Progressive Compression of 3D Graphic Models

Jiankun Li, Jin Li and C.-C. Jay Kuo
Department of Electrical Engineering-System
University of Souther California
http://sipi.usc.edu/~cckuo

Abstract— **A progressive coding method which encodes a 3D graphic model into an embedded bit stream is investigated in this research. The coder first encodes the coarsest resolution of the model, and then includes the information of finer details gradually. The embedding property allows the decoder to terminate the decoding at an arbitrary position in the compressed bit stream, and more accurate model can be obtained when more bits are decoded. Numerical experiments are provided to demonstrate the excellent rate-distortion performance and progressive compression performance of the new method.**

I. INTRODUCTION

3D graphic models become increasingly popular since the advent of 3D laser scanning system and the boom of VRML (Virtual Reality Modeling Language) models. Generally speaking, these models are expensive to render, awkward to edit and costly to transmit through the network. Wider applications of 3D graphics could be potentially limited due to these obstacles. To reduce the storage space and transmission bandwidth, it is desirable to compress 3D graphic models with lossy compression method which keeps the distortion within a certain tolerable level while maximizing the data reduction objective. Another important consideration is to apply compression in a progressive fashion to allow easier control of data such as progressive display and level-of-detail control. It requires that the information is stored at different resolutions. The most important information of the model is encoded first, and finer details are added gradually. As a result, the decoder can construct the models of different resolutions, from the coarsest approximation to the finest replica, depending on the application requirement.

Compared with the large amount of work on image and video compression, research on the compression of 3D graphic models is relatively few. Over years, there are quite a few papers on simplification and succinct representation of graphic models, e.g. [1]-[4], [7], [8], [10], [11]. Turán [10] gave a binary representation of an unlabeled planar graph with n vertices whose length is guaranteed to be no greater than $12n$. Schröeder [7] proposed a decimation algorithm that significantly reduces the number of polygons required to represent an object by repeatedly removing non-essential vertex. Deering [2] introduced the concept of *generalized triangle strip* which allows a compact representation of a planar graph by using a linear data structure and a set of stack operators. Lounsbery and DeRose [5] established a wavelet transform defined on arbitrary 3D domain to compress a graphic model at several detail levels. Taubin and Rossignac [8] represented a triangulated mesh by using two interlocked trees which compress the connectivity information into, roughly, an average of two bits per triangle. Hoppe [3] constructed a progressive mesh by recursively applying the edge collapse operation on an arbitrary mesh.

Although promising, previous work lacks some important ingredients to achieve a better performance. First, the compression of attribute data associated with each vertex, such as vertex position, normal and color, was performed by the traditional run length coding in a single solution manner. Second, there is no integration between the coding of the structure data of the model and the coding of the attribute data. Thus, in this sense, none of schemes proposed before can be viewed as a fully multiresolution algorithm.

In this work, we improve both deficiencies and propose a new hierarchical 3D graphic compression scheme. This scheme progressively compresses a model, including both structure and attribute data, into a single bit stream. Along the encoding process, every output bit contributes to reduce the distortion, and the contribution of bits decreases according to their order of position in the bit stream. At the receiver end, the decoder decodes from the bit stream the most important information first and then finer details. 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 resolution can thus be con-

0-8186-7819-4/97 $10.00 © 1997 IEEE

structed from a single bit stream. This functionality is also referred to as the embedding property since the coding of a coarser model is embedded in the coding of a finer model.

This paper is organized as follows. An overview of the proposed compression scheme is first presented in Section 2. The coding of the mesh structure and mesh attribute data is presented in Sections 3 and 4, respectively. Section 5 describes a complete scheme which integrates the coded bit stream for the structure and the attribute information into a single one. Numerical Examples of mesh compression are performed to demonstrate the superior compression performance of the proposed coding scheme in Section 6. Concluding remarks are given in Section 7.

II. OVERVIEW

There are two types of data in a 3D graphic model, i.e. structure data and attribute data. Structure data specify the connectivity information among vertices to characterize the topology of the model. Most 3D graphic files consist of a list of polygons each of which is specified by its vertex indices to specify its structure. Attribute data describe other relevant information of the vertex or polygon, including the position, the normal, and application specific information such as the temperature distribution or the flux vector at each vertex.

To recover the model accurately, structure data have to be coded with a lossless compression method and, consequently, it is difficult to achieve a high compression ratio based on these data. This becomes a major obstacle for 3D graphic compression, especially in low bit rate coding applications. On the other hand, the attribute data bear a very strong local correlation with those of neighboring vertices. They can be compressed with lossy compression methods while keeping the error under a certain tolerable level. Many compression techniques have been developed for this purpose and a high compression ratio can be achieved.

To make the overall compression scheme progressive, both structure and attribute data have to be compressed progressively. This can be implemented by building a hierarchical structure for a 3D graphic model. To be more precise, we make multiple passes through all vertices in the original mesh to organize vertices into a layered structure. The removal of a vertex and all triangles depending on the vertex from a coarse resolution layer results in a hole in the mesh. This hole is filled by local re-triangulation as shown in Fig. 1.

The delete-fill operation is repeated until further the removal of any vertex could result in a topo-

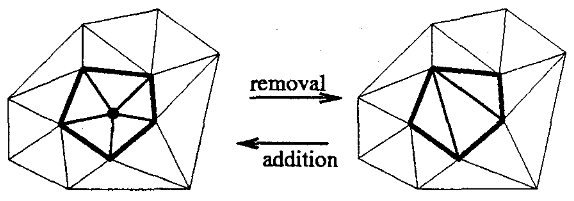


Figure 1: Removal and addition of a vertex.

logical violation. The resulting mesh serves as the base mesh which is the coarsest approximation of the original mesh connectivity. 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 3D mesh: a tetrahedron. Only a few bits is consumed to specify the base mesh. 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 current updated mesh rather than the original mesh structure. The neighborhood is encodes by an expansion method as detailed in Section III. The mesh structure is locally updated according to its neighborhood as shown in Fig.1. This leads to a progressive representation of mesh connectivity. For each deleted vertex, we predict its attribute data by the average of those of its neighboring vertices. Then, the residue of attribute data is coded by using the run length coding. In contrast with the coefficient-by-coefficient encoding approach, we adopts a new approach in which each datum is successively quantized into a certain number of bits. The most significant bits of all data are grouped together to form one layer and encoded first. Then, we move to the layer of the second significant bits and so on. Such a coder is consistent with the importance of each bit so that important information is always encoded first.

The coding of structure data and attribute data have to be integrated into one complete algorithm. That is, we use bits to either introduce a new vertex (coding of structure), or refine the attribute data of the existing vertex (coding of attribute). A rule has to be set to decide what kind of coding the next bit should be assigned to. We adopt a solution by encoding the part which decreases the distortion most at the moment of encoding. Note that it is very hard to achieve a high compression ratio on the coding of structure data as mentioned before. However, we are able to achieve this goal by encoding data in a hierarchical way since we need not encode the entire structure data at the very beginning. The integration is important for the quality control. The quality of the decompressed model is affected by two factors: the number of vertices of the decompressed model and the precision of attribute data of these vertices. An optimal performance can be obtained if the integration keeps a good balance between these two factors. The integration will be described in detail in Section V..

III. Coding of Structure Data

Although the proposed compression scheme can be applied to an arbitrary mesh, we focus on a mesh composed by triangles for simplicity in this section. The same approach can be generalized to arbitrary meshes in a similar way.

According to the local topology, a 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 edge is not shared by two triangles, or 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 semi-cycle of triangles (see Fig. 2 (c)).

In order to preserve the topology, complex vertices are not deleted from the mesh. Only simple and boundary vertices are candidates for removal.

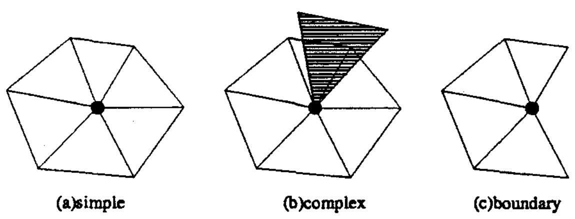


Figure 2: Classification of three types of vertices.

The removal of a vertex results in a loss of information and introduces a certain level of distortion. To optimize the rate-distortion performance, one should first add the most important vertex from a lower to a finer resolution. On the other hand, one should first remove the most non-essential vertex from a finer to a lower resolution. Since the vertex position can be predicted by the average of its neighboring vertices, the prediction residue is used as a measure for its importance. Thus, in the process of vertex removal, the rule is to remove the vertex whose residue has the smallest magnitude. In each removal, the vertex and its links are deleted. The resulting hole is filled by retriangulation of its neighboring vertices. To enable the addition of that vertex in decoding, this neighborhood information has to be encoded. One simple way is to specify the cycle by a list of vertex indices or the area by a list of triangle indices. However, both methods require a lot of coding bits.

Since both the encoder and the decoder have the full knowledge of the structure of the updated mesh, one can grow a neighborhood by specifying a root triangle followed by a local growing path. This growing method requires much less coding bits. The root triangle is specified by a global triangle index while the growing path is an array of boolean variables. Any triangle within the neighborhood area can be chosen as the root triangle. The growing path is determined as follows. First, every edge within the neighborhood is labeled with '1' while every edge on the boundary of the neighborhood is labeled with '0'. Then, all edges are traversed, starting from the first edge of the root triangle. Along the traversal, edge labels are added in order into the growing path.

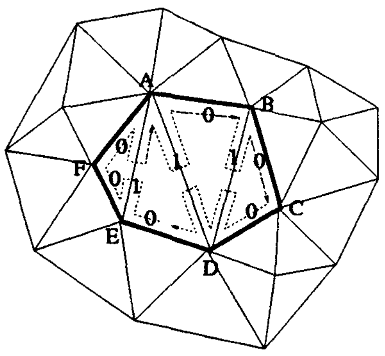


Figure 3: Expansion of a neighborhood.

A simple example is given in Fig.3 to illustrate the proposed algorithm. The hexagon $ABCDEF$ has nine edges, where six of them, i.e. AB, BC, CD, DE, EF, FA are boundary edges labeled with '0' and the other three edges EA, AD, DB are internal edges labeled with '1'. We choose ΔDEA to be the root triangle and DE the edge to begin with. The traversal is counterclockwise. Every boundary edge is visited once and its label '0' is added to the growing path immediately after being traversed. Every internal edge is crossed twice. However, its label '1' is only added once to the growing path after the first crossover since a simple book keeping can automatically detect the second crossover. For example, after we cross AE, visit EF, visit FA, the only possible traversal step is crossing AE again. Then, there is no need to add another '1' to the growing path. Obviously, guided by the grow path, where '1' means crossing the currnt edge and '0' means going through the edge, the decoder can regenerate the traversal procedure and reconstruct the neighborhood area.

The root triangle index can be compressed losslessly. At the early stage of compression, the intermediate mesh has a smaller number of triangles, thus requiring a smaller number of coding bits. For example, if the total number of triangles at a certain level is 64, we only need 6 bits to encode the index. Again, both the encoder and the decoder have the full knowledge of the structure of the updated mesh. Consequently, when the triangle list grows longer and requires more coding bits, both encoder and decoder can be aware of the change at an appropriate time and increase the number of coding bits accordingly.

Suppose that the number of vertices is n and the number of triangles is approximately $2n$. Then, it requires $n(\log_2 n)$ bits of storage. An vertex has average 6 neighbors and there are 9 edges in its neighboring areas (as shown in Fig.3). The grow path can be encoded with 9 bits so that, on the average, this structure representation requires $(\log_2(n) + 9)$ bits per vertex. Experimental results are given in Table 2.

IV. Coding of Attribute Data

A. Quantization

Most conventional coding methods adopts the one-step quantization scheme. The goal is to map the input signal onto a finite index set which can be conveniently converted to intermediate symbols and encoded by an entropy coder. In the proposed new scheme, we adopt a different quantization procedure similar to the approach introduced by Taubman and Zakhor [9]. It is a successive quantization procedure by using a gradually refined step size. Two symbols "0" (insignificant) or "1" (significant) are produced at each quantization step for each attribute datum to form a layer of bits. Depending on whether a bit in one of the previous layers has been identified as nonzero, we consider two different rules: (1) significance identification and (2) refinement quantization.

To begin with, we apply the significance identification rule only to determine the bit-layer L_0. The initial quantization step T_0 for this layer is chosen to be one half of the maximum magnitude of the attribute data. For each attribute datum V_j, if its magnitude is greater than threshold T_0, we use symbol 1 to denote its significance and record its sign as well. Otherwise, we generate symbol 0. In mathematics, we have

$$\begin{array}{lll} V_j > T_0, & S_{j,0} = 1, & E_{j,0} = V_j - 1.5 \cdot T_0, \\ V_j < -T_0, & S_{j,0} = -1, & E_{j,0} = V_j + 1.5 \cdot T_0, \\ \text{otherwise}, & S_{j,0} = 0, & E_{j,0} = V_j, \end{array}$$

where symbol $E_{j,0}$ in the last column denotes the quantization residue at layer L_0. For bit layer L_{i+1}, $i = 0, 1, \cdots$, we consider a successively refined quantization step size defined by $T_{i+1} = T_i/2$. For each attribute datum, if it is marked as insignificant in all previous layers, the significance identification rule is applied. Otherwise, the refinement quantization rule is applied. That is,

1. significance identification:

$$\begin{array}{lll} V_j > T_i, & S_{j,i} = 1, & E_{j,i} = V_{j,i} - 1.5 \cdot T_i, \\ V_j < -T_i, & S_{j,i} = -1, & E_{j,i} = V_{j,i} + 1.5 \cdot T_i, \\ \text{otherwise}, & S_{j,i} = 0, & E_{j,i} = V_j, \end{array}$$

   where symbol $E_{j,i}$ in the last column denotes the quantization residue at layer L_i.

2. refinement quantization:

$$\begin{array}{lll} E_{j,i-1} \geq 0, & R_{j,i} = 1, & E_{j,i} = E_{j,i-1} - T_i, \\ E_{j,i-1} < 0, & R_{j,i} = -1, & E_{j,i} = E_{j,i-1} + T_i. \end{array}$$

Note that symbols $S_{j,i}$ and $R_{j,i}$ are generated with a decreasing importance order to form an embedded coding bit stream.

B. Context Arithmetic Coding

We adopt the context adaptive arithmetic coder, which is used in the JPEG extended system, to encode the significance identification symbols S_i, $i = 0, 1, 2, \cdots$ for layer L_i. An arithmetic encoder encodes the symbol using the probability estimate built inside. In the process of coding, it keeps an approximate count of the 0's and 1's, and change the probability estimate accordingly. That means the coder dynamically estimates probabilities as it codes. The context arithmetic coder is a set of independent arithmetic encoders. Each encoder is referred to by a context index and has its own probability estimate. The symbol to be encoded and its context index are fed to the encoder set and the symbol is encoded by the arithmetic coder referred to by that index. For a sequence of input symbols, the actual probability model of the source is usually not know a priori. However, symbols with a similar probability estimate can be assigned with the same context index and encoded by the same arithmetic coder. The adaptiveness of the arithmetic coder significantly improves the compression performance. We refer to [6] (Chapters 12–14) for more detailed discussion.

The context index in our coder is generated according to the significance identification decision of its neighboring vertices. For each neighboring vertex, we use 1 bit to represent the current significant status of the symbol and simply concatenate all such bits to create a binary representation of the context index. The rational behind this scheme is that attribute data with the same neighboring circumstance most likely have the same significance identification result.

The refinement symbol R_i is distributed evenly to be 1 or -1. Therefore, we adopt an equal probability arithmetic coder in which the probability estimate is fixed at one half.

V. INTEGRATION

In the last two sections, we presented two coding schemes. One is the coding of structure data and the other is the coding of attribute data. Each coding scheme produces its own bit stream, and the contribution of bits decreases according to their order in both bit streams. The quality of the compressed model depends on both the number of vertices and the precision of attribute data. In this section, we examine the integration of these two bit streams into one single bit stream with the objective to preserve the same embedding property.

The rate distortion model of these two coding schemes have a very distinctive feature. For the coding of structure data, the vertex is introduced in the order of the magnitude of its prediction residue so that the distortion reduction by a vertex addition is most likely in a decreasing order. For the coding of attribute data , at a certain threshold T_i, the quantization residue of each vertex is uniformly distributed in the interval $[-T_i, T_i]$, the distortion reduction at a certain vertex is random and only the average distortion reduction is meaningful.

According to these features, we propose an integration 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 attribution data, respectively. Thresholds T_i are defined as

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

while S_i is a monotonically decreasing sequence

$$\infty = S_{-1} > S_0 > S1 > \cdots.$$

At layer i, all vertices with prediction residue in the interval $[S_{i-1}, S_i]$ are added back to the mesh in order. For each newly added vertex, its attribution data are immediately coded progressively up to the quantization layer $i-1$, which is controlled by threshold T_{i-1}. Then, the attribute date of all existing vertices, including 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. This finishes the coding of information at layer i. The same procedure can be repeated for all layers.

The choice of S_i is decided in such a way that the expected distortion reduction of the last vertex introduced in 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. Then, we choose S_i to be the maximum magnitude of its prediction residue r_x, r_y, 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

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

After the vertex addition, the average distortion is

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

Here, we assume that attribute data are quantized by threshold $S_i/2$ and coded by using the coding rule described in Section IV.. Thus, the distortion reduction is

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

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

$$DRPB_s = \frac{3}{2} \cdot \frac{S_i^2}{K}.$$

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

$$\begin{aligned} d_1 &= \frac{T_{i-1}^2}{12} = \frac{T_i^2}{3}, \\ d_2 &= \frac{T_i^2}{12}. \end{aligned}$$

Thus, the distortion reduction can be calculated as:

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

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

$$DRPB_a = \frac{T_i^2}{4}.$$

By requiring $DRPB_s = DRPB_a$, we have

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

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

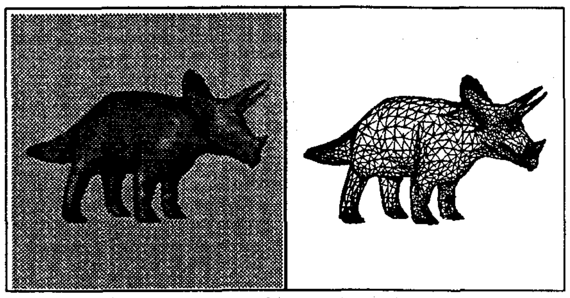
VI. EXPERIMENTAL RESULTS

We have tested our algorithm on various models. Because these models have different degrees of redundancy, different compression performances were achieved. In Table 1, we show the experimental results on various models, where #v is the number of vertices in the model and #v the number of triangles, cr stands for the compression ratio and snr signal-to-noise ratio. We assume in the original model that each vertex index is stored as a 32-bit integer and and that each attribute datum is stored as a 32-bit floating point number. All models include 6 different attribute data: 3 for the vertex position and 3 for the vertex normal. Model Tube includes extra 2D texture coordinates as its attribute data. We show in Fig.4 the original dinosaur model and the 100:1 compressed model with their wireframe structures.

Table 1: Compression Performance

object	original		replica		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

For models with much redundancy, our algorithm can remove the redundancy at an early stage of compression. Only when the compression bit budget is large enough, the non-essential information will be coded. As a result, we can encode the model at a high compression ratio. One such an example is illustrated in Fig.6, where (a) is the original bunny mesh with a total of 34835 vertices, The base mesh has 5 vertices and 5 triangles. (b) is the mesh with a compression ratio 1000:1 and 280 vertices, (c) is the mesh with a compression ratio 100:1 and 2431 vertices, and (d) is the mesh with a compression ratio 40:1 and 5688 vertices. The SNR for mesh (r) is 45.03dB.


(a) original mesh and its wireframe

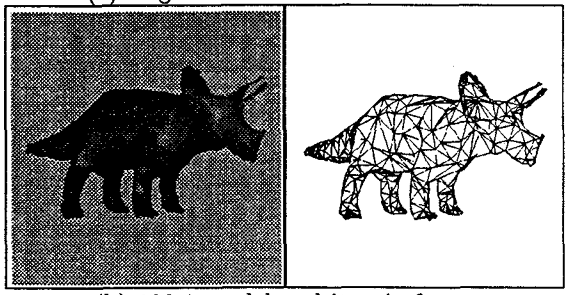

(b) 100:1 model and its wireframe

Figure 4: Compression of the dinosaur.

For models which compactly represents an object, we are able to demonstrate a good graphic quality at a compression ratio around 20:1. One example is given in Fig.7 with the performance measure summarized in Table 2, where BPS is the average number of bits per vertex to encode the structure data, including the root triangle index and the grow path, BPA is the average number of bits per vertex to encode a single attribute datum. The original spock mesh has a total of 16386 vertices as shown in Fig.7 (a). The base mesh is a tetrahedron. In Fig.7 (b), (c) and (d), we show the meshes corresponding to a compression ratio of 1000:1 with 296 vertices, a compression ratio of 100:1 with 1220 vertices, and a compression ratio of 20:1 with 5182 vertices. The SNR for the last result is 45.93dB . It is a very high quality replica of the original mesh even though it only has about one third vertices of the original one. As more bits are decoded, the recovered model becomes more accurate. As show in Fig.5, BPS grows slightly since there are more triangles in the recovered model. The same observation applies to BPA since attribute data are more precisely represented.

Table 2: Experimental results of the spock

CR	#v	#t	SNR	BPS	BPA	dtime
1000	296	590	23.50	15.7	4.3	596
100	1220	2436	35.33	18.7	5.6	598
50	2213	4422	39.84	19.8	6.0	618
40	2734	5464	40.79	20.2	6.2	618
30	3822	7640	41.82	20.6	6.4	618
20	5182	10369	45.93	21.1	6.6	648
10	9708	19412	52.81	22.1	7.1	669

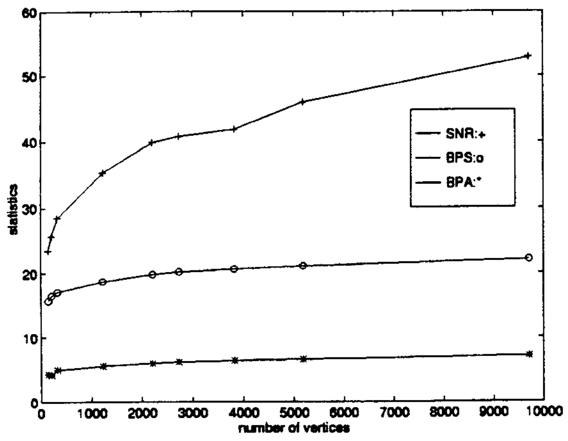


Figure 5: Performance of the spock.

An excellent property of this algorithm is the fast decoding. For each vertex addition, the decoding only involves the local structure update and the attribute data refinement up to several layers. The decoding time is linear with the number of vertices. The last column of Table 2 gives the average decoding time(in *ns*) for each vertex, obtained on a SGI indy with MIPS R4600 and 64MB memory.

VII. CONCLUSION

A progressive compression algorithm, which encodes a 3D geometric model into an embedding bit stream, was studied in this work. A 3D geometric model consists of two kinds of data: structure data and attribute data. They are encoded separately according to their importance and integrated into a single bit stream. In decoding, the decoder retrieves from the bit stream the most important information and gradually adds finer detailed information to provide a more complete 3D graphic model. The decoder can stop at any point while giving a reasonable reconstruction of the original model. We applied the proposed algorithm to complicated 3D meshes and achieved a compression ratio of 20:1 while maintaining a good graphic quality. It is expected that many applications such as progressive display and level-of-detail control will benefit from this progressive compression algorithm.

ACKNOWLEDGMENT

This work was supported by USC-IMSC. Bunny model provided courtesy of Stanford Computer Graphics Laboratory. Spock model provided courtesy of Computer Science Department, University of Washington.

References

[1] R. Bar-Yehuda, "Time/space tradeoffs for polygon mesh rendering," *ACM Trans. on Graphics*, Vol. 15, pp. 141-152, Apr. 1996.

[2] M. Deering, "Geometry compressiom," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 13-20, ACM SIGGRAPH, August 1995.

[3] H. Hoppe, "Progressive meshes," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 99-108, ACM SIGGRAPH, August 1996.

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 19-26, ACM SIGGRAPH, August 1993.

[5] M. Lounsbery, *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*, Ph.D. dissertation, University of Washington, Oct. 1993.

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

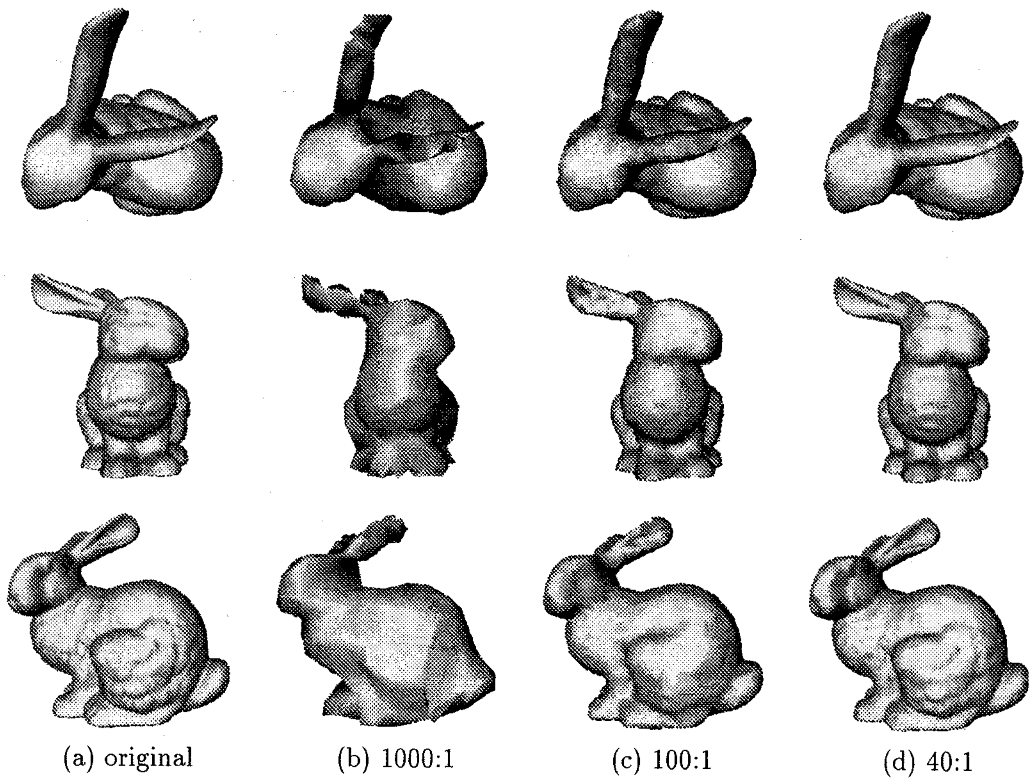
[7] W. J. Schröeder, "Decimation of triangle meshes," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 65-70, ACM SIGGRAPH, July 1992.

[8] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," Tech. Rep. RC-20340, IBM Watson Research Center, Jan. 1996.

[9] D. Taubman and A. Zakhor, "Multirate 3D subband coding of video," *IEEE Trans. on Image Processing*, Vol. 3, No. 3, pp. 572-588, 1994.

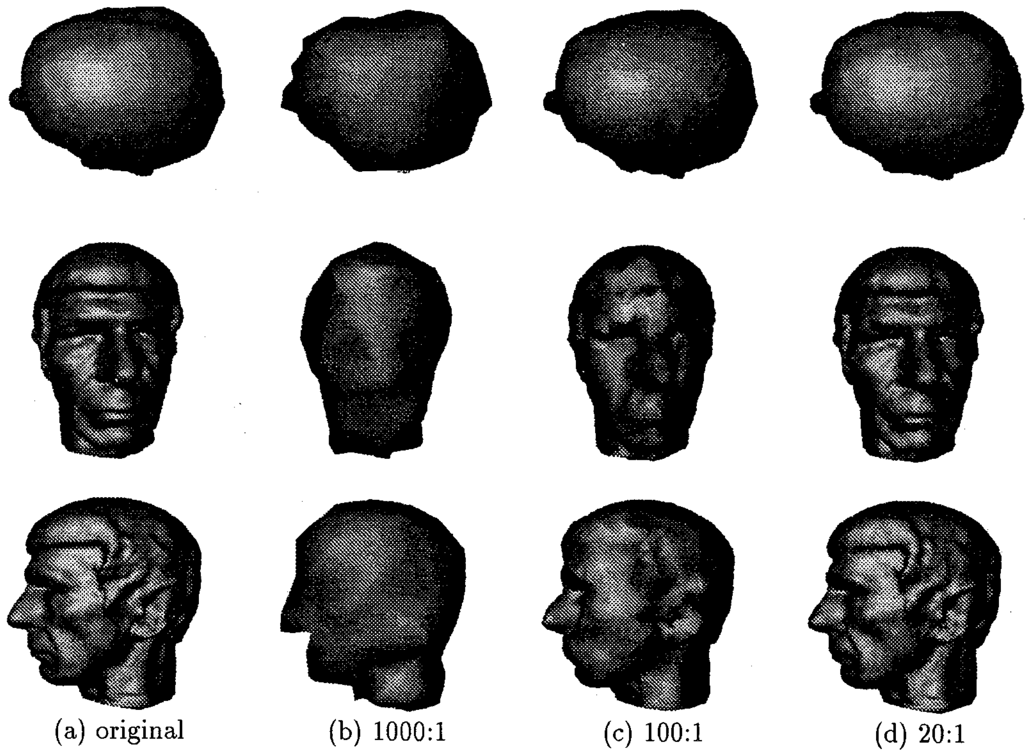
[10] G. Turán, "On the succinct representation of graphs," *Discrete Applied Mathematics*, Vol. 8, pp. 289-294, 1984.

[11] G. Turk, "Re-tiling polygon surfaces," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 55-64, ACM SIGGRAPH, July 1992.



(a) original (b) 1000:1 (c) 100:1 (d) 40:1

Figure 6: Compression of the Bunny



(a) original (b) 1000:1 (c) 100:1 (d) 20:1

Figure 7: Compression of the Spock