ELSEVIER

Computational Geometry 14 (1999) 167–186

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Single resolution compression of arbitrary triangular meshes with properties ☆

Chandrajit L. Bajaj *, Valerio Pascucci [1], Guozhong Zhuang [2]

Department of CS and TICAM, University of Texas, Austin, TX 78712, USA

Abstract

Triangular meshes are widely used as primary representation of surface models for networked gaming and for complex interactive design in manufacturing. Accurate triangulation of a surface with sharp features (highly varying curvatures, holes) may require an extremely large number of triangles. Fast transmission of such large triangle meshes is critical to many applications that interactively manipulate geometric models in remote networked environments. The need for a succinct representation is therefore not only to reduce static storage requirements, but also to consume less network bandwidth and thus reduce the transmission time.

In this paper we address the problem of defining a space efficient encoding scheme for both lossless and error-bounded lossy compression of triangular meshes that is robust enough to handle directly arbitrary sets of triangles including non-orientable meshes, non-manifold meshes and even non-mesh cases. The compression is achieved by capturing the redundant information in both the topology (connectivity) and geometry with possibly property attributes. Example models and results are also reported.

1. Introduction

Polygonal meshes have been used as the primary geometric model representation for complex interactive design in manufacturing or networked gaming. Accurate polygonal mesh approximation of a surface with sharp features (highly varying curvatures, holes) requires an extremely large number of triangles. Transmission of such large triangle meshes is critical to many applications that interactively manipulate geometry models in remote networked environments. The need for succinct representation is therefore not only to reduce static storage requirements, but also to consume less network band-width and thus reduce the transmission time. Although geometry compression and coding is an emerging discipline,

☆ This research is supported in part by grants from NSF-CCR-9732306, NSF-KDI-DMS-9873326, DOE-ASCI-BD-485, and NASA-NCC 2-5276.

* Corresponding author. E-mail: bajaj@cs.utexas.edu; http://www.ticam.utexas.edu/CCV

[1] E-mail: pascucci@cs.utexas.edu

[2] E-mail: zgz@cs.utexas.edu

0925-7721/99/$ – see front matter
PII: S0925-7721(99)00026-7

these techniques have matured for 2D digital images into standards such as JPEG [20] and MPEG [11]. In designing efficient geometry compression schemes, one attempts to take advantage of existing 2D image compression techniques.

1.1. Prior work

Deering [5] represents triangular mesh connectivity by generalized triangle strips. Stack operators are used in order to reuse vertices. In this way, the total number of random accesses to all vertices of the mesh is reduced. This method does not directly compress non-manifold meshes and its compression ratio is not high. Chow [3] presents an algorithm which represents a mesh by several generalized meshes. This method is optimized for real-time rendering but is not compression efficient because of the large requirement of connectivity encoding ($(\log n + 9)$ per vertex). Again, this method only considers manifold meshes.

In Topological Surgery [18], vertices are organized into a spanning tree and triangles into simple polygons which are further grouped into a series of triangle strips. The connectivity coding of this scheme is efficient, about 2–3 bits per triangle. One of its disadvantages is its inability to directly encode non-manifold meshes. As a preprocessing step, it splits a non-manifold object into several manifold components, thereby duplicating all non-manifold features: vertices, edges and faces. Touma and Gotsman present an algorithm for connectivity coding of orientable manifold meshes [19]. The efficiency of this lossless connectivity coding is determined by the distribution of the degrees of all vertices. Progressive transmission and embedded coding are discussed in [12–14]. Gumhold and Strasser propose a fast compression and decompression algorithms for real time applications [8]. A compact representation of multi-resolution surfaces that support progressive transmission is present in [1]. Rossignac introduced recently the Edgebreaker compression scheme [17] that is the first scheme proven to require at most $2n$ bits to encode the connectivity of a triangular mesh with n vertices. This result being valid for simple triangulations is extended with some additional storage to manifolds with multiple boundaries and handles. Another relevant encoding scheme, designed more for multi-resolution representation than for data compression, has been introduced by Popović and Hoppe [16]. This scheme, called Progressive Simplicial Complexes, is based on the edge contraction primitive and is very general in the sense that is capable to represent simplicial complexes of any dimension.

In this paper, we propose a new layering structure to partition an arbitrary triangular mesh (non-manifold, arbitrary-genus and possibly vertex irregularities) into generalized triangle strips. An efficient and flexible encoding of the connectivity, vertex coordinates and attribute data yields excellent single

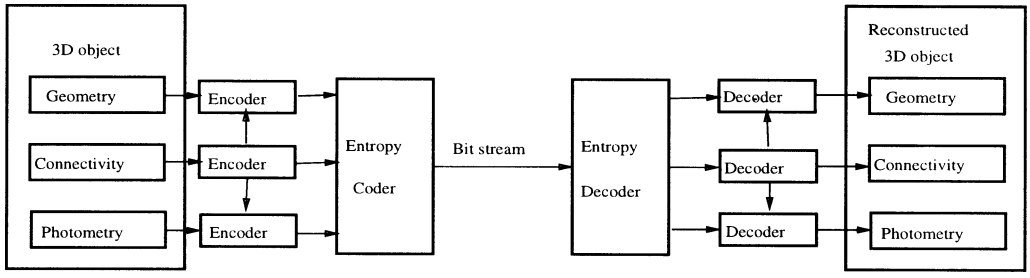


Fig. 1. Block diagram of the encoder and decoder.

resolution compression (an extended abstract has already appeared in [2]). This scheme gracefully solves the "crack" problem due to repeated vertices independently quantized and also prevents error propagation while providing efficient prediction coding for both geometry and photometry data such as colors, normals, and texture coordinates. Fig. 1 shows the diagram for both the encoder and decoder.

The rest of this paper is organized as follows. Section 2 introduces the layering scheme to partition the input data. Sections 3 and 4 address the coding of connectivity (topological information) while Section 5 provides details of the coding of geometric (numeric) data. Attribute (photometry data) coding is discussed in Section 6. Experimental results are presented in Section 7.

2. The layering scheme

Layering is used to encode the connectivity (topological) information of input triangle mesh collections. There are two basic kinds of layers: vertex layers and triangle layers. Assume that a triangle mesh has vertex set $\mathcal{V}$, face set $\mathcal{F}$, and edge set $\mathcal{E}$. A graph of the mesh is given by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with all vertices being the graph nodes and edges being the graph edges. There are no restrictions on the input triangle collection or this graph $\mathcal{G}$.

Definition 2.1 (Vertex layer). The 0th vertex layer is a randomly chosen vertex of the mesh. The kth vertex layer (with $k > 0$) includes a vertex v if v is not included in any previous vertex layer and there exists an edge $e = (v, v^*)$ where v^* is included in the $(k-1)$th vertex layer.

Definition 2.2 (Triangle layer). The kth triangle layer (with $k \geqslant 0$) includes a triangle t if t has at least one vertex in kth vertex layer and t is not included in any previous triangle layer.

Vertices and triangles categorized above have the following properties: any edge in $\mathcal{E}$ can only span two vertex layers; any triangle in $\mathcal{F}$ can only span two vertex layers; all the vertex layers form a partition of $\mathcal{V}$; all the triangle layers form a partition of $\mathcal{F}$. Based on the layering structure, a mesh edge is either

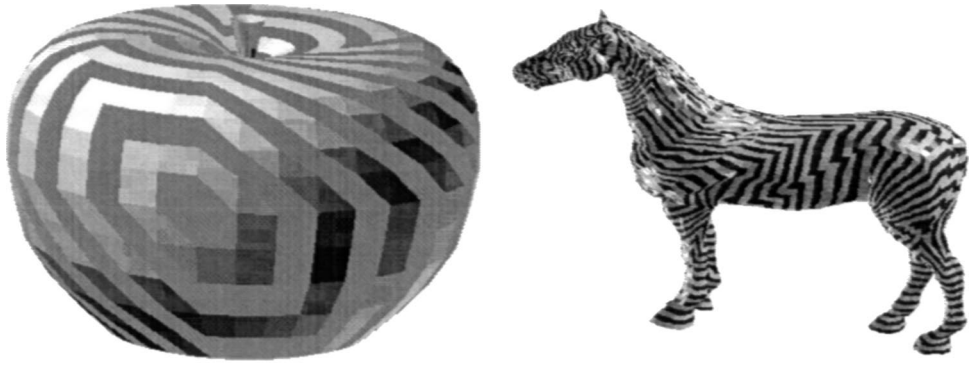


Fig. 2. The layering structure. Triangle layers are alternatively colored gray and yellow for both apple and horse.

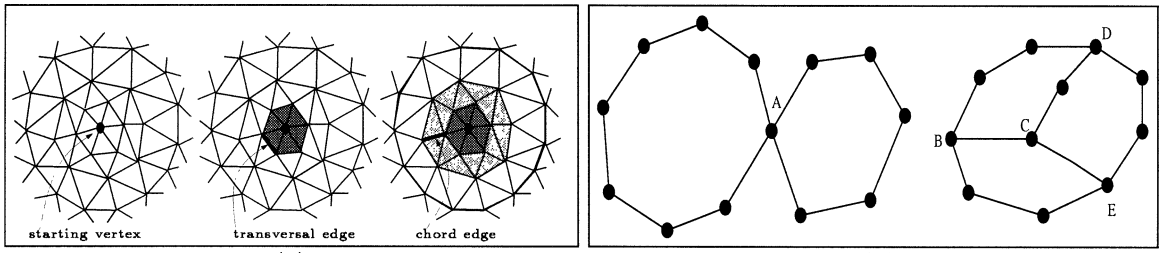


Fig. 3. Layering of a mesh with marked one *transversal edge* and one *chord edge*. (b) A vertex layer whose five branching points are marked with capital letters A–E.

a *chord* or a *transversal* (see Fig. 3). A chord is an edge that connects two vertices in different vertex layers while a transversal is an edge that connects two vertices in the same vertex layer.

2.1. Geometry primitives

The four geometry primitives we use to encode in our single-resolution compression method are: contours, branching points, triangle strips, and triangle fans.

Definition 2.3. A *contour* is an ordered sequence of vertices $\{v_0, v_1, \ldots, v_n\}$ in a vertex layer where each vertex pair (v_i, v_{i+1}) is connected by a transversal edge and every intermediate vertex v_i $(0 < i < n)$ is incident to exactly two transversal edges. Vertices v_0 and v_n can be the same point in which case the contour is called closed. A contour can be a single vertex which is also called an *isolated* point.

We call v_0 the starting vertex and v_n the ending vertex of the contour.

Definition 2.4. A *branching* point is a vertex in a vertex layer which is incident to more than two transversal edges (see Fig. 3(b)).

By this definition, a branching point is contained in more than one contour and thus it can only be either the starting or the ending vertex of a contour.

Definition 2.5. A *triangle strip* is an ordered sequence of triangles in a triangle layer where each pair of consecutive triangles share a common edge. The set of vertices of the triangles in the strip must belong to two contours in two separate vertex layers.

Definition 2.6. A *triangle fan* is an ordered sequence of triangles in a triangle layer where all triangles have a common vertex, each pair of consecutive triangles share a common edge, and no edge is shared by more than two triangles. It is possible that the first and last triangles share a common edge.

2.2. Layering procedure

Given a triangle mesh, vertex layers are formed by using a breadth-first traversal algorithm [4]. According to the definition, triangle layers are formed during the same traversal.

For each vertex layer, we construct the contours according to the set E of transversal edges with all their extreme points located in this vertex layer. To form a contour, first pick up a starting edge with the following rules: (1) if there is an edge in E with one of its extreme points being a branch point; (2) otherwise, arbitrarily choose one in E. In the first case, we define the branching point as the starting vertex of the contour and the other as the current vertex. In the second case, the choice can be arbitrary, in which case the starting edge is removed from E. The second edge of the contour is in E with the current vertex being one of its two extreme points. Remove the newly found edge and update the current vertex. This procedure stops if one of the following occurs: (a) there are no more edges with the current vertex as one extreme point; (b) the current vertex is the same as the starting vertex (thus one closed contour is constructed); (c) the new edge has one branching extreme point. Following the above procedure, contours are repeatedly constructed until the set E is empty. Every vertex that does not lie in any contour will form a contour of a single vertex (isolated vertex).

For each triangle layer, triangle strips are constructed according to the sequences of chords incident to each contour. In particular a strip is formed by repeatedly gluing two triangles that share a chord. The constraint is that each of the boundaries of a strip is located within a single contour. Triangles with all their vertices in the same vertex layer are attached to strips as bubbles. This can improve the actual coding efficiency. Triangle fans are formed from the remaining triangles (those that are not included in any strip).

Claim 2.1. *The layering procedure decomposes an arbitrary mesh into the four geometric primitives*: (i) *contours,* (ii) *branching points,* (iii) *triangle strips and* (iv) *triangle fans.*

Claim 2.2. *Triangle meshes of any topological type can be represented by the layering scheme outlined above* (*see Section 4*).

This layered representation avoids the “crack” problem which occurs when a non-manifold mesh is converted into several manifold components and non-manifold features (vertices, edge, faces) are duplicated [7].

Vertex indexing gives each vertex a reference which is used in connectivity encoding. Vertex indexing is directly related to coding efficiency. Three kinds of vertex indices will be used in our scheme: local indices, global indices and relative indices. The first two are defined below while relative indexing will be explained in the next section.

2.3. Local indexing

Local indexing is only meaningful to individual vertex layers. Every vertex in a vertex layer is assigned a unique local index. If n is the total number of vertices in a vertex layer, then local indices of this vertex layer are $0, 1, \ldots, n-1$. The numbering order of local indices in a vertex layer is as follows: (1) all branching vertices are numbered first; (2) for each contour, all non-branching vertices are numbered, from the starting vertex. It is important that all branching vertices be numbered separately because they will be referenced by more than one contour.

2.4. Global indexing

The global index of a vertex is defined as the summation of its local index value and the total number of vertices in all previous vertex layers. Indexing starts from the 0th vertex layer. Suppose that there are totally V vertices in the mesh, then the smallest global index is 0 and the biggest one is $V-1$. The incidence of the reconstructed mesh is expressed with global indices. Let $\mathrm{BC}(m)$ be the minimal number of bits to correctly code a non-negative integer smaller than m. Obviously, $\mathrm{BC}(m) := \min\{k \mid 2^k > m\}$. Suppose that L is the number of vertices in a vertex layer, then the local index of each vertex in that layer can be coded by $\mathrm{BC}(L)$ bits. However, any global index needs $\mathrm{BC}(V)$ with a total number of V vertices in the mesh. Since L is generally much smaller than V, $\mathrm{BC}(L) < \mathrm{BC}(V)$, local index coding saves $\mathrm{BC}(V) - \mathrm{BC}(L)$ bits. A local index must be converted back to the global index in the decoding process. A global index g and a local index l of a vertex v have the relation $g = l + \sum_{i=0}^{k-1} L_i$ where L_i is the total number of vertices in the ith vertex layer and layer k is where v is located.

3. Connectivity coding

Connectivity encoding is the central part of any 3D surface compression method. It guides both the geometry and photometry coding. Single-resolution compression does not change the connectivity in the sense that the decoder can perfectly recover the connectivity, modulo a permutation of the vertices and triangles.

3.1. Outline of the scheme

Our connectivity scheme is based on the layering structure. The entire connectivity encoding procedure is: (1) encode the total number of layers; (2) encode the layout of each vertex layer; and (3) encode all triangle strips and fans in each triangle layer.

The layout of a vertex layer is specified by the number of branching points; the number of contours; for each contour, the number of vertices, and the characteristics of the starting and ending vertices (one bit each to indicate if it is a branching vertex; a branching vertex has its local index coded). A triangle strip has two boundary contours. The one in the previous vertex layer is called the *parent contour*, the other is called the *child contour*.

3.2. The coding of bubbles

To improve the compression efficiency we encode in a single primitive, called *exceptional triangle strip*, a set of triangles that would be turned into an actual triangle strip (under Definition 2.5) if few *bubble triangles* were removed. A *bubble triangle* is just a triangle with all its vertices in the same vertex layer (see Fig. 4). Note that an exceptional triangle strip can have bubble triangles attached only to its parent contour. To encode an exceptional triangle strip, the following information must be coded: local indices of the two starting vertices, the bit march string, the number of bubbles, and the bubbles themselves. The bit march string is further encoded by an entropy coding algorithm (Huffman coding [10] or arithmetic coding [15]). Fig. 4 shows two exceptional strips with attached bubbles. To encode a bubble on a triangle strip, the following information is coded: location of the starting vertex involved; vertex span

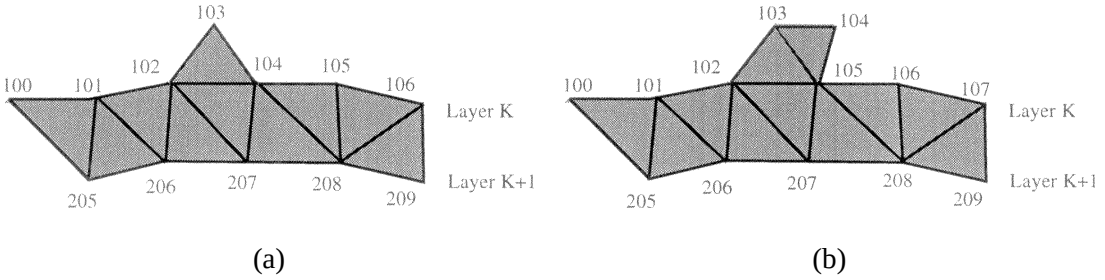


Fig. 4. Exceptional triangle strips where local indices are listed: (a) one bubble triangle $(102, 104, 103)$, (b) two bubble triangles $(102, 105, 103)$ and $(103, 105, 104)$.

which is the total number of involved vertices; number of triangles in the bubble; triangle index triples. Bubble coding is expensive, however, not all the above values need to be put into the bit stream. We use relative indexing to reduce the overall number of coding bits. A relative index is defined with respect to a bubble. Suppose that a bubble spans n vertices and the local index of the starting vertex of the bubble is X. Then the relative indices of all involved vertices are defined as $0, 1, \ldots, n-1$. A triangle in a bubble is encoded by a triple of relative indices. For instance, since the relative indices of 102, 103, 104, 105 in Fig. 4(b) are 0, 1, 2, 3, the two bubble triangles are encoded as $(0, 3, 1)$ and $(1, 3, 2)$. In this case, coding a bubble triangle only costs 6 bits.

Experiments show that most bubbles appear in the form of a one-triangle bubble with a vertex span of 3, and relative index triple fixed as $(0, 2, 1)$. The efficiency of the bubble encoding can be improved by using one bit to indicate if a bubble has only one triangle. No more information is needed if it does.

3.3. The coding of bit march strings

Once the two starting vertex positions and possible bubbles are coded, the remaining problem of coding an exceptional strip is reduced to coding exceptional triangle strips. A simple and direct way to encode a bit march string is to put the 0s and 1s directly into the bit stream. However, this is not efficient. Symbols 0101 and 1010 are found to appear more frequently than other symbols such as 1111 or 0000 in a typical bit march string if every four consecutive bits are grouped into a symbol.

A static Huffman table is designed for all the sixteen symbols. Construction of this table is based on the symbol occurrences over a number of large models. In case the length of a bit march string is not a multiple of 4, its remaining bits are simply coded one by one. Entropy coding of bit march strings uses about 25% less space than direct coding.

3.4. The coding of triangle fans

Triangles that do not belong to any strip are special. Their vertices usually span three contours or two contours in the same vertex layer. A triangle fan can be expressed by a sequence of vertices $\{v_0, v_1, \ldots, v_m\}$ where v_0 is the common vertex and (v_0, v_i, v_{i+1}) $(i = 1, \ldots, m-1)$ is a triangle. To code a triangle fan, the following information is put into the bit stream: the number $m-1$ of triangles in the fan; a sequence of local indices of $m+1$ vertices with the first one as the center (the commonly shared vertex) of the fan.

Table 1
Percentages of triangles coded in bubbles and fans. The fifth column (CC) is the total connectivity coding cost in bytes. The sixth column (CC/t) is the connectivity coding cost per triangle in bits.

Model	Triangles	Bubble	Fan	CC	CC/t
Crocodile	34,404	626 (1.8%)	1,036 (3.0%)	13,055	3.04
Teapot	144,384	157 (0.1%)	532 (0.3%)	12,644	0.70
Honda	13,594	96 (0.7%)	290 (2.1%)	3,768	2.22
Phone	165,963	638 (0.4%)	476 (0.3%)	19,284	0.93
Engine	584,793	7,017 (1.2%)	14,619 (2.5%)	713,809	2.69

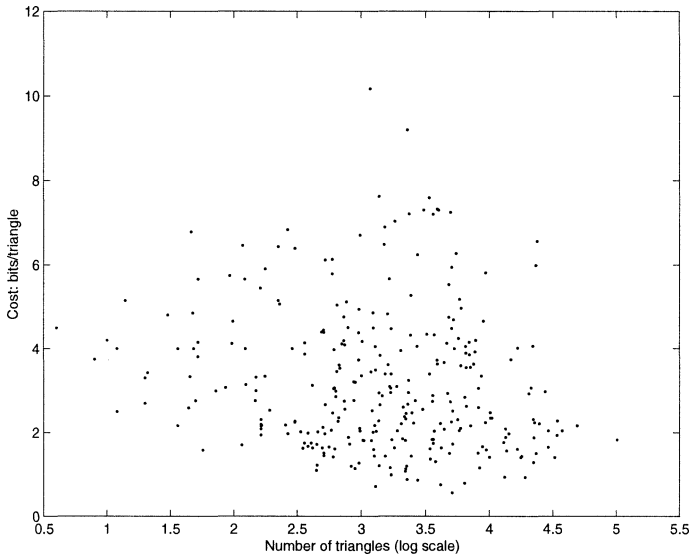


Fig. 5. Cost of the connectivity coding.

For larger models, the connectivity cost of our scheme can be as good as less than one bit per triangle. Experiments show that connectivity coding cost is on average 3 bits per triangle for common objects (see Fig. 5).

4. Representation power

In this section we qualify our *Layering Scheme* from the point of view of its representation power. In particular we prove the claim that this scheme can represent any set of triangles with no constraint on their topology. Consider a set of triangles represented by an array of n vertices and an array of m triangles, where each vertex is a triple of (x, y, z) coordinates and each triangle is a triple (i, j, k) of vertex indices in the range $[0, n-1]$ as in the following table:

Vertices			
V_1	x_1	y_1	z_1
V_2	x_2	y_2	z_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_n	x_n	y_n	z_n

Triangles			
T_1	V_{i_1}	V_{j_1}	V_{k_1}
T_2	V_{i_2}	V_{j_2}	V_{k_2}
T_3	V_{i_3}	V_{j_3}	V_{k_3}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_m	V_{i_m}	V_{j_m}	V_{k_m}

We call this representation `IndexedTriangleSet`. In `VRML` terminology this is a valid `IndexedFaceSet` restricted to triangles (an invalid `IndexedFaceSet` would have some of the (i, j, k) indices out of the range $[0, n-1]$). It easy to see that using this scheme one can represent any set of triangles with no restriction on the topology of the collection. We show that the layering scheme has the same representation power. Note that to simplify the proof we demonstrate our claim without using any of the efficiency consideration discussed in the previous sections. The point we are making here is that the Layering Scheme is the first compression approach (and at this moment the only one) that is robust enough to encode any `IndexedTriangleSet`. Even a very general scheme like the progressive simplicial complexes [16] cannot deal directly with non-mesh cases like the example in Fig. 6(d) that we can encode. The reason is that the progressive simplicial complexes are based on the edge contraction primitive. But you cannot perform a valid edge contraction between the large top triangle and one of the saw teeth because the set of triangles in Fig. 6(d) does not form a valid simplicial complex. This example shows in fact a set of triangles that do not form a complex and hence some preprocessing that introduces some additional "phantom" edges (edges encoded and marked as non-existing) would be needed.

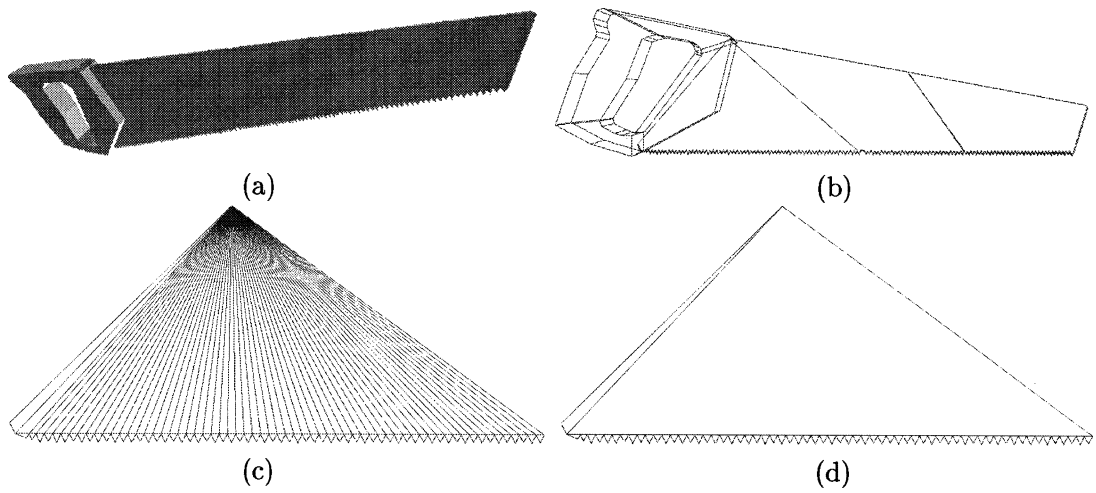


Fig. 6. (a) The vrml model of a hand saw. (b) A wireframe view that shows the decomposition of the model into polygonal pieces. (c) One triangulation of the leftmost piece of the blade. (d) An alternative triangulation of the same piece.

Theorem 4.1. *Any* `IndexedTriangleSet` *can be encoded in the Layering Scheme as a sequence of vertex layers and triangle layers. The* `IndexedTriangleSet` *is recovered exactly modulo*: (i) *renumbering of the vertex/triangle indices and* (ii) *loss of the vertices not used in any triangle.*

Proof. Consider an input `IndexedTriangleSet` $\mathcal{I}$. We build a graph $\mathcal{G}$ by adding in sequence the edges of the triangles T_i in $\mathcal{I}$ as follows.

For each $T_i = \{V_{i_1}, V_{j_1}, V_{k_1}\} \in \mathcal{I}$:

- if $V_{i_1} \notin \mathcal{G}$: create in $\mathcal{G}$ a new node V_{i_1},
- if $V_{j_1} \notin \mathcal{G}$: create in $\mathcal{G}$ a new node V_{j_1},
- if $V_{k_1} \notin \mathcal{G}$: create in $\mathcal{G}$ a new node V_{k_1},
- if $edge(V_{i_1}, V_{j_1}) \notin \mathcal{G}$: create in $\mathcal{G}$ a new $edge(V_{i_1}, V_{j_1})$,
- if $edge(V_{i_1}, V_{k_1}) \notin \mathcal{G}$: create in $\mathcal{G}$ a new $edge(V_{i_1}, V_{k_1})$,
- if $edge(V_{k_1}, V_{j_1}) \notin \mathcal{G}$: create in $\mathcal{G}$ a new $edge(V_{k_1}, V_{j_1})$.

Performing a `BFT` (Breadth First Traversal) of $\mathcal{G}$ starting from any random point we traverse all the vertices of a connected component $\mathcal{G}_i$ of $\mathcal{G}$. If $\mathcal{G}_i$ is a single vertex then it is discarded. We repeat the Breadth First Traversal of $\mathcal{G}$ starting from an untraversed node until all its nodes are traversed. In this way $\mathcal{G}$ is decomposed into k connected components $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k\}$. Each component $\mathcal{G}_i$ is encodable separately. Note that by construction each vertex and triangle appears in only one connected component of $\mathcal{G}$ so that encoding each $\mathcal{G}_i$ separately does not imply the use of redundant information which might produce inconsistencies such as encoding the same vertex in two different positions.

The vertices in $\mathcal{G}_i$ are grouped in Vertex Layers on the basis of the `BFT` used to build $\mathcal{G}_i$. The coordinates of each vertex are encoded once in the order induced by the `BFT`. This means that each vertex of $\mathcal{I}$ that appears in $\mathcal{G}_i$ is encoded exactly once independent from the number of times it is referenced in the set of triangles.

The triangles with vertices in $\mathcal{G}_i$ are then encoded one by one as triangle fans of one triangle each. Note that this can be done because no adjacency information is required to be explicitly encoded. In this way we can encode the entire set of triangle in the input `IndexedTriangleSet`. To achieve better space efficiency one tries to use strips and longer fans as much as possible, but if any additional triangle remains to be represented one can always encode it as a fan of length one. □

The consequence of this proof is that the Layering Scheme is the first *robust* compression method for triangular meshes. It can be used to encode any set of triangles without need for pre-processing that alters the input data to enforce some special properties.

Non-manifold meshes like those in Fig. 11 are encoded directly and reliably without any modification. In several cases this gives also the opportunity to triangulate an object with many fewer triangles. Consider, for example, the model of a saw in Fig. 6(a,b) whose blade is decomposed into several pieces and, furthermore, each single piece is triangulated to yield a triangular mesh. Fig. 6(c) shows one possible consistent triangulation of a piece of the blade. Since every pair of adjacent triangles have the same common edge the number of triangles is twice the number of teeth (plus one). Fig. 6(d) shows an alternative triangulation of the same piece of blade with number of triangles equal to the number of teeth (plus two). This is not often considered as a consistent mesh, and hence usually discarded as "invalid input", since one edge of the large triangle is partially adjacent to many small triangles (the teeth of the blade). However, using our flexible Layering Scheme one can directly encode this collection of triangles with a near 50% savings in encoding cost.

5. Geometry coding

With the encoded connectivity information, the overall geometry encoding scheme is straightforward. For each vertex layer, positions of all branching vertices are encoded directly in the order they are locally indexed. For each contour, positions of all its non-branching vertices are predictively encoded.

Our geometry coding pipeline involves bounding box coding, prediction, quantization, and entropy coding. The bounding box of an input mesh is specified by the maximum and minimum values of all the x, y, z coordinates: Xmin, Xmax, Ymin, Ymax, Zmin and Zmax. Using these values, all vertex positions can be normalized so that they are located in a unit cube.

5.1. Predictive geometry coding

Predictive coding has been extensively utilized in 2D image compression methods. For instance, the JPEG standard [20] provides predictive lossless coding mode for still image compression. The predictive technique can also be used to remove redundant information within the geometry and attributes of a triangular mesh. A vertex position predictor combines the positions of previously encoded neighboring vertices to form a prediction of the current vertex position. There are three commonly used geometry predictors: linear predictors [5], high-order predictors [18], and parallelogram predictors [13]. A correction, more suitable for efficient coding, is defined as the difference between the actual vertex position and its predicted position. Either Cartesian or spherical coordinates can be used to express a position or a correction which is quantized into an integer code and then entropy coded.

To quantize a position or a correction vector, we first compute its spherical coordinates (r, ϕ, θ). Then the three components are vector quantized to a choice of vectors chosen uniformly in a solid sphere. With the bounding box and other prenormalization, r is in the interval $[0, 1]$, and ϕ and θ are in $[0, 2\pi]$ and $[0, \pi]$, respectively. Note that because of the difference in nature and length of the three polar coordinates (r, ϕ, θ) they are usually quantized differently resulting in a number of bits per vertex which is not necessarily multiple of three.

5.2. Codebook design

Without loss of generality, suppose that the sample space is

$$\boldsymbol{S} = \{\boldsymbol{x} \mid \|\boldsymbol{x}\| \leqslant 1,\ \boldsymbol{x} \in \mathbb{R}^3\}.$$

Let the order set $C = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N\}$ be the codebook to be designed. N is the codebook size, which is adjustable according to the user-specified error tolerance. $Y_i,\ i = 1, 2, \ldots, N$, are the code vectors.

A vector quantizer Q of dimension three and size N is a mapping from a vector in $\boldsymbol{S}$ to a code vector in C. Basically, every N-point vector quantizer has an associated partition of $\boldsymbol{S}$ into N regions or cells, S_i $(i = 1, 2, \ldots, N)$, where the ith cell is defined by

$$S_i = \{\boldsymbol{x} \in S:\ Q(\boldsymbol{x}) = \boldsymbol{y}_i\}.$$

The *code rate* of the vector quantizer Q is $r = (\log_2 N)/3$ which measures the number of bits that a vector component uses to express the input vector. The code rate shows the accuracy of a vector quantizer and depends on the goodness of the chosen codebook. Clearly, a large-size codebook gives good accuracy but results in a large bit rate and, thus, poor compression performance.

We present here a productive vector quantizer defined by of three sub-codebooks. A vector (x, y, z) in S can be expressed in the sphere coordinates (r, ϕ, θ) by the relation

$$\begin{aligned} r &= \sqrt{x^2+y^2+z^2}, && 0 \leqslant r \leqslant 1, \\ \phi &= \arctan\left(\frac{y}{x}\right), && 0 \leqslant \phi \leqslant 2\pi, \\ \theta &= \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right), && 0 \leqslant \theta \leqslant \pi. \end{aligned}$$

The three scalar r, ϕ and θ are encoded separately by using the following three scalar sub-codebooks: C_r, C_ϕ and C_θ which are of size $N_r = 2^{K_r}$, $N_\phi = 2^{K_\phi}$ and $N_\theta = 2^{K_\theta}$, respectively.

$$\begin{aligned} C_r &= \{r_i \mid r_i = (i-1)/N_r,\ i = 1, 2, \ldots, N_r\}, \\ C_\phi &= \{\phi_j \mid \phi_j = 2\pi(j-1)/N_\phi,\ j = 1, 2, \ldots, N_\phi\}, \\ C_\theta &= \{\theta_k \mid \theta_k = \pi(k-1)/N_\theta,\ k = 1, 2, \ldots, N_\theta\}. \end{aligned}$$

Hence a vertex will be quantized by $K_r + K_\phi + K_\theta$ bits. And the overall codebook is

$$C = \{\boldsymbol{p}_{ijk} := (x_{ijk}, y_{ijk}, z_{ijk})^{\mathrm{T}} \mid i = 1, 2, \ldots, N_r;\ j = 1, 2, \ldots, N_\phi;\ k = 1, 2, \ldots, N_\theta\},$$

with $x_{ijk} = r_i \sin\theta_k \cos\phi_j$, $y_{ijk} = r_i \sin\theta_k \sin\phi_j$, $z_{ijk} = r_i \cos\theta_k$.

5.3. Nearest neighbor search

The Euclidean distance will be used to measure distortion

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|.$$

For a given vector $\boldsymbol{x} \in S$, the task of nearest neighbor search is to search for its closest code vector in the sense that it has minimal distortion $d(\boldsymbol{x}, \boldsymbol{y}_j)$ over all the code vectors in the codebook C. Naive direct search is of $O(N)$ time complexity. Well-designed codebook [6] can be of $O(\log N)$ time complexity. For our codebook given above, only constant time is required for a one time nearest neighbor search. For any vector $\boldsymbol{x} \in S$, its sphere coordinate (r, ϕ, θ) can first be computed in constant time. Then three intervals $[r_i, r_{i+1}]$, $[\phi_j, \phi_{j+1}]$ and $[\theta_k, \theta_{k+1}]$ are found, within which r, ϕ and θ are located respectively. Distortions between $\boldsymbol{x}$ and each of the eight code vectors $\{\boldsymbol{p}_{i'j'k'},\ i' = i, i+1;\ j' = j, j+1;\ k' = k, k+1\}$ are computed. Finally, since we know that the distortion must be equal to one of eight distortions, the code vector can be easily decided by picking out the one which has the minimal distortion. All this can be done in constant time. We let $\mathrm{PVQ}(\boldsymbol{x})$ be the code vector of $\boldsymbol{x}$ and $D_C(\boldsymbol{x}) = d(\boldsymbol{x}, \mathrm{PVQ}(\boldsymbol{x}))$ be the distortion.

5.4. Distortion upper bound

Given a user specified error tolerance ϵ, one must decide the sizes, N_r, N_ϕ and N_θ of the three sub-codebooks. These three parameters should be chosen as small as possible so as to attain a small bit rate. The following quantitative analysis shows how the error control is derived.

For any vector $\boldsymbol{x} \in S$, its associated sphere coordinates (r, ϕ, θ) must be located in some cube $[r_i, r_{i+1}) \times [\phi_j, \phi_{j+1}) \times [\theta_k, \theta_{k+1})$, so the distortion will not exceed half the length of the diagonal of

the corresponding Cartesian cube within the unit sphere. Let $\boldsymbol{p}_0 = (r_i, \phi_j, \theta_k)$ and $\boldsymbol{p}_1 = (r_{i+1}, \phi_{j+1}, \theta_{k+1})$ be two endpoints of a diagonal. Suppose that (x_0, y_0, z_0) be (x_1, y_1, z_1) are their Cartesian coordinates. Since

$$\begin{aligned}
|x_1 - x_0| &= \left(r_i + \frac{1}{N_r}\right) \sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\left(\phi_j + \frac{2\pi}{N_\phi}\right) - r_i \sin\theta_k \cos\phi_j \\
&= r_i \Bigg(\sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\left(\phi_j + \frac{2\pi}{N_\phi}\right) - \sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\phi_j \\
&\quad + \sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\phi_j - \sin\theta_k \cos\phi_j \Bigg) + \frac{1}{N_r} \sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\left(\phi_j + \frac{2\pi}{N_\phi}\right) \\
&= r_i \Bigg(\sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \Bigg(-2 \sin\left(\phi_j + \frac{\pi}{N_\phi}\right) \sin\left(\frac{\pi}{N_\phi}\right) \Bigg) \\
&\quad + 2\cos\left(\theta_k + \frac{\pi}{2N_\theta}\right) \sin\left(\frac{\pi}{2N_\theta}\right) \Bigg) + \frac{1}{N_r} \sin\left(\theta_k + \frac{\pi}{N_\theta}\right) \cos\left(\phi_j + \frac{2\pi}{N_\phi}\right),
\end{aligned}$$

we have

$$|x_1 - x_0| \leqslant r_i \left(\frac{2\pi}{N_\phi} + \frac{\pi}{N_\theta}\right) + \frac{1}{N_r} \leqslant \frac{2\pi}{N_\phi} + \frac{\pi}{N_\theta} + \frac{1}{N_r}.$$

Similarly,

$$|y_1 - y_0| \leqslant \frac{2\pi}{N_\phi} + \frac{\pi}{N_\theta} + \frac{1}{N_r}, \qquad |z_1 - z_0| \leqslant \frac{\pi}{N_\theta} + \frac{1}{N_r}.$$

Claim 5.1. *If N_r, N_ϕ and N_θ satisfy*

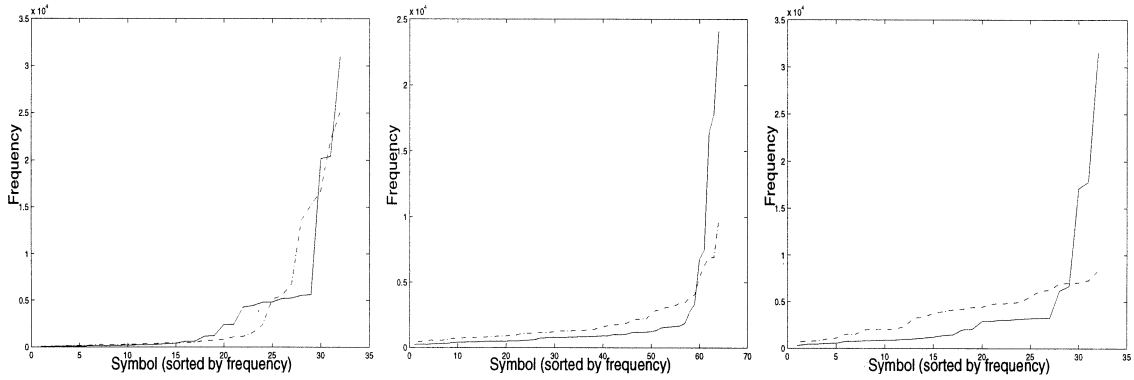
$$\frac{1}{N_r} + \frac{2\pi}{N_\phi} + \frac{\pi}{N_\theta} < \frac{2}{\sqrt{3}}\varepsilon,$$

then for any $\boldsymbol{x} \in S$, $D_C(x) < \varepsilon$.

5.5. Geometric encoding with error propagation prevention

Since the layering structure groups every vertex layer by a set of contours, one needs only to consider the geometry encoding of each of these contours. We use an efficient geometric coding algorithm which controls geometry error by confining the distortion of productive vector quantization. At the same time, error propagation is prevented.

Let $P_0, P_1, \ldots, P_n$ be the vertices which form a contour in that order. Define $\Delta P_i = P_{i+1} - P_i$ for $i = 0, \ldots, n-1$. Using direct prediction, a straight way to encode all the vertex coordinates would be (1) encode P_0 directly; (2) encode ΔP_i, $i = 0, \ldots, n-1$, by the productive vector quantization. But there are two disadvantages with this scheme. On one hand, the error from the productive vector quantization can be accumulated. On the other hand, redundancy still exists in this raw encoding scheme. The accumulating error problem can be solved by taking an on-the-fly computing skill so that any vertex after being recovered by the decoder will not deviate its original position by an Euclidean distance more than the maximum distortion of the vector quantization. To remove the second disadvantage, we design a intuitive and efficient scheme which is based on statistics of a general case behavior of correction vectors ΔP_i in ordinary contours of the layering structure.



(a) r (b) ϕ (c) θ

Fig. 7. Frequency curves for two encoding schemes: solid curves for code difference and dash lines for the direct encoding. The symbols are sorted by frequency for comparison. These frequency curves show that in the code difference approach most codes are accumulated around only a few symbols.

5.6. Second-order code prediction

Direct prediction removes redundancy by identifying similar bit values between coordinates of adjacent vertices. However, it is not optimal, especially for models without many sharp features.

Suppose ΔP_k and ΔP_{k+1} are two adjacent correction vectors and the two integer triples (r_k, ϕ_k, θ_k) and $(r_{k+1}, \phi_{k+1}, \theta_{k+1})$ are their corresponding codes. We observe that since the angular change from ΔP_k to ΔP_{k+1} is generally small, $\Delta\phi_k = \phi_{k+1} - \phi_k$ and $\Delta\theta_k = \theta_{k+1} - \theta_k$ are usually small integers.

It might be helpful to stress that this second order prediction is not the same as coding difference of differences. Our prediction has two phases. The first phase computes differences and then quantizes them into integer codes. The second phase computes the difference of quantized difference codes. Experimental results show that encoding correction code differences is more efficient than directly encoding correction codes. Fig. 7 shows the frequencies of symbols of different sizes where horizontal values are symbol values and vertical values are their corresponding frequencies. In this figure, solid curves show the frequency of correction code differences while dash lines show the frequency of correction codes from the direct encoding scheme. The Huffman coding method [10] is also used to encode code differences.

6. Attribute coding

Besides vertex positions, a mesh may have attached attributes such normals, colors, and texture coordinates which are used for enhancing shading effect. The way to bind these attributes with a triangular mesh in VRML [9] can be either per vertex, or per face, or per corner.

The position predictors can be generalized to code normals, colors, and texture coordinates. In this paper, we present our prediction schemes for normal coding with the “per vertex” binding. RGB color information and (u, v) texture coordinates are handled as vectors similarly to normals.

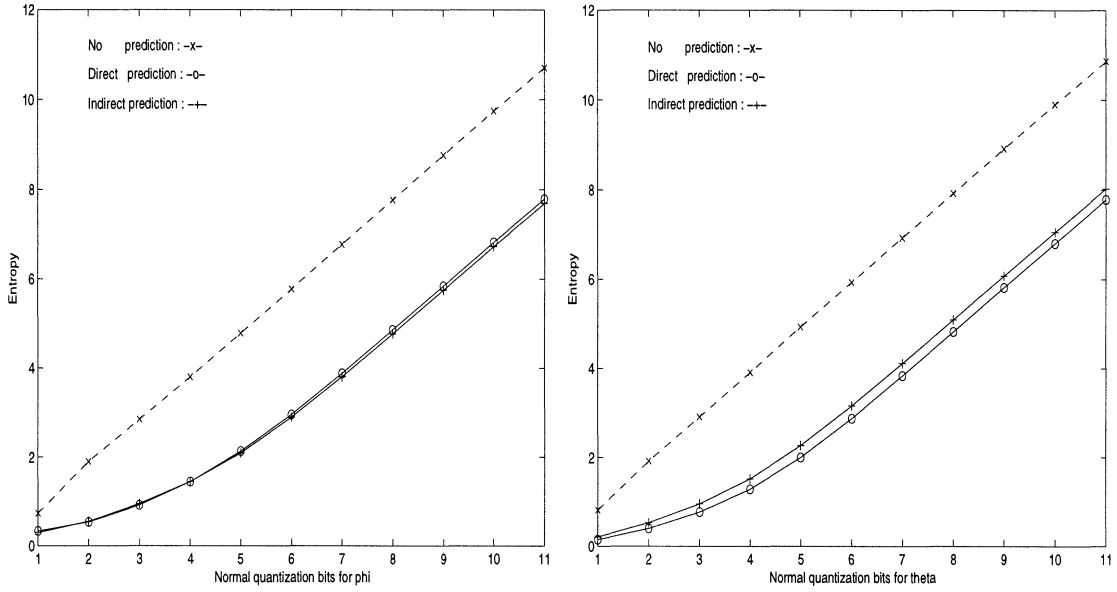


Fig. 8. Comparison of geometry coding: no prediction, indirect prediction, and direct prediction.

Just like coding a vertex position, one can use a linear combination of encoded normals of neighboring entities (vertices or faces) to predict a normal. This is direct prediction. However, the normal associated with each entity usually reflects local geometry variation around the entity. This implies that the normal can be predicted from neighboring geometry which is supposed to be already recovered by the decoder. We call this approach indirect prediction. Several methods to estimate vertex normals can be used in an indirect prediction approach: average of normals of incident faces, quadratic surface fitting, and subdivision. Since indirect predictors are based on the geometry information of the reconstructed mesh, the encoder must use the same geometry information when it does normal prediction. The first predictor averages normals of all incident faces and uses the average normal vector as the prediction vector. The second predictor uses geometry information of neighboring vertices to construct a quadratic fitting surface and uses the normal of the fitting surface at the predicated vertex as the prediction. The subdivision predictor is similar to the first predictor but the incident faces are changed. Fig. 8 compares the efficiency of normal coding of three different methods: no prediction, indirect prediction by averaging, and direct prediction from the normal of a neighboring vertex. Clearly, coding normals without prediction is least efficient. For the other two cases, the coding results are largely dependent on the input data.

For any unit normal (n_0, n_1, n_2), its corresponding sphere coordinates can be written as $(1, \phi, \theta)$. So direct coding (no prediction) only needs to code the ϕ and θ components. Any correction vector of a normal does not need to be of unit length. However, since both the prediction normal and the true normal are of unit length, it is sufficient to code the ϕ and θ components of the correction vector. Fig. 9 shows two test cases of compressed triangulated surfaces with normal and color properties. The corresponding coding cost is reported under each model.

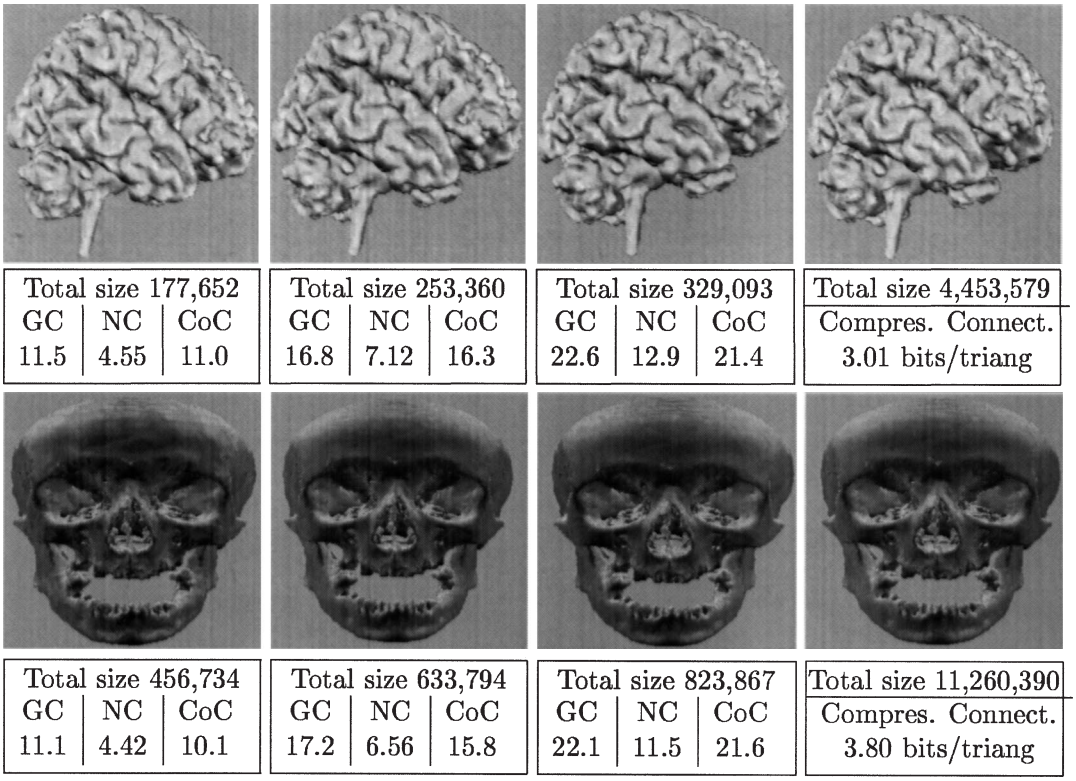


Fig. 9. Compression of normals and colors. Both models (brain and skull), whose originals are shown in the last column (GZIP sizes: 981,827 and 2,431,665 bytes), have normal and color properties. For the first three columns, geometry quantization bits are 12, 18 and 24 per vertex; normal quantization bits are 8, 12, 16 per vertex; color quantization bits are 12, 18 and 24 per vertex. Total storage sizes (in bytes) are reported under each model. Under each model is also reported in bits per vertex the costs of the Geometric Coding (GC), the cost of the Normal Coding (NC) and the Color Coding (CoC). The Connectivity Coding in bits per triangle (equal for all three compressed models) is reported in the last column.

7. Experimental results

Table 2 compares the compression performance of our method with GZIP compression of the VRML ASCII models shown in Fig. 10. The geometric error $E(A, B)$ is reported in parentheses in the last column. Consider a model A and its reconstruction B after lossy compression. The geometric error $E(A, B)$ between A and B is defined as

$$E(A, B) = \frac{1}{n} \sum_{i=0}^{n-1} \min_{0 \leqslant j < n} \|P_i - Q_j\|_2,$$

where P_i $(0 \leqslant i < n)$ and Q_j $(0 \leqslant j < n)$ are the vertex positions of A and B, respectively, and A is normalized such that its bounding box diameter is 100.

Fig. 12 compares the compression results of our method with the compressed gzip files for a standard set of 300 VRML models [9]. Each point in the plots represents one model of the set. The x coordinate

Table 2
Comparison of compressed file sizes in bytes. The fourth column (CC) is the connectivity coding cost. The last column is the geometry coding (GC) for a quantization of 25 bits per vertex (bpv, or total bits for three coordinates). The numbers in parenthesis are geometry errors.

Model	Original	GZIP	CC	GC (25 bpv)
Crocodile	1,212,767	395,849	13,055	63,127 (0.010)
Teapot	5,471,965	1,459,284	12,644	150,900 (0.007)
Honda	397,730	118,377	3,768	25,739 (0.012)
Phone	8,558,332	2,138,042	19,284	240,599 (0.008)
Engine	15,888,473	4,357,764	196,694	815,715 (0.003)

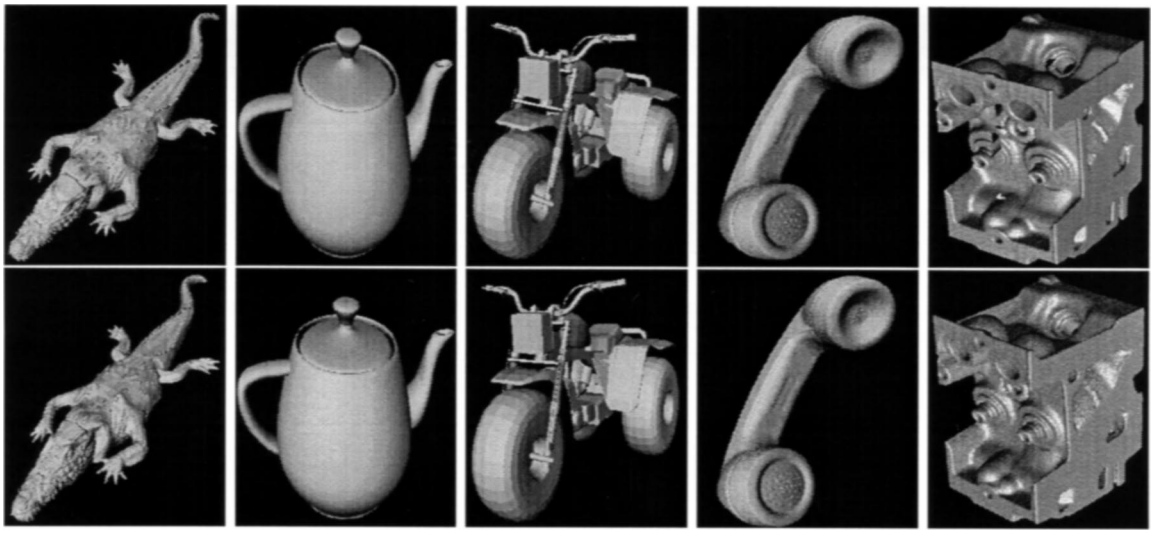


Fig. 10. Compression of triangle meshes. The first row shows the original models. The second row shows the reconstruction after lossy compression using 25 bits per vertex (total for all three coordinates).

of the point is the number of vertices of the corresponding model. The y coordinate is the compression ratio obtained for that model. For the left plot the compression ratio is the size of the original VRML ASCII model over the size of our compressed file. For the right plot the compression ratio is the size of the GZIP compressed VRML file over the size of our compressed file.

Table 3 reports the compression results for the two non-manifold models shown in Fig. 11 with 40 (top row) and 260 (bottom row) non-manifold vertices.

Table 3
Compression results: V is the number of vertices; S_1 is the file size of an uncompressed model in bytes; S_2 is the file size of a compressed model in bytes. Savings are the compression results of non-manifolds over manifold. The quantization bits are 15 per vertex in all cases. The Connectivity Coding (CC) costs are reported in bits per triangle while the Geometric Coding (GC) costs are reported in bits per vertex.

	Manifold					Non-manifold					
Object	V	S_1	S_2	CC	GC	V	S_1	S_2	CC	GC	Savings
Saturn	810	47,303	2,581	2.95	14.6	770	45,728	2,130	2.86	13.2	9.7%
Brain	34,676	2,422,390	124,469	3.03	14.4	34,416	2,385,138	123,096	3.01	14.1	1.1%

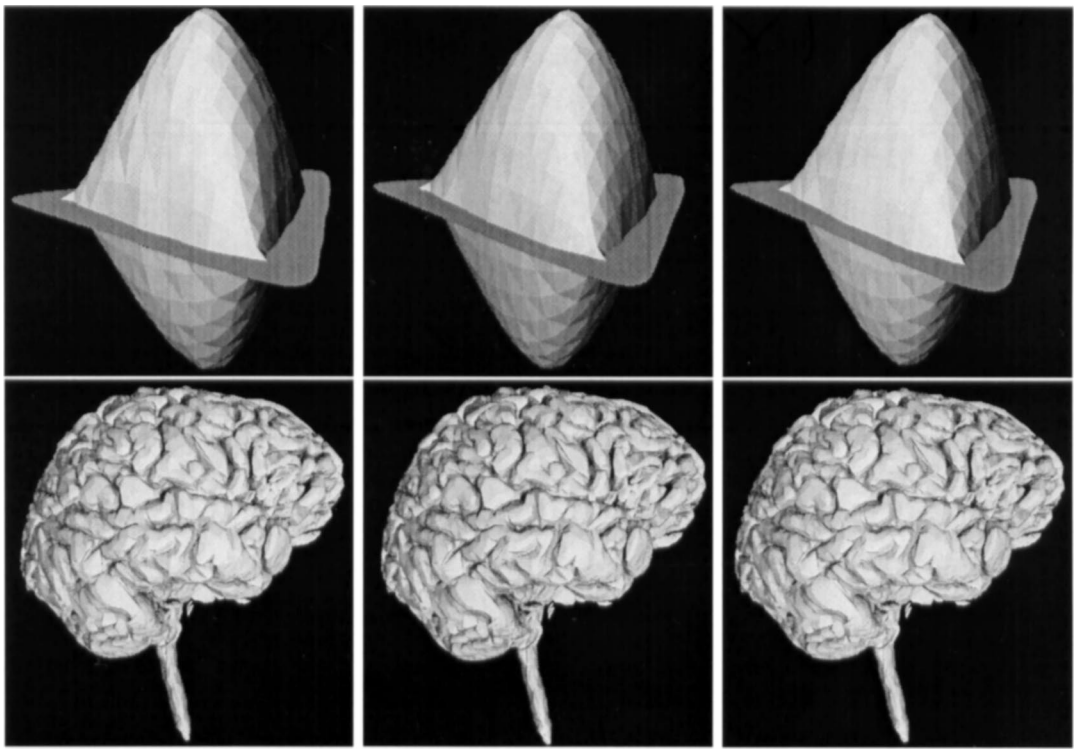


Fig. 11. Compression of non-manifold meshes. The model in the top row has 40 non-manifold vertices. The model in the top row has 260 non-manifold vertices.

8. Conclusion

We have described a space efficient encoding for both a lossless and an error-bounded lossy compression scheme for triangular meshes. The compression is achieved by capturing the redundant information in both the topology (connectivity) and geometry, and possibly property attributes. Error-bounded lossy geometry (without loss of topology) is achieved by a vector predictor and corrector encoding. Example models and results of our implementation are also provided.

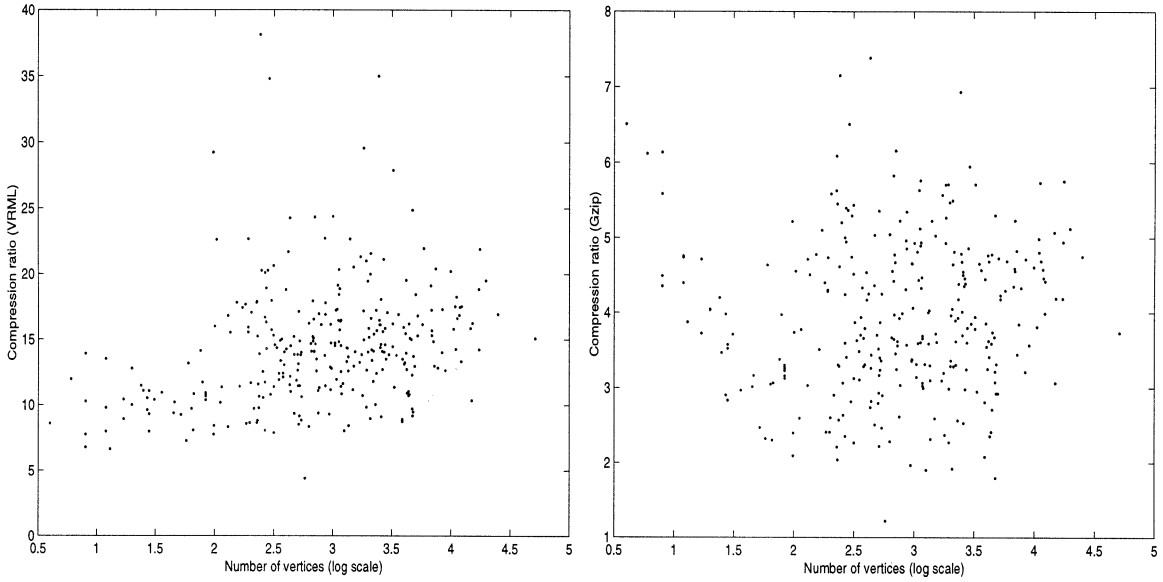


Fig. 12. Compression ratio plotted for a lossless encoding of 300 VRML models. The x coordinate of each point is the number of vertices of the corresponding model. The y coordinate is the compression ratio obtained for that model with respect to the original VRML ASCII (left) and with respect to the GZIP compressed VRML file.

Our single resolution compression and coding scheme could support error resilience because of the following locality property: (i) every triangle layer is dependent on only two adjacent vertex layers, and (ii) every vertex layer is referenced by at most two adjacent triangle layers. This implies that geometric primitives for each vertex layer and triangle layers can be independently encoded and thus decoded. The only change of our single resolution coding scheme is to perform the geometry encoding and connectivity alternatively. Our future research concentrates on taking advantage from these properties to produce incremental and progressive encoding schemes for transmission of topology and geometry of large models allowing for error recovery and error concealment even in one-way transmission channels.

References

[1] C.L. Bajaj, V. Pascucci, G. Zhuang, Progressive compression and transmission of arbitrary triangular meshes, in: Proceedings of IEEE Visualization'99, San Francisco, CA, 1999, pp. 307–316.
[2] C.L. Bajaj, V. Pascucci, G. Zhuang, Single resolution compression of arbitrary triangular meshes with properties, in: Proceedings of Data Compression Conference, 1999, pp. 247–256.
[3] M. Chow, Optimized geometry compression for real-time rendering, in: Proceedings of IEEE Visualization'97, Phoenix, AZ, 1997, pp. 347–354.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, Cambridge, MA, 1991.
[5] M. Deering, Geometric compression, in: Computer Graphics, SIGGRAPH'95 Proceedings, 1995, pp. 13–20.
[6] R.M. Gray, Vector quantization, IEEE ASSP Magazine 1 (2) (1984) 4–29.
[7] A. Gueziec, G. Taubin, F. Lazarus, W. Horn, Cutting and stitching: efficient conversion of a non-manifold polygonal surface to a manifold, Technical Report RC-20935, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1997.

[8] S. Gumhold, W. Strasser, Real time compression of triangle mesh connectivity, in: Computer Graphics, SIGGRAPH'98 Proceedings, 1998, pp. 133–140.
[9] J. Hartman, J. Wernecke, The VRML 2.0 Handbook, Addison-Wesley, Reading, MA, 1996.
[10] D. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the IRE 40 (10) (1952) 1098–1101.
[11] D. Le Gall, MPEG: A video compression standard for multimedia applications, Comm. of the ACM 34 (4) (1991) 46–58.
[12] J. Li, C.C. Kuo, Embedded coding of mesh geometry, Technical Report, ISO/IEC JTC1/SC29/WG11 MPEG98/M3325, March 1998.
[13] J. Li, J. Li, C.C. Kuo, Progressive compression of 3D graphics models, in: IEEE Proceedings of Multimedia Computing and Systems, Ottawa, Canada, 1997, pp. 135–142.
[14] J. Li, J. Li, C.C. Kuo, Progressive coding of 3D graphic models, in: IEEE Multimedia and Systems, 1998.
[15] A. Moffat, R.M. Neal, I.H. Witten, Arithmetic coding revisited, ACM Trans. Inform. Syst. 16 (3) (1998) 256–294.
[16] J. Popović, H. Hoppe, Progressive simplicial complexes, in: T. Whitted (Ed.), SIGGRAPH 97 Conference Proceedings, Annual Conference Series, ACM SIGGRAPH, Addison-Wesley, Reading, MA, 1997, pp. 217–224.
[17] J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, IEEE Trans. Visualization Comput. Graphics 5 (1), 1999.
[18] G. Taubin, J. Rossignac, Geometric compression through topological surgery, ACM Trans. Graphics 17 (2) (1996) 84–115.
[19] C. Touma, C. Gotsman, Triangle mesh compression, in: W. Davis, K. Booth, A. Fourier (Eds.), Proceedings of the 24th Conference on Graphics Interface (GI-98), Morgan Kaufmann, San Francisco, 1998, pp. 26–34.
[20] G.K. Wallace, The JPEG still picture compression standard, Comm. of the ACM 34 (4) (1991) 30–44.