Geometric Compression Through Topological Surgery

GABRIEL TAUBIN
IBM T. J. Watson Research Center
and
JAREK ROSSIGNAC
GVU, Georgia Institute of Technology

The abundance and importance of complex 3-D data bases in major industry segments, the affordability of interactive 3-D rendering for office and consumer use, and the exploitation of the Internet to distribute and share 3-D data have intensified the need for an effective 3-D geometric compression technique that would significantly reduce the time required to transmit 3-D models over digital communication channels, and the amount of memory or disk space required to store the models. Because the prevalent representation of 3-D models for graphics purposes is polyhedral and because polyhedral models are in general triangulated for rendering, this article introduces a new compressed representation for complex triangulated models and simple, yet efficient, compression and decompression algorithms. In this scheme, vertex positions are quantized within the desired accuracy, a vertex spanning tree is used to predict the position of each vertex from 2, 3, or 4 of its ancestors in the tree, and the correction vectors are entropy encoded. Properties, such as normals, colors, and texture coordinates, are compressed in a similar manner. The connectivity is encoded with no loss of information to an average of less than two bits per triangle. The vertex spanning tree and a small set of jump edges are used to split the model into a simple polygon. A triangle spanning tree and a sequence of marching bits are used to encode the triangulation of the polygon. Our approach improves on Michael Deering's pioneering results by exploiting the geometric coherence of several ancestors in the vertex spanning tree, preserving the connectivity with no loss of information, avoiding vertex repetitions, and using about three times fewer bits for the connectivity. However, since decompression requires random access to all vertices, this method must be modified for hardware rendering with limited onboard memory. Finally, we demonstrate implementation results for a variety of VRML models with up to two orders of magnitude compression.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representations*

General Terms: Algorithms, Standardization

Additional Key Words and Phrases: Geometry compression, 3D mesh compression, VRML

Authors' addresses: G. Taubin, IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598; email: ⟨taubin@watson.ibm.com⟩; J. Rossignac, GVU, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280; email: ⟨jarek.rossignac@cc.gatech.edu⟩.

1. INTRODUCTION

Although modeling systems in mechanical CAD and in animation are expanding their geometric domain to free-form surfaces, polyhedral models remain the primary 3-D representation used in manufacturing, architectural, GIS, geoscience, and entertainment industries. Polyhedral models are particularly effective for hardware-assisted rendering, which is important for video games, virtual reality, fly-through, and electronic mock-up applications involving complex CAD models.

In comparison to image and video compression, little attention has been devoted to the compression of 3-D shapes, both from the research community and from 3-D data exchange standards committees. This situation is likely to change rapidly for the following reasons.

(1) The exploding complexity of industrial CAD models significantly raises the cost of the memory and auxiliary storage required by these models.
(2) The distribution of 3-D models over networks for collaborative design, gaming, rapid prototyping, or virtual interactions is seriously limited by the available bandwidth.
(3) The graphics performance of high-level hardware adapters is limited by insufficient onboard memory to store the entire model or by a data transfer bottleneck.

Since arbitrary polygonal faces may be easily and efficiently triangulated (see, e.g., Ronfard and Rossignac [1994]), we restrict the exposition to triangular meshes. A triangular mesh is defined by the location of its vertices (positions), by the association between each triangle and its sustaining vertices (connectivity), and by color, normal, and texture information (properties) that do not affect the 3-D geometry but influence the way it is shaded.

The compressed format and the compression and decompression algorithms introduced in this article expand upon the pioneering work by Deering [1995] by providing:

(1) lossless encoding and higher compression ratios for the connectivity information (two bits less per triangle on average); Figure 1 shows an example;
(2) better organization of vertices for coordinate compression;
(3) efficient methods for building nearly optimal compressions for polyhedral models of arbitrary topology; and
(4) compression and decompression techniques that produce very long triangle strips and are thus suitable for current generation high-end graphics adapters.

2. RELATED WORK

Recent methods in 3-D compression may be divided into three categories: polyhedral simplification, compression of positions and properties, and encoding of the connectivity information.

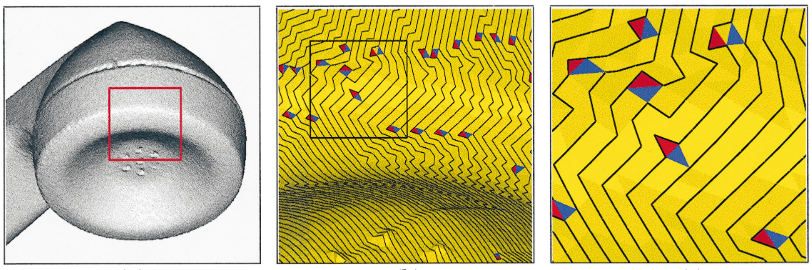


Fig. 1. This model contains 83,044 vertices, 165,963 triangles, and no properties. In VRML format it requires 8,052,266 bytes of storage. Quantizing the vertex positions to 11 bits per coordinate, our compression algorithm reduces its size to 137,130 bytes (i.e., a 58.72:1.00 compression ratio and 6.61 bits per triangle). 25,115 bytes correspond to connectivity (i.e., 1.21 bits per triangle) and 111,832 bytes correspond to positions (i.e., 5.39 bits per triangle). The remaining bytes are used to represent the scene graph structure of the VRML file. The edges of the vertex spanning tree, composed of 162 runs, are shown as black lines. The triangle spanning tree is composed of 1643 runs. Leaf triangles are shown in red, regular triangles in yellow, and branching triangles in blue.

2.1 Polyhedral Simplification

Polyhedral simplification techniques[1] reduce the number of vertices in the mesh by altering the model's connectivity and by possibly adjusting the position of the remaining vertices to minimize the error produced by the simplification. These techniques concentrate on the generation of multiple levels of detail (LOD) for accelerated graphics [Funkhouser and Sequin 1993; Borrel et al. 1995] or data reduction for over-sampled meshes [Hoppe et al. 1992]. Although these techniques could be considered for lossy compression, they are inappropriate for applications that require access to the exact connectivity of the model. In fact, simplification techniques are orthogonal to the compression techniques described here because geometric compression may be applied to each level of detail.

2.2 Compression of Positions and Properties

Lossy or lossless compression methods are used to reduce the storage necessary for the geometric data associated with vertex locations, and possibly with the normals, colors, and texture coordinates. Applying general purpose data compression algorithms to the geometric data stream leads to suboptimal solutions. We build upon Deering's [1995] approach of normalizing the geometry into a unit cube and rounding off the vertex coordinates to fixed length integers. The rounding controls the amount of lost information. We use a spatial organization of the vertices into a spanning tree and geometric predictors to replace position or property coordinates by small corrective terms, which may be encoded losslessly with fewer bits and further compressed with standard lossless entropy

[1]Please see Rossignac and Borrel [1993], Hoppe et al. [1993], Eck et al. [1995], Shroeder et al. [1992], Kalvin and Taylor [1993], Guéziec [1995], and Hoppe [1996].

encoding techniques. Artifacts created by the quantization process in meshes composed of a large number of small triangles can be reduced using mesh smoothing methods [Taubin 1995a,b; Taubin et al. 1996].

2.3 Connectivity Encoding

Connectivity encoding techniques attempt to reduce the redundancy inherent in many popular representations of polyhedral or triangular meshes in 3-D. This is the primary focus and main contribution of the present article.

Consider a triangular mesh of V vertices and T triangles. (Note that for meshes with simple topology there are roughly twice as many triangles as vertices.) Assume that the vertices are listed in a suitable order. What is the minimum number of bits required to define the T triangles sustained by these vertices?

At one extreme, if the vertices are always organized into a regular 2-D grid, the triangle mesh may be completely defined by the number of rows and columns of the grid. Regular grids may be appropriate for terrain modeling in GIS and for rendering uniformly tessellated nontrimmed rectangular parametric patches. However, they are not suitable for modeling the more general 3-D shapes found in CAD, entertainment, and other applications.

At the other extreme, each triangle may be represented by three vertex references (pointers or indices into the vertex positions array). This solution does not impose any topological limitations on the mesh, but requires storing three addresses per triangle (approximately six addresses per vertex). Even if the models were restricted to less than 1,024 vertices, this scheme would consume 60 bits per vertex for the connectivity information alone.

Triangle strips used in graphics APIs such as OpenGL [Neider et al. 1993] provide a compromise where a new vertex is combined with the previous two vertices to implicitly define a new triangle in the current strip. Triangle strips only pay off if one can build long strips, which is a challenging computational geometry problem [Arkin et al. 1994]. Furthermore, because on average a vertex is used twice, either as part of the same triangle strip or of two different ones, the use of triangle strips with OpenGL requires sending most vertices multiple times. The absence of the swap operation further increases this redundancy.

The application of triangle strips as a compression technique, where the locations of all vertices are available for random access during decompression, would still require storing one vertex reference per triangle, two vertex references per strip, the bookkeeping information on the number and length of the strips, and an additional bit of information per triangle indicating which open side of the previous triangle should be used as the basis for the next triangle (this bit is equivalent to the SWAP operation in GL).

Deering [1995] proposes using a stack-buffer to store 16 of the previously used vertices instead of having random access to all of the vertices of the model. This is a suitable solution for adapters with very limited on-board memory. Deering also generalizes the triangle strip syntax by providing more general control over how the next vertex is used and by allowing the temporary inclusion of the current vertex on the stack and the reuse of any one of the 16 vertices of the stack-buffer. The storage cost for this connectivity information is: 1 bit per vertex to indicate whether the vertex should be pushed onto the stack-buffer, 2 bits per triangle to indicate how to continue the current strip, 1 bit per triangle to indicate whether a new vertex should be read or whether a vertex from the stack-buffer should be used, and 4 bits of address for selecting a vertex from the stack-buffer each time an old vertex is reused. Assuming that each vertex is reused only once, the total cost for encoding the connectivity information is: 1 + 4 bits per vertex plus 2 + 1 bits per triangle. Assuming 2 triangles per vertex, the total cost is roughly 11 bits per vertex. (Of course other general-purpose compression schemes may be applied to the resulting bit stream, but this is the case for all the variations of geometric compression, and is ignored for this comparative analysis.) As far as we know, algorithms for systematically creating good traversals of general meshes using Deering's generalized triangle mesh syntax have not yet been developed. Naive traversal of the mesh may result in many isolated triangles or small runs, implying that a significant portion of the vertices will be sent more than once, and hence increase the number of bits per triangle.

Under the assumption that all vertex coordinates are available for random access during decompression, the solution proposed in this article produces two to three times better compression ratios than Deering's solution and outlines practical and efficient algorithms for computing nearly optimal encoding of the connectivity. As a by-product, the decompression algorithm creates very long triangle strips suitable for optimizing communication with current generation 3-D rendering adapters.

Furthermore, our compression algorithm preserves the original connectivity of the mesh, which Deering's method usually does not, and organizes the vertices by proximity, which we use to further improve compression of positions and properties.

3. OVERVIEW

The triangles of a triangle mesh may form one or more connected manifold components. In our compression scheme, the connectivity information of each connected component is encoded by first constructing a *vertex spanning tree* in the graph of vertices and edges of the component.

Because proximity in this vertex spanning tree often implies geometric proximity of the corresponding vertices, we can use ancestors in the tree to predict vertex positions, and thus only need to encode the difference between predicted and actual vertex positions. When vertex coordinates are quantized (i.e., truncated to the nearest number in a fixed-point representation scheme), these corrective vectors have on average smaller magnitude than absolute positions and can therefore be encoded with fewer bits. Furthermore, the corrective terms are then compressed by entropy encod-

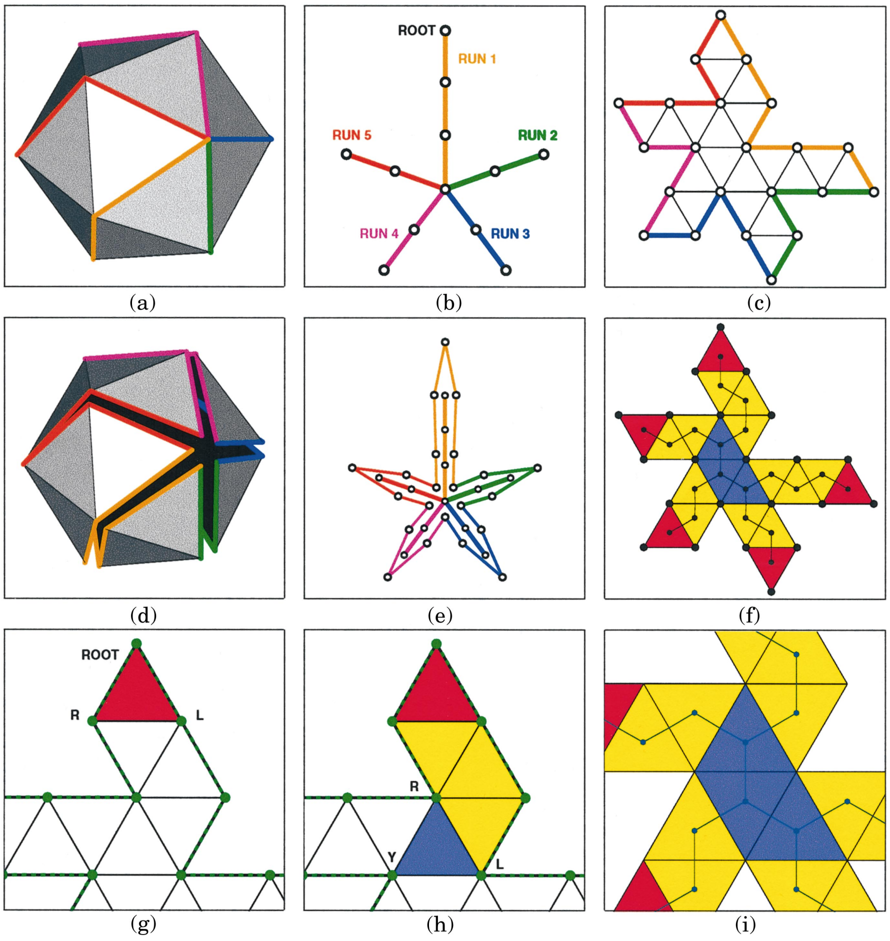


Fig. 2. Representation. The vertex spanning tree (a) (b) composed of vertex runs. Cutting through the vertex tree edges produces topological simply connected polygons (c) (d). The bounding loop (e) is the boundary of the polygon. The dual graph of the polygon is the triangle spanning tree (f). Triangle runs end in leaf or branching triangles. Leaf triangles are red, regular triangles are yellow, and branching triangles are blue. The triangle spanning tree has a root triangle (g). Marching edges (h) connect consecutive triangles within a triangle run. Each branching triangle has a corresponding Y-vertex. Two consecutive branching triangles define a run of length one (i).

ing using, for example, Huffman or arithmetic coding as in the JPEG/MPEG standards [Pennebaker and Mitchell 1993].

To encode the connectivity, the mesh is first cut through a subset of its edges, called the *cut edges*. This subset includes all the edges of the vertex spanning tree. In Section 5 we show that, depending on the topological type of the mesh, a small number of cut edges that are not vertex spanning tree edges may also be required. For example, for a simple mesh (mesh that is homeomorphic to a sphere) such as the one shown in Figure 2, we prove that there are no cut edges other than the vertex spanning tree edges.

The branching nodes and the leaf nodes of the vertex spanning tree are interconnected by *vertex runs* (i.e., by nodes that have a single child). We compress the representation of the vertex spanning tree by encoding for each vertex run: its length plus two bits of information, which collectively capture the topology of the spanning tree. To increase the compression ratio, we strive to build vertex spanning trees with the least number of runs.

This representation only captures the structure of the vertex spanning tree. The correspondence between nodes of the vertex spanning tree and vertex positions is established by storing our compressed encoding of the positions of each vertex (i.e., the entropy encoded corrective terms) in the order in which their corresponding nodes are visited by a pre-order (depth-first) traversal of the spanning tree [Tarjan 1983, Section 1.5].

When treated as a topological boundary, the cut edges organize the mesh into a set of *triangle runs* connected by branching triangles. Each *branching triangle* connects three runs (we treat the adjacency between two branching triangles as a triangle run of length one). The edges that connect triangles within a run or that bound branching triangles are called *marching edges*. We prove that an edge of a simple mesh (mesh that is homeomorphic to a sphere) is either a marching edge or a vertex spanning tree edge.

A graph whose nodes correspond to triangles and whose edges correspond to marching edges forms a binary spanning tree of the triangles of the mesh. It is the dual graph of the mesh resulting from cutting through the edges of the vertex spanning tree. We encode this *triangle spanning tree* in the same way as we encode the vertex spanning tree. However, because the triangle tree is binary, we need only store the length of each triangle run plus a single bit of information.

The combination of these two trees with the compressed vertex positions permits the recovery of the length and boundary of each triangle run and the vertices that bound each triangle. As the first step of decompression the recovery process constructs a lookup table of vertex indices that reflect the order in which the vertices appear along the bounding loop of the mesh formed by the cut edges. Figure 3 illustrates the formation of this boundary by artificially enlarging the topological discontinuity created by the cut edges, and by flattening the triangulated polygon enclosed by the bounding loop. The cutting and flattening process resembles the process of peeling an orange so that the skin remains connected. Observe that different cutting strategies produce triangle trees with different numbers of runs. We have developed a cutting strategy that appears to be effective at minimizing the number of triangle and vertex runs.

Traversing a triangle run along the direction that corresponds to a top-down traversal of the triangle tree defines the left and the right boundaries. Because the left and right boundaries of each triangle run form connected subsets of the bounding loop, we can recover the boundary of each run if we know two starting vertices (one on each side), and the number of vertices along the left and right boundary of the run.

Each marching edge shares a vertex with the previous marching edge in the triangle run. That shared vertex could lie on the left or on the right

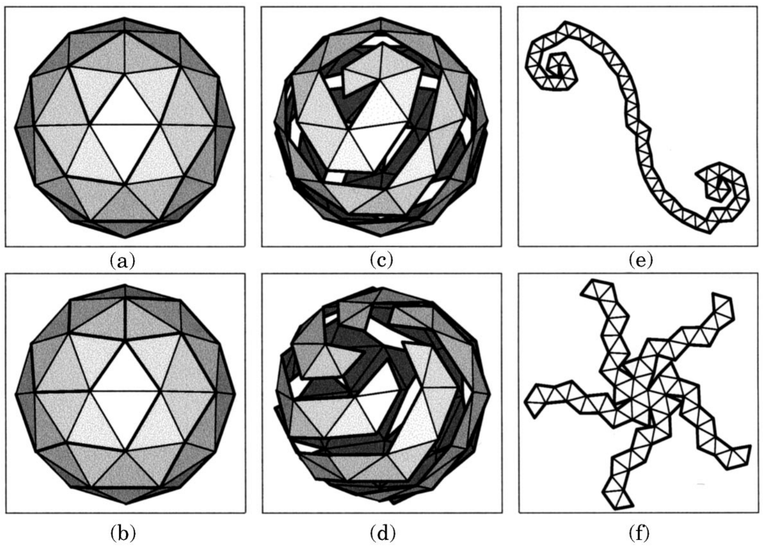


Fig. 3. Two ways of peeling an orange: (a) (b) the thick edges are the edges of the vertex tree constructed on the mesh; (c) (d) the mesh is cut through the vertex tree edges (the vertex positions have been modified here only to illustrate the creation of the cut); (e) (f) the result is a topological simply connected polygon. The dual graph of this polygon is the triangle tree.

boundary. A single bit of information per marching edge is used to encode the correct side. These bits are concatenated in the order in which the corresponding marching edges are visited by the decompression algorithm. They form what we call a *marching pattern* of left or right steps.

An entry of our representation of the triangle spanning tree indicates the number N of marching edges in a run and thus the total number of vertices on both sides of the triangle run. The number of zeros in the corresponding subset of the marching pattern indicates the number of vertices on the left side (the number of ones indicates the number of vertices on the right side).

Given two indices into the lookup table for the bounding loop (one for the starting point of the left boundary of the triangle run and one for the starting point of the right boundary), our decompression algorithm uses the next $N - 1$ bits of the marching pattern and constructs a triangle strip for the run.

At the end of the run we may encounter a leaf of the triangle spanning tree or a branching triangle. In the latter case, the last marching edge of the run forms the base of the abutting branching triangle. The third vertex of the branching triangle is called a *Y-vertex*. The corresponding index in the bounding loop is not explicitly stored in our compressed format, but is derived through a decompression preprocessing step and stored in a lookup table in the form of an index offset relative to the last vertex of the left boundary of the parent triangle run.

The decompression algorithm will visit the two runs connected to the two edges of the branching triangle in a recursive manner until the triangle spanning tree is traversed and all triangles recovered. Long triangle strips (which include SWAP operations) may be constructed by combining triangles encountered by following the connected paths visited during pre-order traversal. These paths start at the root or at a branching node and end at the leftmost leaf of the corresponding subtree.

Normals, colors, and texture mapping coordinates may be associated with the triangulated model. These are specified at the vertices of the model mainly for shading and rendering purposes, but they may be different for each use of the vertex in a different triangle. In Section 6 we show that in the compressed representation with spanning trees there is an implicit ordering for the vertices of the triangles that can be used to compress normals, colors, and texture mapping coordinates.

4. SIMPLE MESH

We first describe the compressed representation and the associated algorithms for a *simple mesh* without photometric information (properties). By a simple mesh we mean a triangulated connected oriented manifold without boundary having Euler characteristic 2 (i.e., a mesh that can be continuously deformed into a sphere). Later we remove these constraints. We extend the representation and propose a polyhedral mesh compression algorithm valid for any manifold, with one or more connected components, oriented or nonoriented, with or without boundary, and of arbitrary Euler characteristic.

4.1 Validity

The compressed representation introduced in this article is motivated by a classical result of elementary algebraic topology. Simply and intuitively, a two-manifold is a surface such that each of its points has a neighborhood that can be continuously deformed into an open disk. The classification theorem [Massey 1967, Theorem 5.1] establishes that all compact two-manifolds can be constructed by identifying pairs of edges of a simply connected polygon. This identification process can be visualized as continuously and smoothly deforming the polygon until the corresponding edges coincide, at which time the edges are sewn together with surface-normal continuity.

Every manifold triangular mesh can be constructed using this approach. It suffices to triangulate a suitable polygon without introducing new internal vertices and to identify the pairs of edges. Edges that are sewn together must belong to different triangles. The compression algorithms presented here compute a suitable polygon for a given mesh, along with the identification of the edge pairs. This alternate representation of the mesh is the basis of our compression approach.

We prove in the Appendix that when a simple mesh is cut through the edges of the vertex tree, the result is a triangulated, simply connected polygon whose dual graph (the graph with triangles as nodes and internal edges as edges) is a tree, the *triangle spanning tree*.

4.2 Representation

Assume without loss of generality that the original triangulated model with V vertices and T triangles is represented with two arrays: V vertices, each represented by its three floating point coordinates, the *vertex positions array*, and T triangles, the *triangle array*, each represented by three indices to the vertex positions array. The compressed representation of this model is composed of:

(1) VTREE: the vertex tree structure table (array of triplets—run length, branching bit, and leaf bit), that efficiently encodes a spanning tree of the graph defined by the vertices and edges of the original mesh (i.e., a tree of the cut edges);
(2) VCOR: the compressed vertex position corrections (bit stream of concatenated variable length tags) representing deviations between predicted and actual vertex positions. The predictions are based on ancestor vertex positions in the vertex tree;
(3) TTREE: the triangle tree structure table (array of pairs—run length and leaf bit), which efficiently encodes the binary tree of the triangle strips and is used to re-create the boundary of each triangle strip;
(4) MARCH: the triangle tree marching pattern (bit stream of concatenated variable length tags) representing left–right moves along triangle strips;

plus some fixed size bookkeeping information, such as the size of these tables, the parameters for geometric normalization, the vertex positions quantization parameters, the vertex positions predictor parameters, the quantized and normalized position of the root vertex of the vertex tree, and the identification of the root vertex for the triangle tree.

Picking a leaf node of a tree as the root defines an orientation for the tree edges and a parent–child relation between nodes. Nodes that have two or more children are called *branching* nodes. A rooted tree can be described as a sequence (ordered according to the pre-order traversal) of runs connecting leaves or branching nodes. The children of the branching nodes of both trees are ordered consistently with the global orientation of the mesh (either clockwise or counterclockwise) with respect to the parent node. Because the nodes are ordered according to the pre-order traversal, only the number of edges in each run and how the runs are connected must be represented.

For the vertex tree, connectivity between the runs is encoded in VTREE with two bits per run, which indicate whether the run has a right sibling (i.e., there is a further run starting at the same branching node), and whether it ends in a leaf. Because the triangle tree is binary, the bit indicating whether the run has a right sibling is omitted in the TTREE table used to encode triangle spanning trees.

Removing the branching triangles that correspond to the branching nodes of the triangle tree decomposes the polygon into connected triangle strips. The loop defined by the cut edges allows us to identify the boundary of each triangle strip. The structure of the marching edges internal to a particular strip cannot be deduced from the descriptions of the two trees. The triangle tree only indicates the number of such edges per strip, and identifies the starting marching edge. The triangles of the strip may be traversed by moving the left or right vertex of the marching edge to an adjacent vertex along the loop. One bit per triangle is needed to encode which vertex of the edge to move. The marching pattern, MARCH, is the concatenation of these bits, according to the triangle tree traversal order. It is entropy encoded for further compression. A triangle strip may end either at a leaf triangle or at a branching triangle. If it ends at a branching triangle, the next vertex is not adjacent to the marching edge along the loop. However, the indices that identify these Y-vertices need not be stored. They are derived by a simple preprocessing step of the decompression algorithm. Indeed, the distance along the loop from either the left or right vertices to a Y-vertex can be derived from the triangle tree independently of the marching pattern.

Within the vertex tree there is a unique path from each vertex to the root. The *depth* of a vertex is the length of this path, with the depth of the root vertex equal to zero. The bounding box for the solid is used to define the fixed precision format. If v_n denotes the result of quantizing to B bits the normalized relative position of a vertex of depth n within the bounding box, then each vertex position v_n is defined by

$$v_n = \epsilon(v_n) + P(\lambda, v_{n-1}, \ldots, v_{n-K}), \tag{1}$$

where $\epsilon(v_n)$ is the vertex position correction associated with that vertex, P is a vertex position's predictor function, λ and K are parameters for the predictor, and $v_{n-1}, \ldots, v_{n-K}$ are the K ancestors of the vertex along the unique path to the vertex tree root. Note that since the top vertices of the tree may not have K ancestors, we define vertex positions corresponding to negative depth as equal to the position of the vertex tree root. The vertex position corrections (integer values) are represented concatenated according to the vertex tree pre-order, and further entropy encoded.

4.3 Decompression Algorithm

Decompressing a simple mesh involves these steps, explained in detail in the following:

(1) reconstructing the table of vertex positions,
(2) constructing the bounding loop (lookup table pointing to the vertex position table),
(3) computing the relative indices for Y-vertices in the order in which they will be used, and
(4) reconstructing and linking of triangle strips.

4.3.1 *Geometry Decompression.* We first use the encoding of the vertex tree to derive the total number of vertices (the sum of the lengths of the vertex runs plus one). The vertex positions array is reconstructed from the encoding of the vertex tree. For this, the tree is traversed (depth-first) using a recursive procedure. During the tree traversal an array of indices to ancestors of vertices is maintained. After entropy decoding the vertex position corrections, the quantized relative position of the vertices is computed according to Equation (1).

4.3.2 *Building the Bounding Loop.* The bounding loop is constructed during the recursive traversal of the vertex tree and represented by a table of $2V - 2$ references to the vertex table. References to vertices encountered going down the tree are added to the table during the traversal. Except for leaf vertices, these references are also pushed onto a stack. The two bits (branching bit and leaf bit) that characterize each run of the vertex tree are used to control the tree traversal and the stack popping. When a leaf is visited, references are popped from the stack and added to the bounding loop table until the reference to the branching vertex where the next vertex run starts is popped, or until the stack is exhausted.

4.3.3 *Computing Boundary Lengths.* For computational convenience, the Y-vertices are identified not by the absolute index in the bounding loop lookup table, but by their offset in that table (i.e., their topological distance along the bounding loop) from the reference to the last vertex of the left boundary of the corresponding triangle run. These offsets are precomputed and stored in the *Y-vertex lookup table*.

For each branching triangle, the distance along the loop from either the left or right vertices to the Y-vertex, the *left branch boundary length* and *right branch boundary length*, can be computed by recursion. The length of the boundary of a branch starting with a run of length n is equal to $n + n_L + n_R - 1$, where n_L and n_R are both equal to 1 if the runs end at a leaf triangle, and equal to the left and right branch boundary lengths of the branching triangle if the run ends at a branching triangle.

The branch boundary lengths are computed for each branch as a preprocessing step of the decompression algorithm, and stored in a table. When a branching triangle is encountered during the triangle reconstruction phase, the identity of the corresponding Y-vertex can be determined by adding the left branch boundary length to the loop index of the left vertex. Because of the circular nature of the bounding loop table, this addition is performed modulo the length of the bounding loop.

4.3.4 *Triangle Reconstruction.* A reference to a single vertex identifies the root triangle in the triangle tree. Its left and right vertices are determined as the predecessor and the successor along the bounding loop. The rest of the triangles are reconstructed by recursion with the help of the left vertex stack and the right vertex stack, both initially empty. With the current left and right vertices at the beginning of a strip of n triangles, described by an entry of the triangle tree structure table, the next n bits of the marching pattern identify how many vertices of the loop are traversed on each side of the strip. If the triangle immediately after the end of the strip (the end node of the triangle run) is a branching triangle, the corresponding Y-vertex is computed as described previously, the triangle determined by the current left and right vertices and the Y-vertex is reconstructed, the Y-vertex and current right vertices are pushed onto the left and right vertex stacks, respectively, and the current right vertex is set equal to the Y-vertex. If the end triangle is a leaf triangle, the successor to the current right vertex should be equal to the predecessor to the current left vertex. In this case the leaf triangle is reconstructed and the current left and right vertices are popped off the left and right vertex stacks, if possible. If instead the stacks are empty, the reconstruction process stops (all the triangles have been reconstructed).

4.4 Compression Algorithms

Compressing a simple mesh involves

(1) constructing the vertex spanning tree,
(2) encoding the vertex tree,
(3) compressing the vertex positions,
(4) encoding the triangle tree, and
(5) computing and compressing the marching pattern.

4.4.1 *Constructing the Spanning Trees.* The compression ratio is determined mainly by the total number of runs of the vertex and triangle trees. The optimal compression is achieved by minimizing this number. We conjecture that this combinatorial optimization problem, which is very close to the construction of a Hamiltonian path, is NP-complete. Instead of attempting to compute an approximate solution using a stochastic optimization algorithm, we describe here fast deterministic methods that produce very good compression ratios.

We assign a cost to every edge of the mesh. The spanning tree of minimum total cost is constructed using a minimum spanning tree construction algorithm. Many such algorithms have been proposed [Tarjan 1983]. We first order the edges by increasing cost. Then we traverse the ordered list of edges while maintaining two structures: the set of cut edges (the edges of the vertex spanning tree) and a forest of graphs made by the marching edges. Each edge is tested and inserted into one of these two structures. If the edge connects two graphs of the forest or if it can be added to one of these graphs without forming a loop, it is a marching edge and is added to the forest structure. Otherwise it is a cut edge and is added to the set of cut edges. At the end of this process we construct a spanning tree with all the cut edges of the set.

Using the length of an edge as its cost does not produce good results, because the resulting trees have far too many branches. This is shown in Figure 4 (a). Better results are obtained by choosing a vertex as the root of the vertex tree, and setting the edge cost equal to the Euclidean distance from the edge midpoint to the vertex tree root. In this way edges closer to the vertex tree root are considered before those that are far away, and both

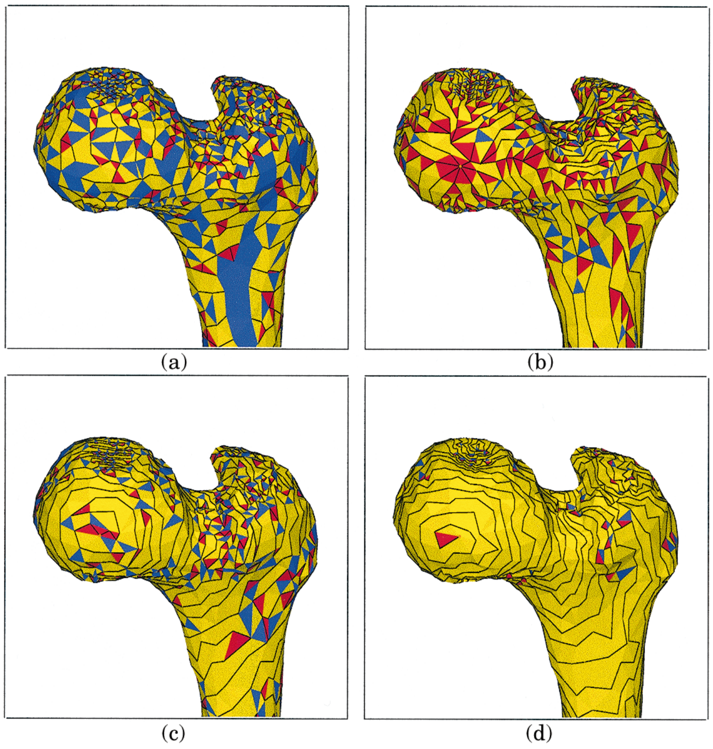


Fig. 4. Tree construction heuristics: (a) minimum spanning trees with edge cost equal to edge length; (b) minimum spanning trees with edge cost equal to distance from midpoint to vertex tree root; (c) minimum spanning trees with edge cost equal to distance from midpoint to vertex tree root, and giving priority to vertex tree edges that do not create branching nodes; (d) layered decomposition.

trees grow away from the vertex tree root eventually covering the whole mesh. This is shown in Figure 4(b).

To reduce the number of branches even further, we build the vertex tree by modifying the algorithm previously described so that during the first pass, edges are included in the forest if they create neither loops nor branching nodes. Edges that fail the test need not be cut edges. They are marked to be tested again during a second pass, this time only to verify that they do not create loops in the forest. Figure 4(c) shows an example of this construction.

Although this modified algorithm produces good results, even better compression ratios are obtained by the following approach, which performs a layered decomposition of the mesh, and an incremental construction of both trees. Intuitively, this process mimics the act of peeling an orange by cutting concentric rings, cutting the rings open, and joining them as a spiral, illustrated in Figures 3(a), 3(c), and 3(e). A vertex is chosen as the root of the vertex tree. The singleton consisting of the root vertex is the first boundary. The nth triangle layer is the set of triangles incident to one or more vertices of the nth boundary, not belonging to a previous triangle

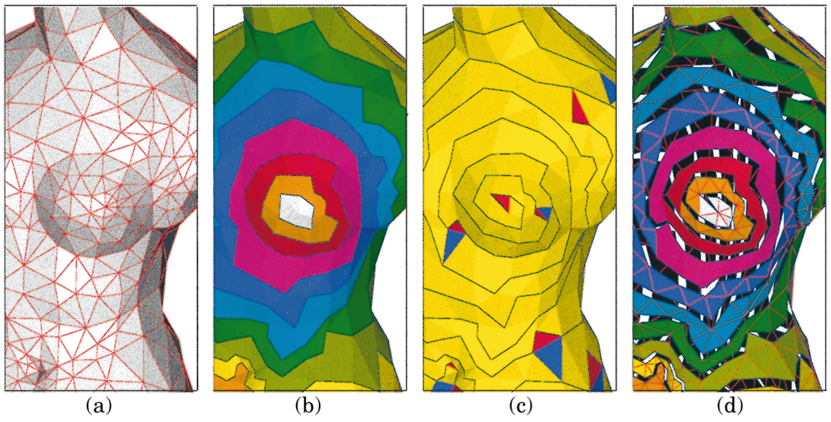


Fig. 5. Compression algorithm: (a) triangular mesh; (b) topological distance from a chosen vertex defines the layers; (c) vertex tree and triangle tree are constructed by traversing the layers in order; (d) polygon resulting from cutting along cut edges with artificial gap introduced. Triangles are color coded according to their corresponding layer.

layer. The $n + 1$st boundary consists of all the edges of triangles of the nth layer with neither of the two end vertices belonging to the nth layer. The boundary edges do not constitute a tree, but most typically each boundary is composed of one or more cycles. The layers are also typically composed of cyclical triangle paths. This construction can incrementally generate both trees by converting the rings into a spiral. Let us assume that a vertex tree has been constructed with all the vertices included in the first n boundaries, and a triangle forest has been constructed with all the triangles included in the first $n - 1$ layers. For each connected component of the $n + 1$st boundary, one edge connecting that component to a vertex of the nth boundary is chosen and added to the vertex tree. All these cross-edges are chosen minimizing the number of new branches added to the two trees. Then the edges of the $n + 1$st boundary are included in the vertex tree after removing a minimum number of edges to maintain the tree structures. These edges are also chosen minimizing the number of new branches. Figure 5 illustrates this construction.

Figure 4 illustrates the four techniques on a mesh of 5,138 triangles. Table I summarizes the results.

4.4.2 *Vertex Tree Encoding.* The vertex and triangle trees constructed by one of the algorithms described in the previous section are not rooted. To encode the vertex tree a leaf is chosen as the root node, and the tree is traversed in pre-order, with the children of the branching nodes ordered consistently with the global orientation of the mesh (either clockwise or counterclockwise) with respect to the parent. Each run of the vertex tree is represented as a record in the VTREE table. The run length is the number of edges of the run. The branching bit indicates if a run subsequent to the current one in the table starts at the same branching node. The leaf bit indicates if the run ends in a leaf node. For example, when this algorithm is

Table I.

	(a)	(b)	(c)	(d)
Bits per triangle	5.00	4.23	2.77	2.16
Vertex runs	1292	1528	80	168
Triangle runs	2388	1612	1340	526
Bits per v-run	5	4	10	8
Bits per t-run	5	5	6	8

applied to the tree of Figure 2(b), the following vertex tree structure table is generated

$$(3, 0, 0), (2, 1, 1), (2, 1, 1), (2, 1, 1), (2, 1, 0).$$

During the vertex tree traversal the bounding loop, represented by a table of $2V - 2$ references to the vertex table, is also created as in the decompression algorithm (described in Section 4.3.2).

4.4.3 *Compressing Vertex Positions.* The vertex tree is also used to encode the vertex positions based on the predictor equation (1). For example, a *linear predictor* is defined by the vertex positions predictor function

$$P(\lambda, v_{n-1}, \ldots, v_{n-K}) = \sum_{i=1}^{K} \lambda_i v_{n-i},$$

where $\lambda = (\lambda_1, \ldots, \lambda_K)$ is a vector of integers, but nonlinear predictors are also contemplated in this scheme.

Note that by choosing $K = 1$ and $\lambda_1 = 1$, the deltas used by Deering [1995] are covered as a particular case. The variables K and $\lambda_1, \ldots, \lambda_K$ can be chosen in many different ways. The compression ratio depends on this choice. We have decided to estimate $\lambda_1, \ldots, \lambda_K$ by minimizing the least square error

$$\sum_{n \geq K} \|\epsilon_n\|^2,$$

where the sum is over all the vertices of depth $n \geq K$. The concatenation of predictor errors ordered according to the vertex tree pre-order traversal are then encoded using an entropy encoding technique such as Huffman or arithmetic coding, as in the JPEG/MPEG standards [Pennebaker and Mitchell 1993]. The number of quantization bits B, and the bounding box transformation matrix are part of the compressed representation of the vertex positions as well.

4.4.4 *Triangle Tree Encoding.* During the triangle tree traversal the triangle tree structure table and marching pattern are generated. A leaf of the triangle tree is chosen as the root triangle. This triangle has two edges

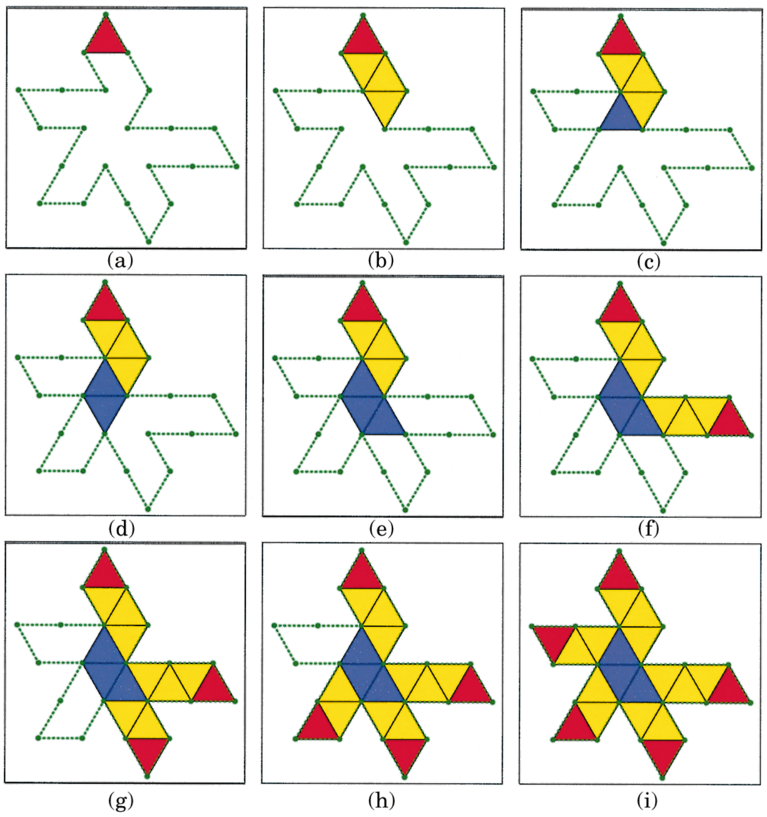


Fig. 6. Mesh reconstruction algorithm: (a) leaf triangle is reconstructed from triangle tree root id; (b) first triangle tree run is reconstructed by advancing the left and right pointers according to the marching pattern; (c) the branching triangle found at the end of the first triangle run is reconstructed from the current values of the left and right pointers and bounding loop distance from the left pointer to the corresponding Y-vertex; (d) (e) the second and third triangle runs have length one and also end in branching triangles; (f) (g) (h) (i) the remaining triangle runs end in leaf triangles.

on the bounding loop. The bounding loop index of the vertex common to those two edges is the root vertex for the triangle tree, or triangle tree root id. It is part of the compressed representation of the tree as well. This index can be determined by traveling on the surface along the cut without crossing it, keeping the cut edges on the right, and counting the visited right vertices. We start at the root of the vertex tree and stop when we encounter the tip of the first leaf triangle. See Figure 6.

Then we perform a pre-order traversal of the triangle tree. Each triangle strip connecting two leaf or branching triangles is represented by one record in the triangle structure table. The run length is the number of edges in the run, which is equal to the number of triangles in the strip plus one. The leaf bit represents whether the run ends at a leaf triangle. Since the triangle tree is binary, no branching bit is necessary. As described in the reconstruction algorithm (Section 4.3.4), the marching pattern is determined during the traversal of the strips in pre-order, as a concatenation of bits representing left or right movements of the marching edge, and further entropy encoded.

5. MORE GENERAL MESHES

Triangular manifold meshes of a Euler characteristic other than 2, nonorientable, and with boundaries, require minor extensions of the representation, compression, and decompression algorithms presented in Section 4. The compressed representation of meshes with multiple connected components consists of the concatenation of the compressed components, perhaps with common compression parameters (bounding box, number of bits per vertex coordinate, number and value of predictor coefficients, and Huffman encoding tables).

5.1 Arbitrary Euler Characteristic

When a connected oriented manifold without boundary is cut through the edges of the vertex tree, the resulting mesh is a connected oriented manifold with a simple loop as boundary, but not a simply connected polygon. As shown in the Appendix, if $\chi = E - V + T$ is the Euler characteristic of the original mesh, the new mesh has a Euler characteristic equal to $\chi - 1$. This is topologically equivalent to removing one triangle from the mesh, which in turn is equivalent to making a hole on the surface. This is illustrated in Figure 7 for a torus. Nevertheless, the fact that the Euler characteristic of a connected oriented manifold without boundary is never less than 2 [Massey 1967] implies that a simply connected polygon may be obtained by making $\chi - 1$ extra cuts, along jump edges.

Since the Euler characteristic is invariant only if the mesh remains connected, the jump edges are determined in the compression algorithm as follows. After constructing the vertex tree, the triangle tree is constructed as a spanning tree in the graph defined using the triangles as nodes, and the edges that do not belong to the vertex tree as edges. The edges of this tree are the marching edges. Edges that belong to neither the vertex tree nor the triangle tree are the *jump edges*. The cut edges now include both the vertex tree edges and the jump edges. The cut edges define a topological boundary of a simply connected polygon, and thus can be organized as a single closed loop, the *extended bounding loop*. Our representation defined for simple meshes needs to be extended to account for the jump edges. We use a new table with one entry per jump edge, the *boundary jump lengths*, indicating the number of edges in the original bounding loop it short circuits. The index in the bounding loop indicating where a jump edge starts is derived from additional information that we encode in the triangle tree table and in the marching pattern. Our decompression algorithm must be modified to reconstruct the extended bounding loop.

In this extended representation, regular (i.e., nonbranching and nonleaf) triangles of the triangle tree incident on a jump edge are treated as branching triangles with one run of length zero starting at the jump edge.

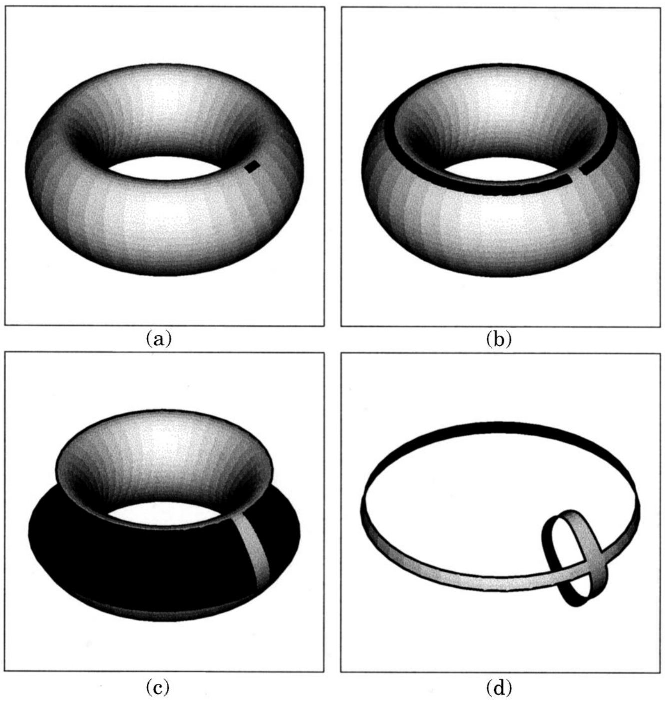


Fig. 7. Surfaces topologically equivalent to the result of cutting a torus through the vertices of the vertex tree. A connected oriented manifold with a simple loop as boundary, but not a simply connected polygon.

Leaf triangles, including the root triangle, may be incident to zero, one, or two jump edges. Furthermore, in the case of one jump edge, it may be the one incident to either the left or the right vertex. This is encoded with two extra bits per leaf of the triangle tree in the marching pattern.

5.2 Nonorientable Meshes

If the initial mesh is a nonorientable connected manifold without boundary, the children of the branching nodes of the trees do not inherit a traversal order from the orientation of the mesh. However, the local orientation of the neighborhood of the root vertex of the vertex tree can be propagated along the tree edges to all the other vertices of the mesh. The orientation of a vertex can be propagated along an edge using one triangle incident to the edge. The orientation of the first vertex is first transferred to the triangle, and then to the other vertex. Figure 8 illustrates this procedure. One of the two orientations is chosen for the root vertex. When a new vertex is visited, it can be consistently reoriented by transporting the orientation from its parent vertex. For orientable surfaces this construction produces the same result as using the global orientation of the surface to define the ordering of incident edges. However, this construction is valid for nonorientable manifolds as well. The simply connected polygon whose dual is the triangle tree is orientable. One of the two possible orientations is chosen.

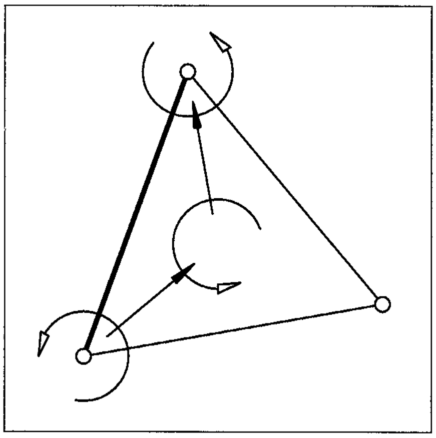


Fig. 8. Transporting the orientation from vertex to vertex along a common edge of a manifold.

In the orientable case, when a jump edge is crossed the loop path encountered after the jump is traversed in the same direction as the one before the jump. In the nonorientable case the direction of loop traversal may or may not change across a jump edge. An extra bit per jump edge is added to the marching pattern to represent the changes of direction.

5.3 Meshes with Boundary

If the initial mesh is a connected manifold with boundary, the boundary of this mesh is composed of a number of closed polygonal curves. Let us assume that the vertex tree is first constructed as usual in the graph of the mesh. If the boundary polygons are triangulated without adding new vertices, a connected manifold without boundary is obtained. We are now in the previous case, where a spanning triangle tree can be constructed, and jump edges determined. If the triangles previously added to close the boundaries of the mesh are now removed from the triangle tree, the result is a spanning forest, that is, a collection of trees. When one or more nodes are removed from a tree, together with their incident edges, the result is a forest. However, if each of the connected subtrees formed by the removed nodes and their incident edges is connected to the remaining nodes by a single edge, the remaining forest turns out to be composed of a single tree. In our case, this means that it is sufficient to include all but one of the boundary edges of each closed boundary polygonal curve in the vertex spanning tree to be sure that the mesh resulting from cutting through the vertex spanning tree edges is connected. After eventually cutting this mesh through a number of jump edges (determined as in the case of an arbitrary Euler characteristic described previously), we obtain a single simply connected polygon whose dual graph is the triangle spanning tree. Care should be taken when describing leaf triangles incident to a boundary edge of the original mesh. These must be treated as jump edges. Note that one or more of the loop boundary indices may not be used as a vertex of a triangle here.

6. PROPERTIES

Normals, colors, and texture mapping vectors are usually provided as additional information, used for shading purposes. A normal is a three-dimensional floating point vector of unit length, a color is a three-dimensional floating point vector that belongs to the unit cube $[0, 1]^3$. A texture mapping vector is a two-dimensional floating point vector. Normals, colors, and texture mapping vectors can be encoded in a similar manner. We describe the process only for normals and consider these cases:

(1) one normal per triangle,

(2) one normal per vertex, and

(3) one normal per corner (three normals per triangle).

Flat shading requires one normal per triangle. The illusion of smoothness is produced by specifying a common vertex normal for all the triangles that share a certain vertex, and then either extending the vertex intensity by continuity to the interior of the triangles (Gouraud shading), or extending the normals by continuity to the interiors of the triangles (Phong shading). Sharp edges must be shaded with different normals for each use of the shared vertices of the abutting triangles. Which of the three cases is used is included in the compressed representation of the model.

In the three cases the normals are organized into trees, which determine how they are ordered (pre-order traversal). In the case of one normal per triangle, there are T normals, one per node of the triangle tree. In the case of one normal per vertex, there are V normals, one per node of the vertex tree.

In the case of one normal per corner (three normals per triangle), there are $3T$ normals. To encode these corner properties, we use a corner tree derived from the order in which the corners are visited during the mesh reconstruction (decompression) algorithm. Many such trees can be defined. This is one possibility. The corner associated with the triangle tree root id is the root of the corner tree, followed by the left and right vertices of the triangle tree root, in that order, forming the beginning of the first run. Then, every time a new triangle is constructed of the two corners of the previous triangle incident to the connecting edge, the last one visited is connected to the corresponding corner of the new triangle on the other side of the edge; this corner is connected to the other corner of the new triangle incident to the connecting edge; and this corner is connected to the third corner of the new triangle.

Once normals are associated with the nodes of a tree they are quantized, and predictor errors are entropy encoded as in the case of the vertex positions. The normals can be quantized using Deering's [1995] nonrectilinear method, or using the same rectilinear method used for the vertex positions.

To prevent repeating values in the case of one normal per corner, the Huffman tables can be modified so that the tag consisting of the single bit "0" corresponds to "no corner value transmitted" (use last value associated

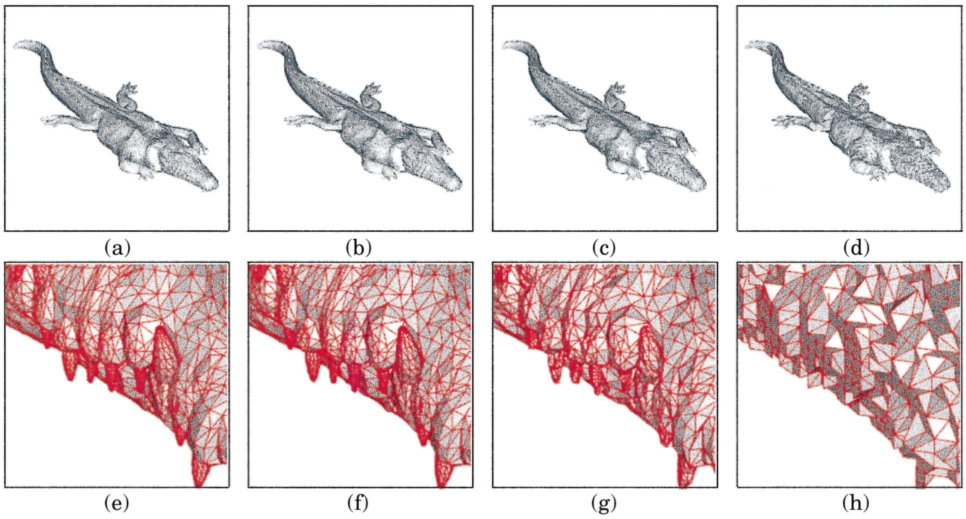


Fig. 9. Crocodile model of Table I: (a) (e) source; (b) (f) 12 bits per coordinate; (c) (g) 10 bits per coordinate; (d) (h) 8 bits per coordinate.

with corresponding vertex), and all the tags starting with bit "1" correspond to new values. A more complex scheme is implemented in Taubin et al. [1997].

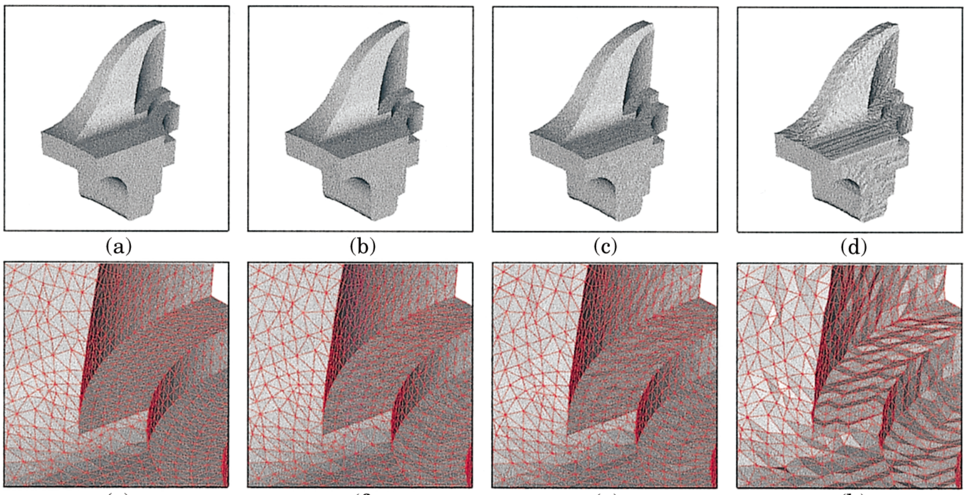
7. IMPLEMENTATION RESULTS

7.1 Test Models

The results shown in this section have been produced by our first implementation of the algorithms described in this article, which is restricted to models stored in VRML 1.0 format without properties. A few representative models are shown in Figures 9 and 10, together with compression ratios for different choices of number of bits per coordinate, and timing statistics in Tables II and III, to demonstrate the effectiveness of the technique on models of different topological and morphological characteristics. This implementation does not support all the features of the VRML 1.0 language. For example, it does not support compression of individual fields, and the geometric compression/decompression algorithms do not compress all the geometric features. The code is not fully optimized for speed.

Each example has its own page with compression results under different compression parameters and timings for parsing, compressing, decompressing, and writing. Parsing times include the time required to build the memory representation of the data, not only checking syntax. We have decided to render them with a single color and flat shaded to enhance the discretization effect.

For the experiments we have used an IBM RS/6000 model 42T, which has a PowerPC 604 processor running at 120MHz. All the timings are real-time



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 10. Fandisk of Table II: (a) (e) source; (b) (f) 12 bits per coordinate; (c) (g) 10 bits per coordinate; (d) (h) 8 bits per coordinate.

Table II. Crocodile Model with 17,332 Vertices, 34,404 Triangles, 65 Connected Components, and 0 Properties. Compression Ratios and Timing Statistics

File sizes and compression ratios

FORMAT	RESOLUTION	SIZE	PERCENTAGE	RATIO
uncompressed	32 bits/coord	1,327,559	100.00%	1:1
gziped	32 bits/coord	386,408	29.11%	3.43:1
compressed	16 bits/coord	80,056	16.58%	19.63:1
compressed	14 bits/coord	63,196	4.76%	21.00:1
compressed	12 bits/coord	48,960	3.69%	27.12:1
compressed	10 bits/coord	36,248	2.73%	36.62:1
compressed	8 bits/coord	26,788	2.02%	49.56:1

Timing

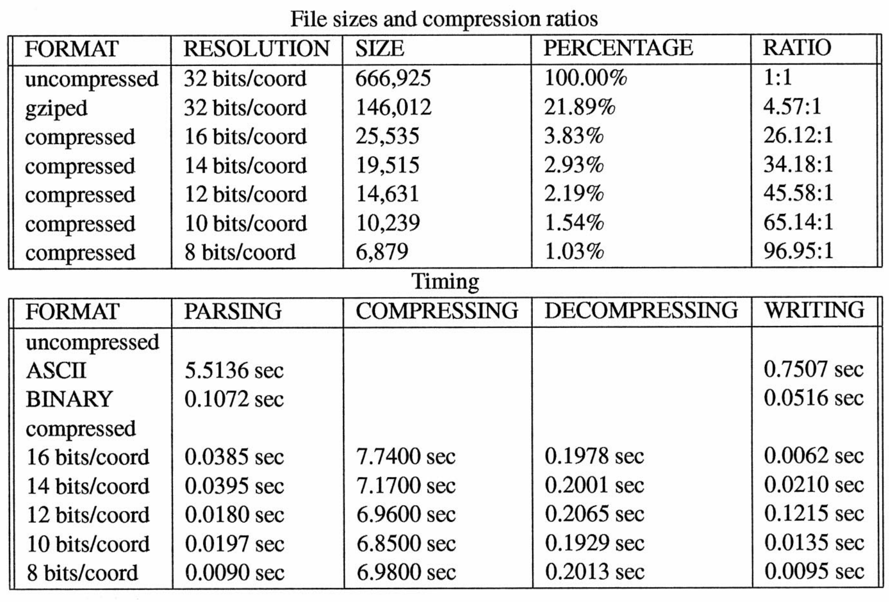
FORMAT	PARSING	COMPRESSING	DECOMPRESSING	WRITING
uncompressed				
ASCII	10.5873 sec			2.4442 sec
BINARY	0.2201 sec			0.0605 sec
compressed				
16 bits/coord	0.0982 sec	12.6789 sec	0.5276 sec	0.0280 sec
14 bits/coord	0.1023 sec	12.5634 sec	0.4974 sec	0.0255 sec
12 bits/coord	0.0871 sec	12.0230 sec	0.4757 sec	0.0242 sec
10 bits/coord	0.0736 sec	11.8670 sec	0.4636 sec	0.0243 sec
8 bits/coord	0.0507 sec	11.5300 sec	0.4383 sec	0.0266 sec

system clock measurements that include whatever else the machine was doing at the time.

Roughly speaking, these are the conclusions.

—Parsing and decompressing a file in compressed binary format is at least 20 times faster than parsing the corresponding file in ASCII format.

Table III. Fandisk Model with 6,475 Vertices, 12,946 Triangles, 1 Connected Component, and 0 Properties. Compression Ratios and Timing Statistics



File sizes and compression ratios

FORMAT	RESOLUTION	SIZE	PERCENTAGE	RATIO
uncompressed	32 bits/coord	666,925	100.00%	1:1
gziped	32 bits/coord	146,012	21.89%	4.57:1
compressed	16 bits/coord	25,535	3.83%	26.12:1
compressed	14 bits/coord	19,515	2.93%	34.18:1
compressed	12 bits/coord	14,631	2.19%	45.58:1
compressed	10 bits/coord	10,239	1.54%	65.14:1
compressed	8 bits/coord	6,879	1.03%	96.95:1

Timing

FORMAT	PARSING	COMPRESSING	DECOMPRESSING	WRITING
uncompressed				
ASCII	5.5136 sec			0.7507 sec
BINARY	0.1072 sec			0.0516 sec
compressed				
16 bits/coord	0.0385 sec	7.7400 sec	0.1978 sec	0.0062 sec
14 bits/coord	0.0395 sec	7.1700 sec	0.2001 sec	0.0210 sec
12 bits/coord	0.0180 sec	6.9600 sec	0.2065 sec	0.1215 sec
10 bits/coord	0.0197 sec	6.8500 sec	0.1929 sec	0.0135 sec
8 bits/coord	0.0090 sec	6.9800 sec	0.2013 sec	0.0095 sec

—The decompression algorithm reconstructs 60–90K triangles per second in memory, and we believe that the code is not yet fully optimized.

—The compression algorithm takes the same time to compress a scene graph, once it is in memory, as the parser takes to parse an ASCII VRML 1.0 file and to construct the memory representation.

—Writing a scene graph in uncompressed binary form is about 30 times faster than writing it in ASCII form.

—Writing the same scene graph in compressed binary form is at least 10 times faster than writing it in uncompressed binary form.

We are currently finishing a second implementation, with extended data structures and algorithms to support 3-D models described in VRML 2.0 format [Carey et al. 1997] in its full generality, including all the possible property attachments. This implementation is compliant with the *VRML Compressed Binary Format* proposal based on these extensions and on a binary encoding of the VRML scene graph structures [Taubin et al. 1997]. This proposal has been submitted to the VRML Consortium to be considered as an extension of the VRML standard. This new implementation has been tested on a large collection of 3-D models. The VRML 2.0 extensions, as well as the results of the extensive testing, are reported elsewhere [Taubin et al. 1998].

7.2 Model Size

To analyze the relation between model size (number of triangles) and compression ratios, starting from the model shown in the upper left corner

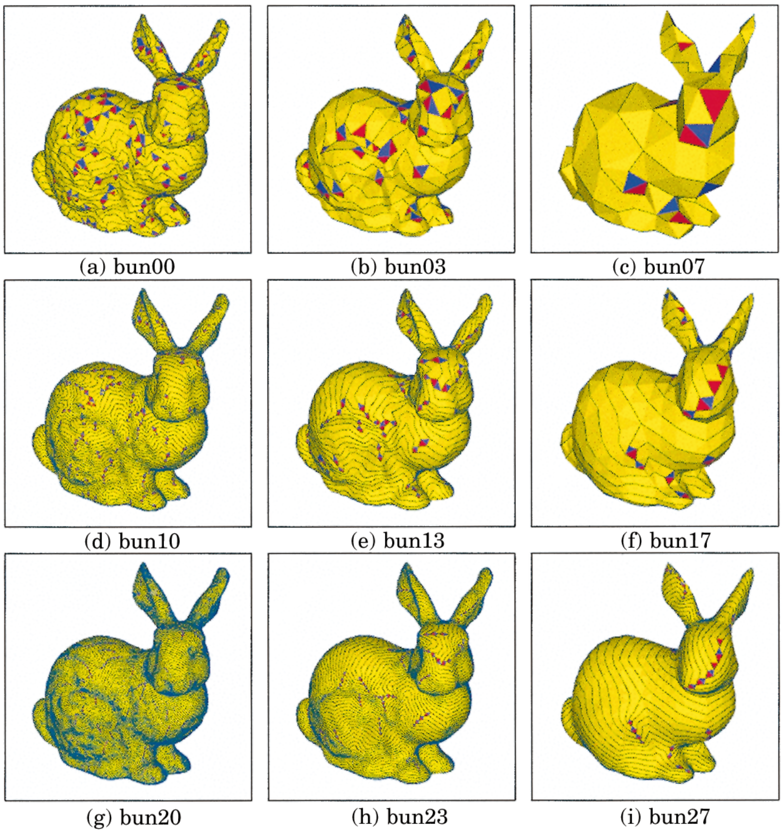


Fig. 11. Compression ratios as a function of model size. A hierarchy of 6 simplified models was generated using Guéziec's [1995] method from model (a); (b) and (c) are two of these simplified models. Then each of these models was recursively subdivided and smoothed three times (multiplying the number of triangles by four on each subdivision step) using Taubin's [1995a] method. The first two levels of subdivision and smoothing corresponding to the models in the first row are shown in the second and third rows. Complete statistics on the 28 models are shown in Table IV.

of Figure 11 (4,737 triangles), we generated a hierarchy of 7 simplified models using Guéziec's [1995] method. We then recursively applied three consecutive subdivision and smoothing steps using Taubin's [1995a] method to increase the number of triangles. The size of the resulting 28 models, all having the same topology and similar shape, span more than three orders of magnitude (279 to 303,168 faces). Some of these models are shown in Figure 11 with the vertex tree edges drawn in black, and the triangles color-coded according to their position in the triangle tree (leaf triangles are red, regular triangles are yellow, and branching triangles are blue). Table IV contains the number of vertices and faces of each model, as well as their uncompressed sizes (as VRML files), total compressed sizes, total number of bits per triangle, size of the connectivity information alone, number of bits per triangle used to encode the connectivity information, and compression ratios. The plots of Figure 12 show how the total number

Table IV. Relation Between Model Size (Number of Faces) and Compressed Size per Face for Models of Same Topology and Similar Shape. Some Models Are Shown in Figure 11.

NAME	MODEL		SIZE	COMPRESSED				RATIO
	NUMBER OF		ASCII	TOTAL SIZE		CONNECTIVITY		
	VERTICES	FACES	(VRML)	BYTES	BITS/TRI	BYTES	BITS/TRI	
bunny-00	2,425	4,737	232,529	5,505	9.29	1,497	2.52	42.24:1
bunny-01	1,709	3,308	163,155	4,101	9.91	1,075	2.59	39.78:1
bunny-02	1,178	2,254	111,856	3,170	11.25	988	3.50	35.26:1
bunny-03	758	14,35	71,662	2,188	12.19	578	3.22	32.75:1
bunny-04	524	990	49,562	1,583	12.79	368	2.97	31.30:1
bunny-05	342	644	32,376	1,121	13.92	234	2.90	28.88:1
bunny-06	226	426	21,488	835	15.68	167	3.13	25.73:1
bunny-07	149	279	14,201	624	17.89	115	3.29	22.76:1
bunny-10	9,589	18,948	924,395	15,208	6.42	4,608	1.94	60.78:1
bunny-11	6,728	13,232	647,034	11,279	6.81	3,276	1.98	57.37:1
bunny-12	4,612	9,016	442,198	8,617	7.64	2,467	2.18	51.32:1
bunny-13	2,953	5,740	282,367	5,941	8.28	1,633	2.27	47.53:1
bunny-14	2,040	3,960	195,002	4,253	8.59	1,009	2.03	45.85:1
bunny-15	1,330	2,576	127,068	3,066	9.52	697	2.16	41.44:1
bunny-16	880	1,704	84,146	2,173	10.20	462	2.16	38.72:1
bunny-17	579	1,116	55,313	1,590	11.39	333	2.38	34.79:1
bunny-20	38,128	75,792	3,686,594	43,402	4.58	14,333	1.51	84.94:1
bunny-21	26,690	52,928	2,577,420	32,259	4.87	10,412	1.57	79.90:1
bunny-22	18,242	36,064	1,758,796	23,534	5.22	7,529	1.67	74.73:1
bunny-23	11,648	22,960	1,121,362	16,020	5.58	4,830	1.68	70.00:1
bunny-24	8,042	15,840	773,972	11,499	5.80	3,207	1.61	67.30:1
bunny-25	5,238	10,304	503,856	8,039	6.24	2,167	1.68	62.68:1
bunny-26	3,466	6,816	333,428	5,632	6.61	1,336	1.56	59.20:1
bunny-27	2,276	4,464	218,726	4,126	7.39	945	1.69	53.01:1
bunny-30	152,050	303,168	15,037,160	141,852	3.74	47,267	1.24	106.00:1
bunny-31	106,310	211,712	10,326,564	104,667	3.95	35,118	1.32	98.66:1
bunny-32	72,550	144,256	7,015,648	75,426	4.18	25,110	1.39	93.01:1
bunny-33	46,258	91,840	4,469,698	48,675	4.24	17,679	1.54	91.83:1
bunny-34	31,926	63,360	3,084,272	34,595	4.36	11,194	1.41	89.15:1
bunny-35	20,782	41,216	2,007,048	23,345	4.53	6,980	1.35	85.97:1
bunny-36	13,750	27,264	1,327,856	15,911	4.66	4,453	1.30	83.46:1
bunny-37	9,018	17,856	870,308	11,260	5.04	3,171	1.42	77.29:1

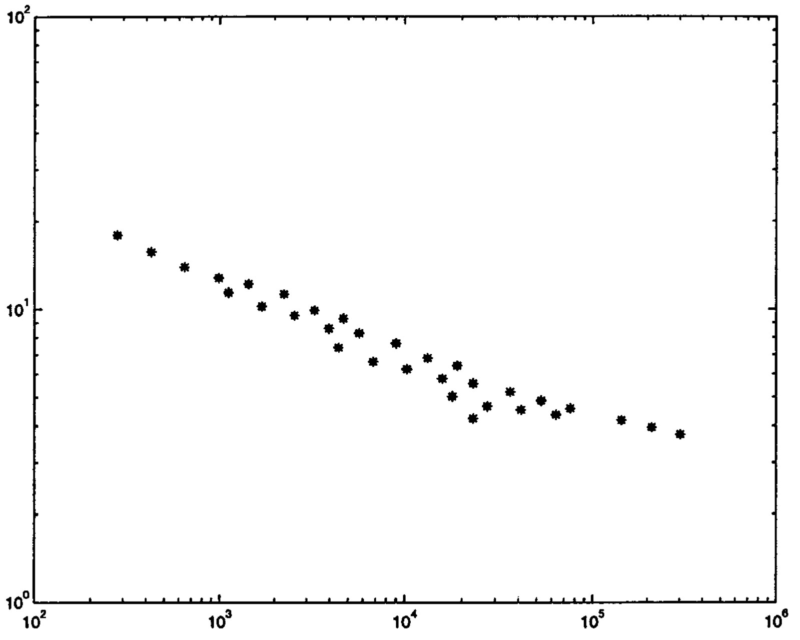
of bits per triangle and the number of connectivity bits per triangle vary as a function of the number of triangles. As expected, the compression pays off more for larger models and models with more regular meshes (such as those obtained from recursively subdividing a mesh).

7.3 Algorithmic Complexity

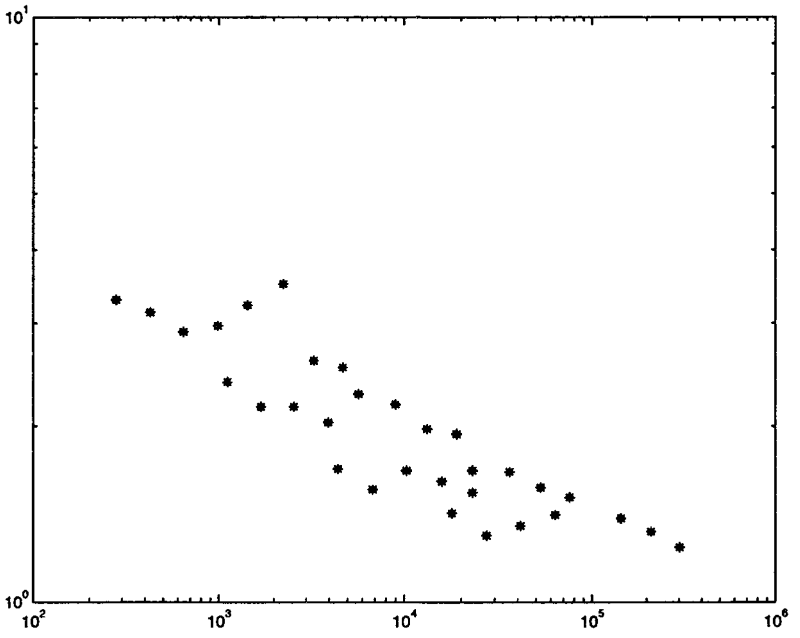
Let N be the maximum of the number of vertices V, the number of edges E, and the number of triangles T of the triangular mesh.

7.3.1 *Decompression Algorithm.* Overall, the time and space complexity of the decompression algorithm is $O(N)$.

Since the number of runs of the vertex tree is always less than the number of vertices, the time complexity of traversing the vertex tree and constructing the bounding loop lookup table is clearly $O(V)$. In terms of



A



B

Fig. 12. Relation between body size (number of faces) and compressed size per face for the 28 models of Table IV: (a) number of triangles vs total number of bits per triangle; (b) number of triangles vs number of connectivity bits per triangle.

storage, a stack of at most depth V is needed for the traversal, and an array of length $2V - 2$ is needed to store the bounding loop lookup table.

The complexity of decoding the entropy encoded vertex positions is clearly linear in the number of vertices, because the total number of bits of the encoded data is bounded above by the total number of bits per coordinate times the number of coordinates times the number of vertices, and these bits are sequentially read during decompression. The entropy decoding requires an array of length $3V$ to store the decoded vertex positions and a few local variables.

The Y-vertex lookup table is constructed by traversing the triangle tree. Since there is one Y-vertex per branching triangle, no more than one branching triangle per triangle run, and no more triangle runs than triangles, the time and space complexity of building the Y-vertex lookup table is at most $O(T)$.

The time and space complexity of reconstructing the triangles is also $O(T)$ because each triangle is visited exactly once during the depth-first traversal of the triangle tree, reconstructing the triangle only requires looking at a value in the bounding-loop lookup table or in the Y-vertex lookup table (both of which require constant time access, for regular triangles consuming a bit from the marching pattern), the stacks needed for the traversal are at worst of length T, and the array needed to store the reconstructed triangles is of length $3T$.

7.3.2 *Compression Algorithm.* Let us first assume that the triangular mesh is a connected manifold.

For efficient implementation, the compression algorithm requires average constant time access to the edges of the mesh indexed by the two vertex indices of the endpoints. The edge data structure should also provide constant time access to the two incident triangles. In our implementation we represent an edge as an array of four integers, the first two corresponding to the two vertex indices, and the second two to the triangle indices (boundary edges only have one incident triangle; a special marker is used to indicate invalid triangles). The edges are organized as a hash table indexed by the sorted pair of vertex indices (smaller vertex index followed by larger vertex index). In our implementation, this hash table is an array of length V of linked lists. When an edge is created, it is inserted in the list corresponding to the smaller vertex index. Each list is sorted by the value of the other vertex index. Since for most typical meshes the maximum number of edges incident to a vertex is reasonably small, we have verified that in practice this implementation satisfies the requirement. The edges are constructed in $O(T)$ time by visiting each triangle, and for each of the three pairs of vertices, first looking for the corresponding edge in the hash table. If the edge is not found in the hash table, it is created and the two vertex index entries filled, one of the triangle index entries with the current values, and the other triangle index with the special marker previously mentioned. If the edge is found in the hash table the special marker is replaced with the index of the current triangle.

Building the vertex and triangle spanning trees (i.e., classifying each edge as belonging to the vertex spanning tree, triangle spanning tree, or being a jump edge) requires maintaining forest data structures for both the vertex and triangle spanning trees. This can be done using the fast union-find algorithms of Tarjan [1983] to maintain a partition of the vertices. Each part represents one tree of the forest. Initially, there are as many trees as vertices, and each tree is composed of one node and no edges. A similar structure is used for the triangle forest. Then one by one the edges are considered for inclusion in one of the two forests. If considered for inclusion in the vertex forest, it is included only if the two endpoints belong to different trees (different subsets of the partition), in which case the edge is marked as belonging to the vertex tree, and the two corresponding subsets in the partition are joined into a single subset. The same is done when considering the edge for inclusion in the triangle tree. If an edge can neither be included in the vertex tree nor in the triangle tree, then it is marked as a jump edge. The order in which the edges are considered and whether each edge is first considered for inclusion in the vertex forest or in the triangle forest depends on the particular algorithm. But once these decisions are made, this process requires essentially $O(N)$ time and space.

All the tree construction algorithms based on variations of the minimum spanning tree algorithm have a time complexity of $O(N \log N)$ because of the sorting of the edges as a function of cost. The complexity of the algorithm based on the layer decomposition is $O(N)$ however, because of its mesh traversal nature.

Once the edges are classified, the vertex tree is depth-first traversed on the mesh to encode it as a table of vertex runs. The runs incident to each branching node are ordered according to the orientation of the mesh, which requires turning around the branching vertex. The time and space complexity of this operation is $O(N)$. The situation is similar for the triangle spanning tree, but the marching pattern is also generated while traversing the triangle spanning tree. Choosing the triangle tree root requires traversing the vertex tree and looking at the incident triangles until a triangle with two consecutive edges on the vertex spanning tree is found. This also requires $O(N)$ time to complete.

Quantizing entropy encoding the vertex positions clearly requires $O(N)$ space and time.

Overall, all the algorithms require $O(N)$ space. The algorithms based on the minimum spanning tree constructions require $O(N \log N)$ time, and our best algorithm based on the layered decomposition requires only $O(N)$ time.

The determination of the number of connected components, and the partition of the triangle mesh into connected components comes for free from the construction of the spanning forests described previously. If at the end of the construction the vertex forest consists of a single tree (i.e., the partition is composed of a single subset including all the vertices), then the triangle mesh is connected. Otherwise each tree of the vertex forest corresponds to one connected component.

There are other preprocessing steps that are necessary in a practical implementation, such as checking whether the triangular mesh is a manifold, and if not, converting it to a manifold. In this article we assume that the input triangular mesh is a manifold, perhaps with many connected components, and we do not attempt to check and/or fix it. Algorithms to do so, and their complexity, are reported elsewhere.

7.4 Optimality Analysis

The method described in this article to losslessly encode the connectivity of a triangle constitutes a variable-length encoding scheme. We do not have any optimality claims on it, but in practice it seems to approach one bit per triangle as the number of triangles grows with constant topology. This is essentially because the size of the vertex and triangle trees becomes negligible with respect to the marching pattern, which in our current implementation is not compressed. By compressing the marching pattern we could probably obtain even smaller asymptotic rates.

To our knowledge, there are no results on the minimum number of bits required to encode a triangular mesh. However, it is well known how many different triangulations of a simply connected polygon exist. The number of different triangulations of a simply connected polygon of $n + 2$ vertices (which is triangulated with n triangles) is given by the Catalan number [Knuth 1973]

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{2n!}{n!(n+1)!}.$$

If these triangulations are systematically enumerated, and the corresponding number is used as the encoding of the triangulation, then $\log_2(\lceil C_n \rceil)$ bits are needed to encode it. Note that this would be the optimal fix-length encoding scheme. Using Stirling's formula it is not difficult to verify that $\log_2(\lceil C_n \rceil) \rightarrow 2$, as $n \rightarrow \infty$. Our variable-length encoding scheme produces better results on the class of polygon triangulations than our mesh-cutting scheme produces (i.e., triangulations with very few branching and leaf triangles).

8. CONCLUSION

In this article we have introduced a new compressed representation for triangular meshes. In this representation the connectivity of the mesh is encoded with no loss of information, and the vertex positions and properties are compressed with variable loss of information. The scheme is particularly appropriate for network-based 3-D applications. We have shown examples and benchmarks produced with our first VRML 1.0 implementation. Results obtained with the more complete VRML 2.0 implementation, as well as the extensions necessary to support the 3-D models that can be represented in that format, are reported elsewhere. As shown in the examples, compression ratios of 50:1 are not unusual. Other existing compression schemes either do not preserve the connectivity of the mesh, or do not achieve these compression ratios. The natural next step is to extend and/or modify this scheme to support progressive transmission, that is, level-of-detail hierarchies, of 3-D models.

9. APPENDIX

THEOREM 1. *The vertex tree cuts a simple mesh into a triangulated simply connected polygon (i.e., a topological disk).*

PROOF. Cutting an orientable surface does not destroy its orientability. The vertex tree has no cycles, and therefore the surface it bounds is connected. The Euler characteristic $\chi = V - E + T$ of a simple mesh with V vertices, E edges, and T triangles is 2 by definition. The vertex tree is composed of the V vertices of the mesh as nodes and $V - 1$ edges. Cutting the mesh through the edges of the vertex tree produces a new connected mesh with a single boundary loop of edges composed of $2V - 2$ nodes and $2V - 2$ edges (each cut edge is used twice in the loop). The mesh bounded by the cut loop has no internal vertices. Each vertex of the original mesh is used once, twice, or more times in the boundary loop, depending on whether it is a leaf node, a regular node, or a branching node of the vertex tree. Since the resulting mesh has $2V - 2$ vertices, $E + V - 1$ edges, and T triangles, its Euler characteristic is $(2V - 2) - (E + V - 1) + T = (V - E + T) - 1 = 1$. Because the mesh is orientable and connected, it is homeomorphic to a topological disk, such as a triangle or a simply connected polygon (both are triangulated connected, orientable 2 manifolds with the same Euler characteristics) (Massey 1967, Theorem 8.2). □

THEOREM 2. *The dual graph of a triangulated simply connected polygon is a binary tree.*

PROOF. The tree is clearly binary. If the dual graph had a cycle, it would decompose the loop into two disjoint components (one of which could be an isolated point, or would imply that the polygon has an internal vertex). □

ACKNOWLEDGMENTS

We would like to thank Bill Horn for providing us with a VRML translator and for integrating it with these technologies, Francis Lazarus and Andre Guéziec for contributing ideas and helping with the implementation, Greg Turk and David Luebke for pointing out the results on the number of triangulations of planar polygons, and Chandra Narayanaswami for pointing out the results on graph encoding.

REFERENCES

ARKIN, E. M., HELD, M., MITCHELL, J. S. B., AND SKIENA, S. S. 1994. Hamiltonian triangulations for fast rendering. In *Proceedings of Algorithms—ESA '94* (Utrecht, NL, Sept.), J. van Leeuwen, ED., LNCS 855, 36–47.

BORREL, P., CHENG, K. S., DARMON, P., KIRCHNER, P., LIPSCOMB, J., MENON, J., MITTLEMAN, J., ROSSIGNAC, J., SCHNEIDER, B. O., AND WOLFE, B. 1995. The IBM 3D interaction accelerator (3DIX). Tech. Rep. RC 20302, IBM Research.

CAREY, R., BELL, G., AND MARRIN, C. 1997. The virtual reality modeling language. Tech. Rep. ISO/IEC DIS 14772-1, April. Available at `http://www.vrml.org/Specifications/VRML97/DIS`.

DEERING, M. 1995. Geometric compression. *Comput. Graph. (Proc. SIGGRAPH)* (August), 13–20.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. *Comput. Graph. (Proc. SIGGRAPH)* (August), 173–182.

FUNKHOUSER, T., AND SEQUIN, C. 1993. Adaptive display algorithm for interactive frame rate during visualization of complex virtual environments. *Comput. Graph. (Proc. SIGGRAPH)* (August), 247–254.

GUÉZIEC, A. 1995. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery* (Baltimore, MD, Nov.).

HOPPE, H. 1996. Progressive meshes. *Comput. Graph. (Proc. SIGGRAPH)* (August), 99–108.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *Comput. Graph. (Proc. SIGGRAPH)* (July), 71–78.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. *Comput. Graph. (Proc. SIGGRAPH)* (August), 19–26.

KALVIN, A. AND TAYLOR, R. H. 1996. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Comput. Graph. Appl. 16*, 3 (May), 64–77.

KNUTH, D. E. 1973. *The Art Of Computer Programming*. Addison-Wesley, Reading, MA.

MASSEY, W. S. 1967. *Algebraic Topology, An Introduction.* Harcourt, Brace & World, Orlando, FL.

NEIDER, J., DAVIS, T., AND WOO, M. 1993. *OpenGL Programming Guide*. Addison-Wesley, Reading, MA.

PENNEBAKER, B. P. AND MITCHELL, J. L. 1993. *JPEG, Still Image Compression Standard.* Van Nostrand Reinhold, New York.

RONFARD, R. AND ROSSIGNAC, J. 1994. Triangulating multiply-connected polygons: A simple, yet efficient algorithm. In *Proceedings of Eurographics '94 Computer Graphics Forum*, (Oslo, Norway).

ROSSIGNAC, J. AND BORREL, P. 1993. *Geometric Modeling in Computer Graphics.* Springer Verlag, 455–465.

SHROEDER, W. J., ZARGE, A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. *Comput. Graph. (Proc. SIGGRAPH)* 65–70.

TARJAN, R. E. 1983. *Data Structures and Network Algorithms.* In *CBMS-NSF Regional Conference Series in Applied Mathematics*, No. 44, SIAM, Philadelphia.

TAUBIN, G. 1995a. A signal processing approach to fair surface design. *Comput. Graph. (Proc. SIGGRAPH)* (August), 351–358.

TAUBIN, G. 1995b. Curve and surface smoothing without shrinkage. In *Proceedings of the Fifth International Conference on Computer Vision* (June), 852–857.

TAUBIN, G., HORN, W. P., AND LAZARUS, F. 1997. The VRML compressed binary format proposal, June. Available at `http://www.research.ibm.com/vrml/binary`.

TAUBIN, G., HORN, W. P., LAZARUS, F., AND ROSSIGNAC, J. 1998. Geometric coding and VRML. *Proc. IEEE 86*, 7 (Special issue on multimedia signal processing).

TAUBIN, G., ZHANG, T., AND GOLUB, G. 1996. Optimal surface smoothing as filter design. In *Proceedings of the European Conference on Computer Vision* (Cambridge, UK, April), 283–292.

Received July 1997; accepted November 1997

Editor: Ari Rappoport