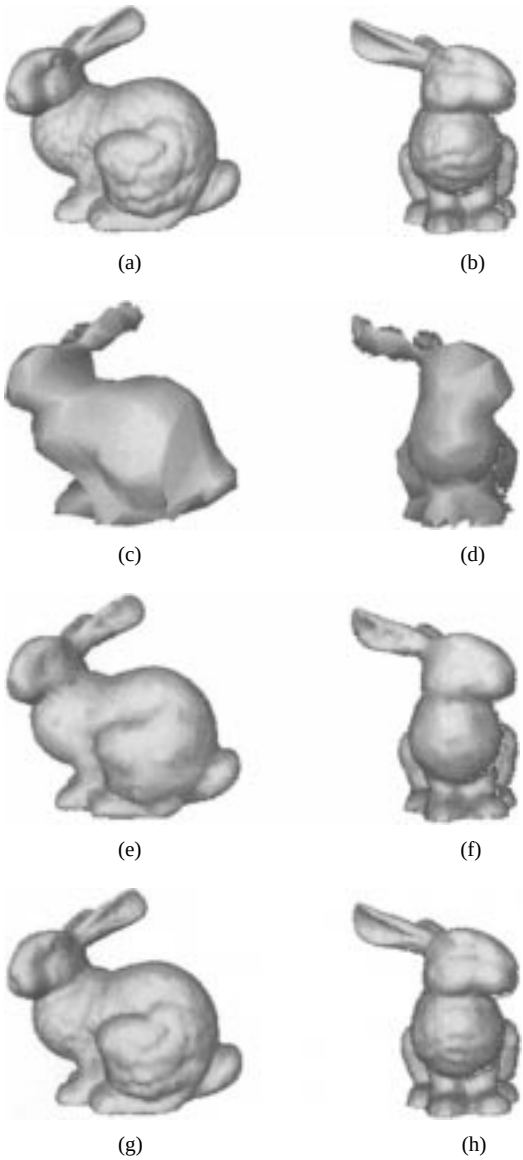


Fig. 13. Compression of the bunny model. (a) Original (side view). (b) Original (front view). (c) 1000 : 1 (side view). (d) 1000 : 1 (front view). (e) 100 : 1 (side view). (f) 100 : 1 (front view). (g) 40 : 1 (side view). (h) 40 : 1 (front view).

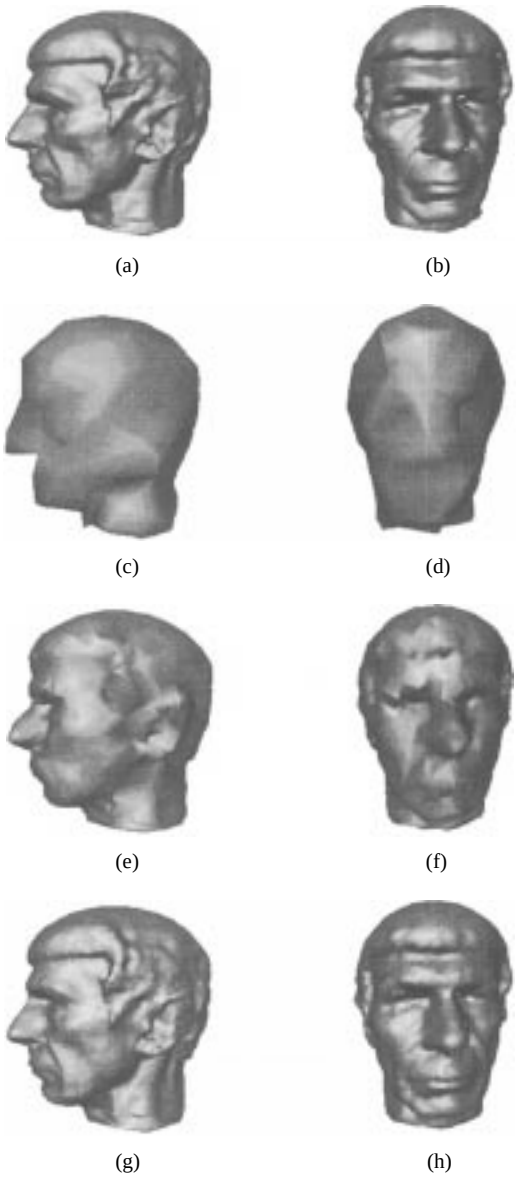


Fig. 14. Compression of the Spock model. (a) Original (side view). (b) Original (front view). (c) 1000 : 1 (side view). (d) 1000 : 1 (front view). (e) 100 : 1 (side view). (f) 100 : 1 (front view). (g) 20 : 1 (side view). (h) 20 : 1 (front view).

two views (front and side) of rendered images at different compression ratios for the bunny and the Spock models in Figs. 13 and 14, respectively. The original bunny mesh with a total of 34 835 vertices is shown in Fig. 13(a). The base mesh has five vertices and five triangles. Meshes with a compression ratio of 1000 : 1 (280 vertices), 100 : 1 (2431 vertices), and 40 : 1 (5688 vertices) are shown in Fig. 13(b)–(d), respectively. The SNR value for the mesh with a compression ratio of 40 : 1 (with about one-sixth of the total numbers of vertices and triangles of the original) is 45.03 dB. In other words, for simple graphic models, we can achieve a compression ratio of 40 : 1 without visible distortion. For more complicated models, such as the dinosaur and Spock, we can obtain a good graphic quality at a compression ratio of around 20 : 1. The Spock mesh, with a total of 16 386 vertices, is shown in Fig. 14(a). The base mesh is a tetrahedron. In Fig. 14(b)–(d), we show meshes corresponding to compression ratios of 1000 : 1 (150 vertices), 100 : 1 (1220 vertices), and 20 : 1 (5182 vertices), respectively. The SNR value for the last case is 45.93 dB. It is a high-quality replica of the original mesh, even though it has only about one-third of the vertices of the original one.

V. Conclusion and Extension

A progressive compression algorithm, which encodes a 3-D geometric model into an embedded bitstream, was studied in this work. A 3-D geometric model consists of two kinds of data: structure data and attribute data. Structure data characterize the connectivity information among vertices, while attribute data describe other relevant information of vertices, such as positions, colors, and normals. They are encoded separately according to their importance and then integrated into a single bitstream. In

decoding, the decoder decodes from the bitstream the most important information first and gradually adds finer detailed information to provide a more accurate approximation. The decoder can stop at any point while giving a reasonable reconstruction of the original model. We applied the proposed algorithm to several test 3-D meshes and achieved a compression ratio of 20 : 1 while maintaining an excellent graphic quality. The resulting compression method not only allows a progressive representation of a 3-D graphic model but also maintains an excellent rate-distortion performance. It is expected that many applications, including progressive display and level-of-detail control, will benefit from the proposed coding scheme.

There are still several interesting problems to be solved in making our work more complete. One is the smooth transition between the coarse- and fine-resolution models. With the current scheme, we observe sometimes an abrupt change in a certain part of the object due to the removal or addition of a vertex. This effect is visually annoying, and it is desirable to make the transition as smooth as possible. Another is the efficient rendering of progressively coded graphic models. This appears to be a very challenging problem with great impact.

REFERENCES

[1] R. Bar-Yehuda and C. Gotsman, "Time/space tradeoffs for polygon mesh rendering," *ACM Trans. Graph.*, vol. 15, no. 2, pp. 141–152, Apr. 1996.

[2] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwat, F. Brooks, and W. Wright, "Simplification envelopes," in *Proc. Computer Graphics Ann. Conf. Series*, New Orleans, LA, Aug. 1996, pp. 119–128.

[3] M. Deering, "Geometry compression," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1995, pp. 13–20.

[4] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution analysis of arbitrary meshes," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1995, pp. 173–182.

[5] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1997, pp. 209–217.

[6] A. Guéziec, "Surface simplification inside a tolerance volume," IBM T. J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC-20440, 1996.

[7] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang, "Voxel based object simplification," in *Proc. Visualization*, 1995, pp. 296–303.

[8] P. Heckbert and M. Garland, "Multiresolution modeling for fast rendering," in *Proc. Graphics Interface'94*, Canadian Information Processing Society, Banff, Alta., Canada, May 1994, pp. 43–50.

[9] P. Hinker and C. Hansen, "Geometric optimization," in *Proc. Visualization*, 1993, pp. 189–195.

[10] H. Hoppe, "Progressive meshes," in *Proc. Computer Graphics Ann. Conf. Series*, New Orleans, LA, Aug. 1996, pp. 99–108.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1993, pp. 19–26.

[12] A. D. Kalvin and R. H. Taylor, "Superface: Polyhedral approximation with bounded error," in *Proc. SPIE Medical Imaging*, Y. Kim, Ed., Feb. 1994, vol. 2164.

[13] M. Lounsbery, "Multiresolution analysis for surfaces of arbitrary topological type," Ph.D. dissertation, Univ. of Washington, Seattle, Oct. 1993.

[14] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand, 1993.

[15] J. Popović and H. Hoppe, "Progressive simplicial complexes," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1997, pp. 217–225.

[16] J. Rossignac and P. Borrel, *Modeling in Computer Graphics: Methods and Applications*. Berlin, Germany: Springer-Verlag, 1993.

[17] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.

[18] G. Schaufler and W. Stürzlinger, "Generating multiple levels of detail from polygonal geometry models," in *Proc. Virtual Environments Eurographics Workshop*, Jan. 1995, pp. 33–41.

[19] W. J. Schröeder, "Decimation of triangle meshes," in *Proc. Computer Graphics Ann. Conf. Series*, July 1992, pp. 65–70.

[20] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Image Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[21] M. Soucy and D. Laurendeau, "Multiresolution surface modeling based on hierarchical triangulation," *Comput. Vision Image Understanding*, vol. 63, pp. 1–14, Jan. 1996.

[22] W. Sweldens and P. Schröder, "Spherical wavelets: Efficiently representing functions on the sphere," in *Proc. Computer Graphics Ann. Conf. Series*, Aug. 1995, pp. 161–172.

[23] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," IBM T. J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC-20340, 1996.

[24] D. Taubman and A. Zakhor, "Multirate 3D subband coding of video," *IEEE Trans. Image Processing*, vol. 3, pp. 572–588, Jan. 1994.

[25] G. Turk, "Re-tiling polygon surfaces," in *Proc. Computer Graphics Ann. Conf. Series*, July 1992, pp. 55–64.

[26] A. Varshney, P. K. Agarwal, F. P. Brooks, Jr., W. V. Wright, and H. Weber, "Generating levels of detail for large-scale polygonal models," Dept. of Computer Science, Duke University, Durham, NC, Tech. Rep., 1995.

Jiankun Li received the B.S. degree in physics from the University of Science & Technology, China, in 1993. He currently is pursuing the Ph.D. degree in the Department of Electrical Engineering-Systems at the University of Southern California, Los Angeles.

His research interests include natural and synthetic image compression, graph compression, postprocessing, and multiresolution graphic rendering.

C.-C. Jay Kuo (Senior Member, IEEE) received the B.S. degree from the National Taiwan University, Taipei, 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.

From October 1987 to December 1988, he was a Computational and Applied Mathematics Research Assistant Professor in the Department of Mathematics at the University of California, Los Angeles. 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, Los Angeles, where he currently has a joint appointment as Associate Professor of Electrical Engineering and Mathematics. His research interests are in the areas of digital signal and image processing, audio and video coding, wavelet theory and applications, multimedia technologies, and large-scale scientific computing. He is the author of more than 280 technical publications in international conferences and journals. He is Editor-in-Chief of the *Journal of Visual Communication and Image Representation*.

Dr. Kuo is a member of the Society for Industrial and Applied Mathematics and the Association of Computing Machinery. He is a Fellow of the International Society for Optical Engineering. He was Associate Editor of 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 and Presidential Faculty Fellow Award in 1992 and 1993, respectively.