3D Geometry Compression and Progressive Transmission

Gabriel Taubin

IBM T.J.Watson Research Center †

Abstract

Polygonal meshes remain the primary representation for visualization of 3D data in a wide range of industries, including manufacturing, architecture, geographic information systems, medical imaging, robotics, entertainment, and military applications. Because of its widespread use, it is desirable to compress polygonal meshes stored in file servers and exchanged over computer networks to reduce storage and transmission time requirements. In this report we describe several schemes that have been recently introduced to represent single and multi-resolution polygonal meshes in compressed form, and to progressively transmit polygonal mesh data. The progressive transmission of polygonal meshes allows the decoder process to make part of a single-resolution mesh, or the low resolution levels of detail of a multi-resolution mesh, available to the rendering system before the whole bitstream is fully received and decoded. It is desirable to combine compression and progressive transmission, but not all the existing methods exhibit both features. These progressive transmission schemes are closely related to surface simplification or decimation methods, which change the surface topology while approximating the geometry, and can be regarded as lossy compression schemes as well. Finally, we describe in more detail the Topological Surgery and Progressive Forest Split schemes that are currently part of the MPEG-4 multimedia standard.

1. Introduction

Polygonal meshes are the primary representation used in the manufacturing, architectural, and entertainment industries for visualizing 3D data, and they are central to Internet and broadcasting multimedia standards such as VRML [49], and MPEG-4 [26] [29] [6]. In these standards, a polygonal mesh is defined by the position of its vertices (geometry); by the association between each face and its sustaining vertices (connectivity); and optional colors, normals and texture coordinates (properties).

It is desirable to compress polygonal mesh data to reduce storage and transmission time requirements. In this report we describe the recent efforts in this area. Deering [9] introduced the first geometry compression scheme to compress the bitstream sent by a CPU to a graphics adapter, generalizing the popular triangle strips and fans. Motivated by Deering's work, but optimized for transmission over the Internet instead, Taubin and Rossignac introduced the Topological Surgery (TS) scheme [45], the first connectivity-preserving single-resolution manifold triangular mesh compression scheme. TS was later extended to handle arbitrary manifold polygonal meshes with attached properties, and proposed as a compressed file format to encode VRML files [42]. And with a more efficient encoding, Topological Surgery is now part of the MPEG-4 standard.

Several closely related methods were subsequently developed by Touma and Gotsman [47], Gumhold and Strasser [17], Li and Kuo [27] and Rossignac [36]. The methods proposed by Gumhold and Strasser, and by Rossignac only encode connectivity. The method proposed by Touma and Gotsman, predicts geometry an properties better, and the method proposed by Li and Kuo improves on the entropy encoding of prediction errors. More recently, Bajaj et. al. [2] proposed yet another method to encode single-resolution triangular meshes. It is based on a decomposition of the mesh into rings of triangles originally used by Taubin and Rossignac in their compression algorithm, but with a different and more complex encoding. All of these schemes require $O(n)$ total bits

† P.O.Box 704 Yorktown Heights, NY 10598, USA, taubin@us.ibm.com

© Gabriel Taubin 1999. Published by the Eurographics Association, ISSN 1017-4656.

of data to represent a single-resolution mesh in compressed form.

While single resolution schemes can be used to reduce transmission bandwidth, it is frequently desirable to send the mesh in progressive fashion. A progressive scheme sends a compressed version of the lowest resolution level of a level-of-detail (LOD) hierarchy, followed by a sequence of additional refinement operations. In this manner, successively finer levels of detail may be displayed while even more detailed levels are still arriving. To prevent visual artifacts, sometimes referred to as *popping*, it is also desirable to be able to transition smoothly, or *geomorph*, from one level of the LOD hierarchy to the next by interpolating the positions of corresponding vertices in consecutive levels of detail as a function of time.

The Progressive Mesh (PM) scheme introduced by Hoppe [19] was the first method to address the progressive transmission of multi-resolution manifold triangular mesh data. PM is an *adaptive refinement* scheme where new faces are inserted in between existing faces. Every triangular mesh can be represented as a base mesh followed by a sequence of *vertex split* refinements. Each vertex split is specified for the current level of detail by identifying two edges and a shared vertex. The mesh is refined by cutting it through the pair of edges, splitting the common vertex into two vertices and creating a quadrilateral hole, which is filled with two triangles sharing the edge connecting the two new vertices. The PM scheme is not an efficient compression scheme. Since the refinement operations perform very small and localized changes, the scheme requires $O(V \log_2(V))$ bits to double the size of a mesh with V vertices. Later on Hoppe proposed a more efficient implementation based on changing the order of transmission of the edge split operations [20].

In the Progressive Forest Split (PFS) scheme introduced by Taubin et. al. [40] a manifold triangular mesh is also represented as a low resolution polygonal model followed by a sequence of refinement operations. But the *forest split* refinement operation, which can be seen as a grouping of several consecutive edge split operations into a set instead of a sequence, provides a tradeoff between compression ratios and granularity. The highest compression ratios are achieved by reducing the high number of levels of detail produced by the PM scheme, which are usually not required. Best compression ratios are achieved in PFS when the size grows exponentially with level of detail, because it requires only $O(V)$ bits to double the size of a mesh with V vertices.

The topological type (genus) of the lowest resolution level of detail stays constant during the refinement process for both the PM and PFS methods. The Progressive Simplicial Complexes (PSC) scheme introduced by Popovic and Hoppe [34] allows changes in topological type to occur during the refinement process. The PSC representation retains most of the advantages of the PM representation, including smooth transitions between consecutive levels of detail and progressive transmission. However, the increased generality of the method requires an even higher number of bits to specify each refinement operation. A closely related scheme, called MetaStream, was more recently introduced by Abadjev et. al. [1].

In all the previous progressive representations, multi-resolution polygonal models are represented in compressed form. But seen as compression schemes are not as efficient as the single-resolution schemes described earlier. Taubin et. al. [41] recently introduced a method to compress any multi-resolution mesh produced by a vertex clustering algorithm with compression ratios comparable to the best single-resolution schemes. In this scheme, the connectivity of the LOD hierarchy is transmitted from high resolution to low resolution, followed by the geometry and properties from low resolution to high resolution. The main contribution of this scheme is a method to compress the *clustering* mappings which relate consecutive levels of detail, from high to low resolution. The method achieves high compression ratios but is not progressive.

The MPEG-4 3D Mesh Coding scheme is based on the Topological Surgery and Progressive Forest Split schemes. But incorporates improvements to connectivity encoding for progressive transmission proposed by Bossen [5], non-manifold encoding proposed by Guéziec et. al. [16], error resiliency proposed by Jang et. al. [24], parallelogram prediction proposed by Touma and Gotsman [47], and error encoding proposed by Li and Kuo [27]. It allows the encoding of any polygonal mesh (including non-manifolds) with no loss of connectivity information and no repetition of geometry and property data associated to singular vertices as a progressive single-resolution bitstream, and any manifold polygonal mesh in hierarchical multi-resolution mode. Extensive experimentation performed during the course of the MPEG-4 process has shown that the resulting methods are state-of-the-art.

2. Compression for Accelerated Rendering

Because polygonal faces can be triangulated, and graphics hardware is optimized for rendering triangles, most geometry compression schemes are restricted to operate on triangular meshes. Each triangle is specified for rendering by its three vertices (36 bytes), and depending on the rendering mode, also by some properties (normals, color, and/or texture coordinates), which in some cases are also bound per vertex. Because each vertex of an average triangle mesh is shared by six triangles, transmitting for each triangle all the data associated with its three vertices is wasteful.

The popular *triangle strips* and *triangle fans* of GL [22] and OpenGL [33] can be regarded as the first geometry compression scheme designed to reduce the amount of geometric data transfered from the CPU to the graphics adapter. By traversing the triangles in a different order, so that subsequent triangles share an edge, only the data associated with

© Gabriel Taubin 1999.

the opposite vertex needs to be transmitted for each new triangle. In the GL triangle strips the connectivity is specified by an *marching* bit per triangle (except for the last triangle of the strip), which determines on which of the two free edges of the current triangle the next triangle has to be attached. In the OpenGL triangle strips no connectivity information is transmitted, requiring the triangles to be attached alternating between left and right free edges. Triangle fans do not include connectivity information either, requiring the triangles to be attached always on the left free edge. Because the average triangle mesh has twice as many faces than vertices, each vertex has to be transmitted at least about twice. No existing algorithm for decomposing a triangle mesh into triangle strips or fans achieves this lower bound. For example, in the algorithm proposed by Evans et. al. [10], which is currently the best algorithm, each vertex is transmitted about 2.5 times.

Since each pair of consecutive triangles share an edge, to render triangle strips and fans a graphics pipeline needs to store two vertices and their associated properties. In the *generalized triangle mesh* representation introduced by Deering [9] the graphics pipeline has an on-line buffer where sixteen vertices and associated properties can be stored. The data stored in the buffer can be replaced, retrieved, and deleted by transmitting one *op-code* per new vertex, followed by *quantized* vertex coordinates and properties, as required by the op-code. With this approach each vertex of the original polygonal mesh is transmitted less than twice, but nevertheless more than once. Chow [7] proposes an algorithm to decompose a triangle mesh into generalized triangle meshes, missing in Deering's original work [9]. Chow's algorithm is also based on the spiraling traversal of the mesh first used by Taubin and Rossignac in the Topological Surgery compression algorithm [45]. Bar-Yehuda and Gotsman analyze the minimum buffer size required to transmit each vertex exactly once [3], and show that it is impossible not to repeat vertex data with a finite size buffer.

3. Geometry Compression

Other application areas exist where minimizing the compressed representation size to make better use of storage space and transmission bandwidth, preserving the connectivity of the original polygonal mesh, or transmitting the data progressively, are more important or required factors. In the rest of the report we concentrate on these methods, with particular emphasis on the techniques incorporated in MPEG-4. The course notes of the two courses on 3D Geometry Compression taught at Siggraph'98 [44] and Siggraph'99 [46] contain reprints of most of the original papers describing the techniques surveyed here.

4. Polygonal Meshes as MPEG-4 BIFS Nodes

The VRML standard has become a popular Internet file format to represent 3D data. A polygonal mesh with V vertices and F faces is represented in VRML by an `IndexedFaceSet` node with a `coord` array, and a `coordIndex` array as a minimum. Figure 1 shows a very simple example. The position of each vertex is described in the `coord` array by three floating-point numbers. Each face of the polygonal mesh with $n \geq 3$ *corners* is represented in the `coordIndex` array by n different indices to the `coord` array, followed by a the value `-1` as a face separator. The polygonal mesh may also have optional *properties* (normals, colors, and texture coordinates) associated to its vertices, faces, or corners, represented by corresponding arrays (`normal`, `normalIndex`, `color`, `colorIndex`, `texCoord`, `texCoordIndex`) and scalar fields (`normalPerVertex`, `colorPerVertex`).

The MPEG-4 standard uses a similar scene graph representation called Binary Encoding For Scenes (BIFS) [31, 32], which contains most VRML nodes as well as many new ones. But instead of the VRML ASCII encoding, BIFS uses a more compact binary encoding. Polygonal meshes can be

```
#VRML V2.0 utf8
Shape {
  geometry IndexedFaceSet {
    coord Coordinate {
      point [
        -1.63 -0.94 -0.66
         1.63 -0.94 -0.66
         0.00  0.00  2.00
         0.00  1.88 -0.66
      ] }
    coordIndex [
      1 2 0 -1
      3 2 1 -1
      0 2 3 -1
      0 3 1 -1
    ] } }
```

Figure 1: *A tetrahedron represented as an `IndexedFaceSet` node.*

```
#VRML V2.0 utf8
Shape {
  geometry IndexedFaceSet {
    coord Coordinate {
      point [
         0.00 -0.00  2.00
        -1.63 -0.94 -0.66
         0.00  1.88 -0.66
         1.63 -0.94 -0.66
      ] }
    coordIndex [
      0 1 3 -1
      3 1 2 -1
      3 2 0 -1
      0 2 1 -1
    ] } }
```

Figure 2: *An equivalent `IndexedFaceSet` representation for the tetrahedron of figure 1.*

© Gabriel Taubin 1999.

represented in BIFS not only by `IndexedFaceSet` nodes, but also by the new `Hierarchical3DMesh` nodes. The latter extending the `IndexedFaceSet` functionality by allowing progressive and asynchronous decoding (streaming) of the polygonal mesh data into the scene graph through a separate input stream, and browser control of level of detail, as soon as the levels of detail are made available by the decoding process. Polygonal meshes represented by `IndexedFaceSet` nodes are embedded in the stream containing the scene graph description, and so, are available for rendering only after the whole scene graph is decoded. The `Hierarchical3DMesh` functionality was modeled on the prototype VRML implementation demonstrated by Guéziec et. al. [15].

5. Connectivity-Preserving Schemes

In all the geometry compression schemes described in this report the `coordIndex` arrays of the `IndexedFaceSet` nodes given as input to the encoder and reconstructed by the decoder, are usually not the same. This is so because the order of traversal of vertices and faces has to be changed to exploit the coherence associated with closeness within the mesh, as in the methods designed for accelerated rendering described above.

But in most cases these arrays are related by a one-to-one transformation decomposable as a sequence of the following three types of basic transformations: a cyclical permutation of the corner values (vertex indices) of a face; a permutation of the F faces within the `coordIndex` array; and a permutation of the V vertex coordinate vectors within the `coord array`, with the corresponding inverse permutation applied to the face corner values. When this is the case we say that the compression scheme *preserves the connectivity* of the given polygonal mesh. These transformations, which define an equivalence relation among `IndexedFaceSet` nodes, are regarded as producing new representations of the *same* polygonal mesh. When the polygonal meshes have properties, the corresponding permutations of the property arrays must be considered as well. Figures 1 and 2 show two representation of the same polygonal mesh: a tetrahedron without properties.

6. Dual Graph Traversal

An edge of a polygonal mesh is *singular* if it is shared by three or more faces, *internal* if it is shared by exactly two faces, and *boundary* if exactly one face is incident to it. The *dual graph* of a polygonal mesh is the graph composed of the faces of the mesh as dual graph nodes, and the internal edges of the mesh as dual graph edges.

All the geometry compression schemes reorder the faces of the mesh according to an order of traversal of a maximal spanning forest in the dual graph of the mesh, so that each connected component of the mesh corresponds to one tree of the forest, and each face is visited exactly once. Since compressed representations of the different connected components are usually concatenated in the bitstream , it is sufficient to consider connected polygonal meshes. In the Topological Surgery scheme, although the choice may affect the compression ratios, any spanning tree is acceptable. A *root face* is chosen as the root of the tree, and the tree is traversed in depth-first order using the orientation of the polygonal face to decide which branch to follow first when a branching face is visited: leftmost branches are traversed first. In the modification proposed by Jang et. al. [24] to add error resiliency, the order of the branches is explicitly specified in the bitstream. Taubin and Rossignac report best compression ratios obtained by constructing the face trees spiraling around [45]. Bossen [5] experimented with several alternative construction schemes, including some which construct the trees as a function of the geometry and property prediction errors, and concluded that a hybrid method performs best. All the other single-resolution compression schemes visit the faces in a similar spiraling fashion, but in breath-first order.

As in the Deering's scheme, in all these connectivity-preserving schemes some information has to added to the bitstream to determine which vertices correspond to which previously indexed vertices, so that each vertex is transmitted exactly once. This *stitching* information is explicitly transmitted up-front in the Topological Surgery scheme as a *vertex graph*, which optionally includes information to recover non-manifold connectivity, or implicit and interleaved with the rest of the data in the other methods.

7. Encoding of Planar Graphs

Early work on encoding of planar graphs is closely related to the connectivity-preserving geometry compression schemes described in this report. In particular, the method introduced by Turan [48] is closely related to the Topological Surgery representation. Since a planar graph can be drawn on a sphere, encoding a planar graph is the same as encoding the connectivity of a polygonal mesh with sphere connectivity. Turan builds a spanning tree in the graph of V vertices and uses it to represent the boundary of a simple polygon of $2V - 2$ vertices resulting of cutting the polygonal mesh with sphere connectivity through the vertex graph edges. He also presents an encoding scheme which requires slightly less than 12 bits per vertex. The Topological Surgery representation extends Turan's construction to oriented manifold polygonal meshes of arbitrary topological type, with a significantly more efficient encoding. Itai and Rodeh [23] propose a method to encode planar graphs, closely related to Gumhold and Strasser's scheme [17], which requires no more than 4 bits per vertex. Keeler and Westbrook [25] describe a new encoding for Turan's construction which requires no more than 4.6 bits per vertex.

© Gabriel Taubin 1999.

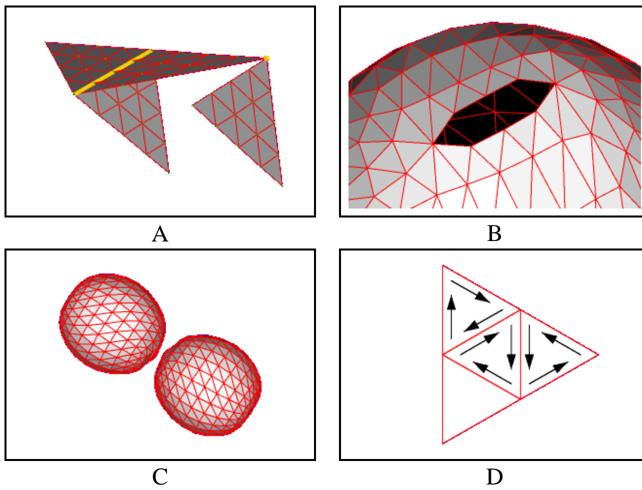


Figure 3: *(A) Non-manifold mesh, (B) mesh with boundary, (C) mesh with two connected components, and (D) orientable but not oriented mesh.*

8. Oriented Manifolds Meshes

A vertex of a polygonal mesh is a *boundary* vertex if one or more boundary edges are incident to it, otherwise it is an *internal* vertex. A boundary vertex is *regular* if it has no incident singular edge, and the set of its incident faces can be organized as a *list* such that every pair of consecutive faces in the list share exactly one regular edge. Similarly, an internal vertex is *regular* if has no incident singular edge, and the set of faces incident to it can be organized as a *cycle* such that every pair of consecutive faces in the list share exactly one regular edge. A polygonal mesh is a *manifold* if all its vertices and edges are *regular*, or *non-singular*. Figure 3 shows examples of manifold and non-manifold polygonal meshes with different properties.

Each edge of a polygonal mesh has two possible orientations corresponding to the two possible orderings of the two vertices. Each face of a polygonal mesh has two possible orientations corresponding with the two possible cyclical orderings of its corner values. Each of these two orientations induces a consistent orientation on the edges incident to the face. A manifold polygonal mesh is *orientable* if an orientation can be chosen for each of its faces such that for each internal edge, the two incident faces induce opposite orientations on the common edge. All non-manifold polygonal meshes are considered non-orientable. Polygonal meshes approximating the boundary surface of solid objects are normally orientable manifold meshes. An orientable manifold mesh is *oriented* if an orientation has been chosen for each of its faces such that for each internal edge, the two incident faces induce opposite orientations on the common edge.

When a polygonal mesh is represented as a VRML `IndexedFaceSet` node, each of its faces has an orientation. A manifold polygonal mesh so defined may be orientable but not oriented. Such a mesh can be oriented by choosing a consistent orientation for its faces (which may require to invert the orientation of one or more faces), but we regard this transformation as producing a *different* polygonal mesh.

Most geometry compression schemes are restricted to, and preserve the connectivity of, oriented manifold polygonal meshes. Determining whether a manifold polygonal mesh is orientable or not, and if so choosing a consistent orientation for its faces is an additional pre-processing step that some geometry compression scheme do perform. To let the decoder recover the original orientation of the faces, an additional bit per face must be saved in the compressed bitstream. But the methods that reorient faces to convert an orientable manifold into an oriented one usually discard these bits needed to reorient the faces back to their original orientation. We do not regard these methods as preserving the connectivity.

Although Many real-world polygonal meshes are *non-manifold* meshes, they can be compressed by converting them to oriented manifold meshes by cutting through singular vertices and edges [16] without affecting the geometry of the mesh, and then encoding the additional *stitching information* in the bitstream [13]. This procedure can also be applied to any orientable manifold polygonal mesh which is not oriented, recovering the original connectivity (including face original face orientations) after decompression. To do this it is sufficient to regard edges which are not consistently oriented by their incident faces as singular edges.

9. Topological Surgery

The Topological Surgery scheme introduced by Taubin and Rossignac [45] was the first method proposed to compress the connectivity of manifold polygonal meshes of arbitrary topological type with no loss of information, as well as their geometry and associated properties with controlled loss only due to quantization. Because this method was at the core of the VRML Compressed Binary Format proposal [43] [42], a preprint description of the method available since early 1996 had wide circulation. We explain in detail how it was implemented in MPEG-4.

As in all the other single-resolution geometry compression schemes, in the Topological Surgery representation the faces of an oriented manifold polygonal mesh are interconnected by a *face forest* spanning the dual graph of the mesh, with each tree of the face forest spanning one connected component. The edges of the mesh that do not belong to the face forest define a *vertex graph* interconnecting all the vertices of the polygonal mesh. Connected components of the vertex graph are in one-to-one correspondence with connected components of the mesh, and so, also with trees of the face forest. Figure 4 shows an example of this construction.

Note that, since the polygonal mesh is manifold, each ver-

© Gabriel Taubin 1999.

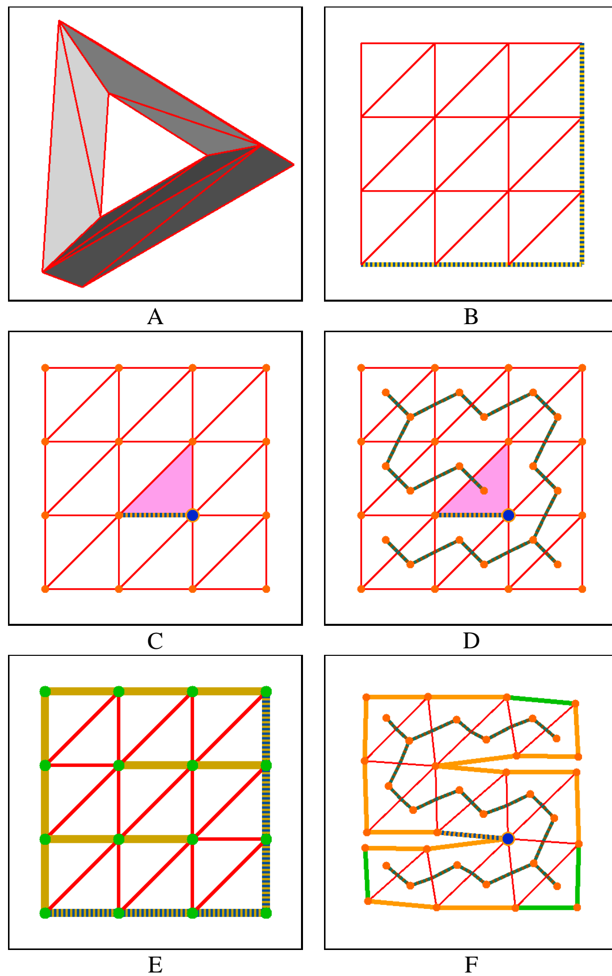


Figure 4: *Topological surgery representation. (A) A torus with 9 vertices and 18 triangular faces. (B) It can be constructed by identifying edges (stitching) along vertical and horizontal boundaries. (C) Choose a root face, root edge, and root vertex for each connected component. (D) Traverse the dual graph of the mesh constructing a spanning forest. (E) Remaining edges form the vertex graph. (F) Cut through vertex graph to create one simple polygon per connected component.*

tex graph edge is either a boundary edge of the mesh or has exactly two incident faces. If the polygonal mesh is now cut through the (internal) edges of the vertex graph, a *cut mesh* is obtained with the same number of connected components as the original polygonal mesh, and with the face forest as its dual graph. Having a tree as a dual graph, each connected component of the cut mesh has *simple polygon* connectivity, with all its vertices on its boundary and a single *boundary loop* joining all the vertices and boundary edges. Each internal edge of the polygonal mesh belonging to the vertex graph corresponds to exactly two boundary loop edges.

The encoding of this representation in the compressed data stream is composed of: the encoding of the vertex graph, the encoding of the simple polygons, and the quantized, predicted, and compressed geometry and property data. It is important to pay special attention to how each of these elements is encoded, but also to their order in the bitstream, and how they are interleaved, because all of these issues affect not only the compression efficiency, but also the decoder latency. In the MPEG-4 implementation the connectivity information is partitioned into *global information*, and *per-triangle* information. The global information is transmitted first, followed by the per-triangle information interleaved with the geometry and property data corresponding.

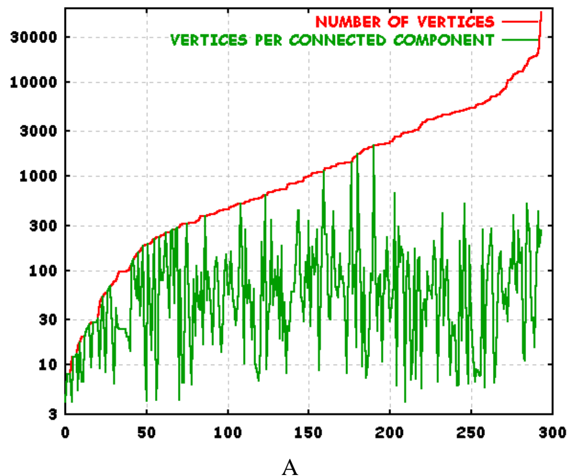
10. Topological Surgery in MPEG-4

In the original formulation [45], the compressed bitstream was composed of, in order: the encoded vertex graph; the quantized geometry and property data bound per vertex to the mesh; the simple polygons; and the property data bound per face and per corner. As a result, the decoder had to decode and store a significant proportion of the bitstream before it could made the first face available for rendering.

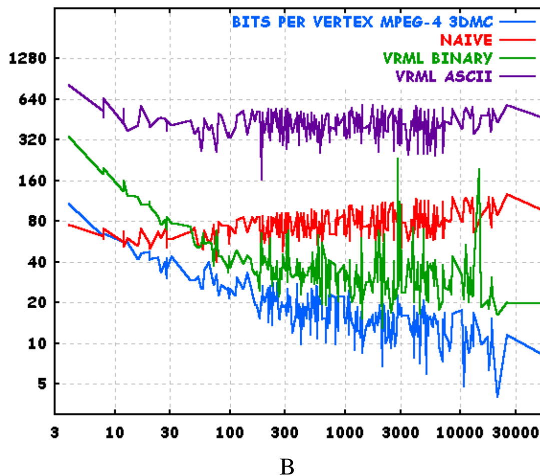
The single-resolution geometry compression scheme of MPEG-4 is also based on the Topological Surgery representation, but incorporates significant improvements in encoding efficiency, and other extensions. The improvements to the encoding of connectivity and overall bitstream organization and entropy encoding were proposed mainly by Bossen [5]. But elements of other competing schemes, such as the parallelogram prediction method used by Touma and Gotsman [47], and a variation of the progressive quantization scheme used by Li and Kuo [27], were incorporated as well. The extension proposed by Guéziec et. al. [16] allows the encoding of any polygonal mesh (including non-manifolds) with no loss of connectivity information and no repetition of geometry and property data associated to singular vertices. And the changes proposed by Jang et. al. [24] add data partitioning for error resiliency features to the encoding.

Extensive experimental data collected within the MPEG-4 core experiments process validated all the accepted modifications. Figures 5 and 6 show plots of results of some of these experiments. A database of about 300 VRML models without properties was collected from different sources in the Internet. About half of these models represent triangular meshes, and the other half represent meshes with one or more polygonal faces. In all the plots the models are sorted by total number of vertices. Figure 5-A shows the total number of vertices and the average number of vertices per connected component. Figure 6-B plots the absolute compression efficiency of the MPEG-4 Topological Surgery encoding compared to: VRML Compressed Binary Format,

© Gabriel Taubin 1999.

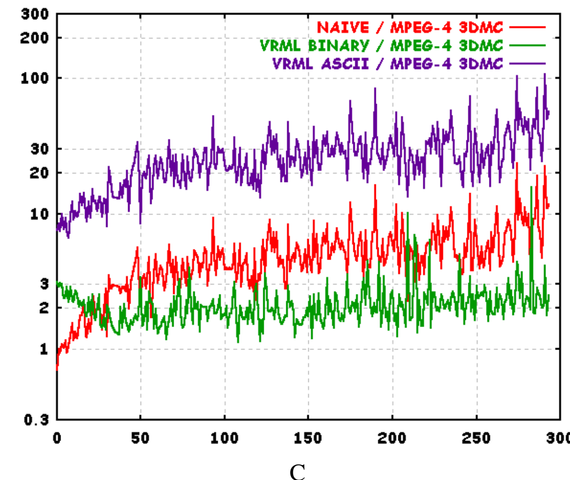


A



B

Figure 5: *MPEG-4 Results: Models sorted by total number of vertices, and average number of vertices per connected component.*



C

Figure 6: *MPEG-4 Results: Compression efficiency (bits per vertex), and relative performance.*

naive encoding, and original VRML representation (ascii). Figure 6-C is similar to Figure 6-B, but showing the relative compression ratios. In all the cases but the original VRML files, vertex coordinates are quantized to 10 bits per coordinate. The naive encoding is a straightforward encoding of the quantized `coord` and `coordIndex` arrays. Note that the MPEG-4 Topological Surgery encoding is more than twice as efficient as the VRML Compressed Binary Format encoding, and about 30 times more efficient than VRML for this family of models. The average size of the whole compressed bitstream including connectivity and geometry is 10-15 bits *per vertex*, and decreases with the size of the mesh, with large models requiring as low as 4 bits per vertex.

10.1. MPEG-4 Encoding

Figure 7 illustrates the structure of the Topological Surgery syntax in MPEG-4. After a global header information (not shown in the figure) which includes quantization parameters, the compressed connected components are concatenated. The vertex graph and triangle tree data constitute the global information each connected component. The triangle data contains not only the property data, but also a *marching bit* and a *polygon bit*, which constitute the per triangle connectivity information. The per triangle data includes an error vector to correct the the position of the opposite vertex predicted by the parallelogram rule from the previous three vertices, error vectors to correct other property predictions bound per vertex, per face, or per corner. All the different fields are arithmetic coded.

10.2. Vertex Graph Encoding

When the polygonal mesh has sphere topology, the vertex graph is in fact a vertex tree. Let us consider this case first. The vertex tree is run-length encoded. The tree is decomposed into *runs*. A run connects a leaf or branching node to another leaf or branching node through a path of zero or more regular nodes. The order of traversal of the tree, determined in the encoder process by the orientation of the mesh, defines an order of traversals of the runs, and a first and last node for each run. Each run is encoded as a record composed of three fields *(vlast,vlength,vleaf)*. The *vlast* field is a bit that determines if the runs shares the first node with the next run or not. It determines the pushing of branching node indices onto a *traversal stack*. The *vlength* field is an integer with a value equal to the number of edges in the run. The *vleaf* field is a bit which determines if the run ends in a leaf or branching node, and the popping of branching node

© Gabriel Taubin 1999.

```
base_layer() {
   do {
      connected_component()
   } while ( not(last_component))
}

connected_component() {
   vertex_graph()
   triangle_tree()
   triangle_data()
}

triangle_data() {
   root_triangle_data()
   for each other triangle {
      if(marching_triangle)
         marching_bit
      if(not(triangular_mesh))
         polygon_bit
      other_triangle_data()
   }
}
```

Figure 7: *Topological Surgery bitstream syntax in MPEG-4 (simplified).*

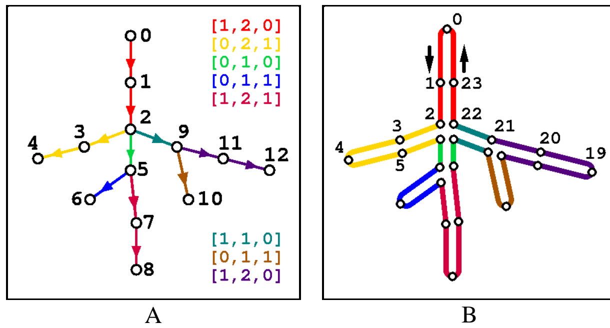


Figure 8: *Encoding the vertex tree. (A) Decompose into runs and run-length encode as table of vertex runs. (B) Each vertex tree edge corresponds to two vertex loop edges.*

indices from the traversal stack. Figure 8 shows an example of this construction, and the relation between the vertex tree and the *boundary loop* look-up table build by the decoder process to stitch the boundary edges of the simple polygon back together to reconstruct the polygonal mesh.

In general, the vertex graph is not a tree. For a connected manifold without boundary, each edge of the vertex graph corresponds to two polygon loop edges (i.e., boundary edges of the simple polygon), but for manifolds with boundary, some edges of the vertex graph (those which correspond to boundary edges of the mesh) correspond to only one polygon loop edge. As we mentioned above, in the case of a simple mesh (manifold without boundary with Euler number 2, i.e., topologically equivalent to a sphere), the vertex graph is a tree. When the manifold mesh is not simple (i.e., when the mesh has boundary edges, or when the mesh does not have sphere topology), the vertex graph contains *loops*. If the vertex tree is constructed as a spanning tree by depth first traversing the vertex graph, then for each loop, one back-edge, or *jump edge*, connects a leaf node of the vertex tree with a previously visited node, forming the loop. We represent the vertex graph as an *extended vertex tree*, as illustrated in figure 9, with each jump edge corresponding to two runs of the extended vertex tree ending in leave nodes.

To determine which edges of the vertex graph should be classified as jump edges, the compression algorithm traverses the boundary edges of the simple polygon, finds the corresponding edges in the graph, and constructs a spanning tree with a standard algorithm [38].

The *extended vertex tree* is represented by a table of runs, as in the case of a mesh with sphere topology. Figure 9 also illustrates how the decoder process builds the *boundary loop look-up table* from the data contained in this data structure.

Even though each jump corresponds to a single edge in the vertex graph, we represent a jump by two runs in the extended vertex tree table, one of them having zero length. The first, and true, run starts at the first node the edge gets visited. The second run is generated when we reach the previously visited node, which, then, needs to be considered as a branching node, and the run generated has zero length.

Each run record of the extended vertex tree table is composed of a variable number of fields. A vertex run connects a leaf or branching node to another leaf or branching node through a path of zero or more regular nodes. The order of traversal of the tree defines a sequential order for the vertex runs, and a first and last node for each run. Each vertex run is represented as a vertex run record composed of the following fields. The *vlast* field is a bit that determines if the run shares the first node with the next run or not. The *vlength* field is an integer with a value equal to the number of edges in the run, not counting the jump edges. The *vleaf* field is a bit which determines if the run ends in a leaf or branching node. Each extended vertex run ending in a leaf node (*vleaf*=1) has an additional *vjump* field, indicating if the last node of the vertex run is connected to another node through a jump edge. Note that each extended vertex run of length zero, which is always associated with a jump edge, should be considered as ending in a leaf. For these extended vertex runs, first and last node are the same. If *vjump*=1, the record also has a *jumpStart* field indicating whether this run corresponds to the start or end of the run. Each jump edge connects the last node of a run to a previously visited node. The node visited first is the start of the jump (*jumpStart*=1), and the other one is the end of the jump (*jumpStart*=0). If *jumpStart*=0, the record also has a *jumpDepth* field. Every time the start of a jump is encountered, the compression and decompression

© Gabriel Taubin 1999.

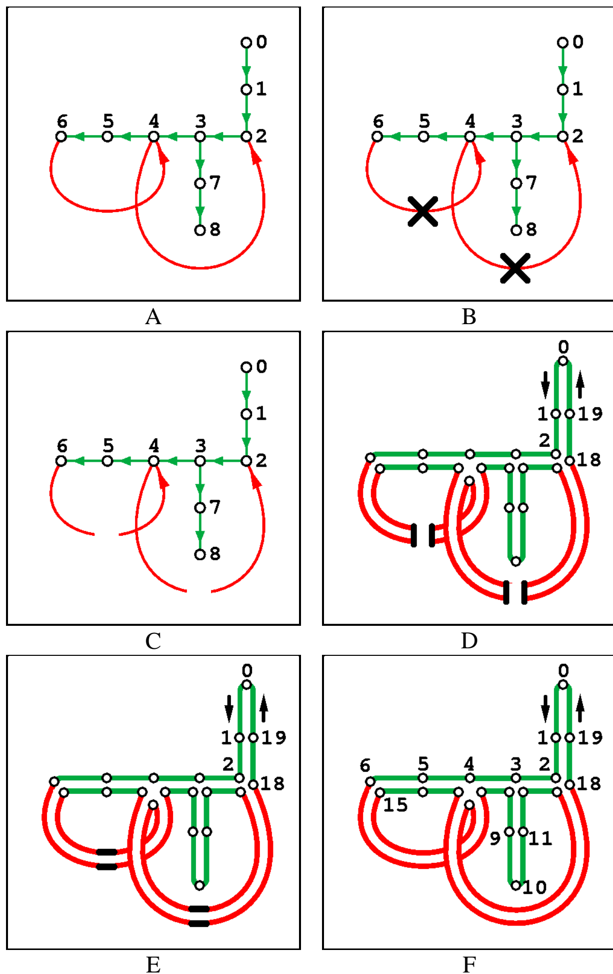


Figure 9: *Building the boundary loop. (A) Decomposed vertex graph into vertex tree (green) and jump edges (red). (B) Create extended vertex tree by cutting jump edges in half. (C) The extended vertex tree has two leaves for each jump edge. (D) Build the extended vertex tree loop. (E) Connect start and end of each jump edge. (F) Boundary loop.*

algorithms push the reference to the jump edge onto an auxiliary *jump stack*. The value of *jumpDepth* is the depth of the corresponding jump edge in the jump stack at the time the compression or decompression programs encounters the end of the jump. The reference is immediately removed from the stack, moving subsequent references up, and reducing the total depth of the stack by one (and so reducing the number of bits required to represent the depth of subsequent jump edges).

10.3. Simple Polygon Encoding

If the simple polygon is not composed of triangular faces, the first step is to triangulate the faces, creating new *virtual* internal edges. In this case one *polygon bit* per triangle is added to the bitstream to indicate which internal edges are virtual and which ones are real. Since the virtual edges will be removed by the decoding process, reconstructing the original faces, it is not necessary to take into account the potential creation of geometric artifacts.

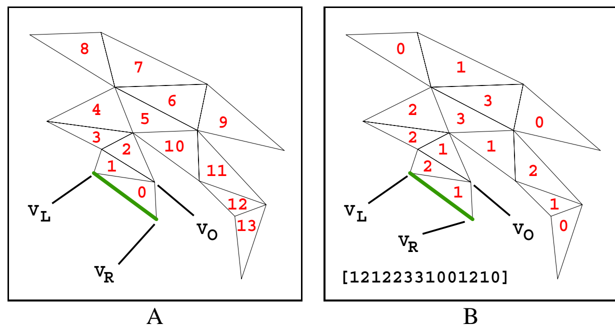


Figure 10: *Constant-length encoding of a simple polygon. (A) Triangles labels according to their order of traversal. (B) Triangles labels according to their two bit code. The encoding of the polygon is the sequence between the brackets.*

The triangulated simple polygon can be constant-length encoded with two bits per vertex (plus one polygon bit if necessary). This encoding process, illustrated in Figure 10, is performed by traversing the triangle tree. The traversal starts by entering the first triangle through the root edge, with the root vertex assigned to the *left vertex* v_L, and the second vertex assigned to the *right vertex* v_R. The third vertex of the triangle is assigned to the *opposite vertex* v_O, the edge $e_L = (v_L, v_O)$ is the *left edge*, and the edge $e_R = (v_R, v_O)$ is the *right edge*. One bit is used to indicate whether each edge (left and right) is a boundary edge or an internal edge of the polygon. If only the left edge is internal, we set $v_R = v_O$ and we continue with the other triangle incident to e_L. If only the right edge is internal, we set $v_L = v_O$ and we continue with the other triangle incident to e_R. If both edges are internal, we push v_O and v_R onto a traversal stack, we set $v_R = v_O$ and we continue with the other triangle incident to e_L. If both edges are boundary and the traversal stack is not empty, we pop v_R and v_L from the traversal stack, and we continue with the other triangle incident to the edge (v_L, v_R). If both edges are boundary, and the traversal stack is empty, we have finished visiting all the triangles of the simple polygon. For example, in Figure 10-A, the triangles are labeled with their order of traversal, and in Figure 10-B, with their corresponding two-bit code, as a number in the range $0, \ldots, 3$. Here, for each digit the first bit equals 0 if the left edge is boundary, 1 if the left edge is interior. The second bit represents the right side and uses the same convention. The encoding of this polygon is [12122331001210].

The simple polygon can also be represented as a table of triangle runs in the same way as the vertex tree, except

© Gabriel Taubin 1999.

that each triangle run record is composed of only two fields *(tlength, tleaf)*. A *tlast* field is not necessary because the triangle tree is a binary tree, and each triangle run which ends in a branching node must be followed by exactly two runs. The structure of the triangle tree does not completely describe the triangulation of the simple polygon, though. To complete the description, an extra bit per triangle associated with a regular node of the triangle tree must be included. This sequence of marching bits determines how to triangulate the triangle runs by advancing either on the left or on the right on the boundary of the simple polygon.

Since in general many more *marching triangles* (codes 1 and 2) are obtained than *leaf* (code 0) or *branching* (code 3) triangles, we run-length encode this representation. The 3s and 0s in the constant-length encoded sequence mark the end of the triangle runs. In this example, the triangle-runs are defined by the sub sequences [121223], [3], [10], [0], and [1210]. Each run is encoded as a *tlength* field, which is the number of two-bit codes in the corresponding sub sequence (6,1,2,1, and 4 in this case), and by a *tleaf* bit value determined by the last code in the sub sequence ($3 \to 0$, $0 \to 1$). The sequence of 1s and 2s in each run becomes a sequence of *marching bits* which determine how to triangulate the run by marching either on the left or on the right boundary of the simple polygon. There is one marching bit for each triangle of the run, except for the last one (which is leaf or branching). In this example the sequence of marching bits is [010110010].

10.4. Simple Polygon Decoding

The simple polygon decoding process reconstructs the triangles of each simple polygon as triplets $t = \{i, j, k\}$ of polygon boundary loop indices. These indices are subsequently replaced with the vertex indices *boundary loop look-up table* constructed during the vertex graph decoding process.

Since the order in which the polygon vertices are visited during tree traversal is usually not the sequential order of the boundary loop, the following recursive procedure is used to reconstruct the triangles of each simple polygon. As described above the traversal of a simple polygon starts by entering the first triangle crossing the root edge, with the left boundary loop index $i_L = 0$ corresponding to the root vertex, and the right boundary loop index $i_R = 1$ corresponding to the second vertex. In general, when we enter a triangle, we know the values of i_L and i_R, and only the opposite boundary loop index i_O must be determined.

If the two-bit code is 1 (leaf node with next triangle on the left), we set $i_O = i_R + 1$ (addition and subtraction is modulo the length of the polygon boundary loop) and reconstruct the triangle $\{i_L, i_O, i_R\}$, we set $i_R = i_O$, and continue. If the two-bit code is 2 (leaf node with next triangle on the right), we set $i_O = i_L - 1$, reconstruct the triangle $\{i_L, i_O, i_R\}$, we set $i_L = i_O$, and continue. To determine the value of i_O for a branching triangle, if we know the distance d along the boundary loop from the left vertex to the right vertex for the run attached to the left edge, we set $i_O = i_L + d$, reconstruct the triangle $\{i_L, i_O, i_R\}$, push i_R and i_O onto the traversal stack, set $i_R = i_O$, and continue. As explained by Taubin and Rossignac [45], these lengths can be recursively computed for all the runs from the encoding of the polygon based on the formula $d = l - 1 + d_L + d_R$, there d is the distance of one run, l is the length of the run. If the run ends in a branching node, d_L is the distance of the run attached to the left edge of the last triangle of the run, and d_R is the distance of the run attached to the right edge of the last triangle of the run. If the run ends in a leaf node, $d_L = d_R = 1$. If the two-bit code of the triangle has the value 0 (leaf node of the triangle tree), we set $i_O = i_L - 1$ or $i_O = i_R + 1$, and reconstruct the triangle $\{i_L, i_O, i_R\}$. If the stack is empty we have finished reconstructing the polygons. Otherwise we pop i_L and i_R values from the stack, and continue.

10.5. Compression of Non-Manifold Connectivity

Since non-manifold meshes can be converted to manifold by cutting through singular vertices and edges [16] without affecting the geometry, most geometry compression schemes are restricted to manifold polygonal meshes. Guéziec et. al. [13] recently proposed an extension to the Topological Surgery scheme to preserve the connectivity of any polygonal mesh. In principle, a stack-based approach with one op-code per vertex in the order of traversal, and interleaved with the vertex tree data, can be used to establish the extra stitching information generating during the conversion to manifold. But since the vertices are interconnected forming a spanning forest, the unique paths from each vertex to the root of the tree it belongs to is used here to define more compact *stitches*.

10.6. Compression of Geometry and Properties

Vertex coordinates, colors, and texture coordinates are first enclosed in a bounding box and quantized to a given number of bits per coordinate. Normals are quantized differently, as in the VRML Compressed Binary Format proposal [42]. Vertex coordinates and properties bound per vertex are predicted using the parallelogram rule, and the corresponding errors are encoded the first time the vertex is visited. Properties bound per face and per corner are predicted as a linear combination of ancestors along a tree. Per face properties use the triangle tree. The corner tree used to predict properties bound per corner is constructed by connecting the corresponding corners across the marching edges, and by connecting the opposite corner to the left corner of the triangle.

11. Multi-Resolution meshes

When the number of vertices and faces in a polygonal mesh is very large, the graphics rendering hardware may not be

© Gabriel Taubin 1999.

able to achieve frame rates high enough for interactive applications. Multi-resolution polygonal meshes, or levels of detail (LOD) hierarchies, solve the problem by trading image quality for speed. Since the time required to render a frame grows with the complexity of the scene, rendering lower resolution levels yield higher frame rates. When there is relative motion of the objects with respect to the camera, the distance between an object and the camera is used to select a sufficiently low level of detail and still maintain an interactive frame rate. When the scene becomes static, lower frame rates become acceptable and higher resolution levels are rendered yielding better image quality.

Several existing methods to generate multi-resolution polygonal meshes are based on *vertex clustering algorithms*. In one of these multi-resolution meshes, the set of vertices of each level of detail is partitioned into disjoint subsets of vertices called *clusters*. The next (lower resolution) level of detail is determined by collapsing all of the vertices in each cluster into a single vertex.

To prevent visual artifacts, sometimes referred to as *popping*, this correspondence between vertices of consecutive levels of detail can be used to animate the transition, or to *geomorph*, from one level of detail to the next by interpolating the positions of corresponding vertices as a function of time.

Some vertex clustering algorithms are based on edge collapses [21] [35] [8], and others are based on triangle collapses [18]. Because they are designed to operate on manifolds, these methods are efficient at simplifying meshes composed of a small number of large parts. Other methods based on edge collapse are constrained to maintain a constant topological type [12, 19] [40].

An early vertex clustering method due to Borrel and Rossignac [37] [4] uses the geometric proximity of vertex positions to construct clusters. This method can transform the topology (connect disconnected parts) and is particularly efficient at simplifying meshes composed of many small distinct components. The method produces non-manifold conditions (singular vertices and edges) when pairs of vertices not connected by edges are clustered. Some edge-collapse based methods also allow the identification of pairs of vertices not connected by edges by creating *virtual* edges [11].

It is desirable to transmit a multi-resolution mesh progressively from low to high resolution, so that the client could render a level of detail as soon as it is received and decoded, but before finishing receiving and decoding the whole hierarchy. This technique permits a user to interact with the mesh prior to receiving more detailed levels. Of course, it is also desirable to transmit only the differences between consecutive levels of detail, and to transmit this information in compressed form. But progressive transmission of multi-resolution polygonal mesh data with many levels of detail and high compression ratios are difficult to achieve at the same time. On one end of the scale we have methods to progressively transmit meshes with as many levels of detail as vertices [19] [34] with low compression efficiency. On the other end we have methods to compress any multi-resolution mesh produced by a vertex clustering algorithm with compression ratios comparable to the best single-resolution schemes [41], but not progressively.

Furthermore, most polygonal mesh simplification algorithms have been designed for accelerated rendering, without taking into account the constraints that progressive transmission and a compressed representation may impose.

12. Progressive Transmission

The Progressive Meshes (PM) method for progressive transmission of triangular meshes proposed by Hoppe [19] is restricted to meshes generated by edge collapses with no change of topological type. Since it requires $O(V\log_2(V))$ bits to double the size of a mesh with V vertices [40], the PM scheme is not an efficient geometry compression scheme . However, by reordering the sequence of edge collapses Hoppe can improve compression efficiency [20]. By restricting the edge collapses to form a forest in the graph of the mesh, the Progressive Forest Split (PFS) scheme of Taubin et. al. [40] requires only $O(V)$ bits to double the size of a mesh with V vertices.

The Progressive Simplicial Complexes (PSC) scheme of Popovic and Hoppe [34] is an extension of the PM scheme that transmits both the connectivity and the geometry in progressive fashion, but allowing changes in topology to occur. However, the scheme also requires $O(V\log_2(V))$ bits to double the size of the mesh and the compression algorithms are very costly, requiring several hours to compress a mesh of moderate size.

A scheme closely related to PM and PSC is the MetaStream format of Abadjev et. al. [1].

A method for representing any mesh generated by vertex clustering in progressive (but not compressed) form was introduced by Guéziec et. al. [14], together with a data structure to efficiently organize the LOD hierarchy in the client. The data structure is progressively loaded as the levels of detail are received.

A method for representing any mesh generated by vertex clustering in compressed (but not progressive) form was introduced by Taubin et. al. [41]. Compared to PSC, the compression algorithm of this method requires seconds to compress similar meshes, requires only a third of the memory, and achieves compression efficiency comparable with the most efficient single-resolution schemes. But the data is not transmitted progressively.

© Gabriel Taubin 1999.

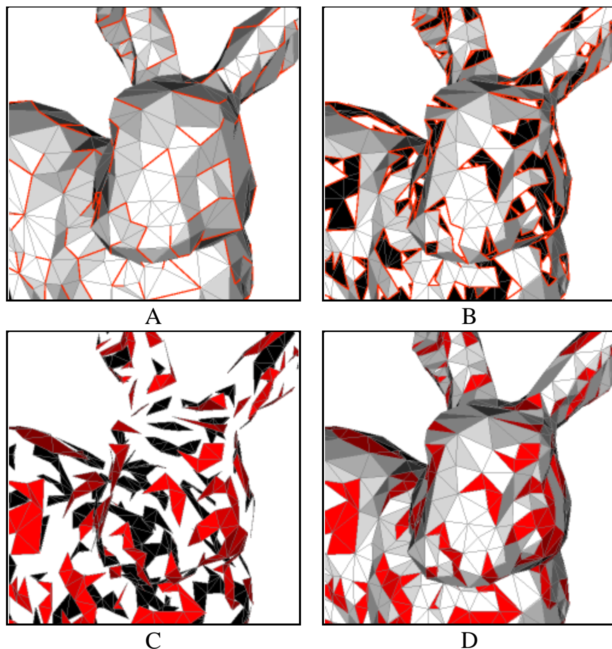


Figure 11: *The forest split operation. A: A triangular mesh with a forest of edges marked in red. B: Resulting mesh after cutting through the forest edges and splitting vertices in the resulting tree boundary loops. C: Simple polygons to be stitched to the boundary loops. The correspondence between polygon boundaries and tree boundary loops is implicit. D: The refined mesh. Normally, to produce a smooth transition, the vertices are displaced only after the boundary loops are triangulated. In the figure they have been displaced immediately after the cutting to illustrate the connectivity refinement process.*

13. Progressive Forest Split

The Progressive Forest Split (PFS) scheme [40] features an adaptive refinement scheme for storing and transmitting triangle meshes in progressive and highly compressed form. In this scheme a manifold triangular mesh is represented as a low resolution polygonal model followed by a sequence of refinement operations. The scheme permits the smooth transition between successive levels of refinement. High compression ratios are achieved by using a new refinement operation which can produce more changes per bit than existing schemes. The scheme requires only $O(V)$ bits to double the size of a mesh with V vertices.

The *forest split* operation, the refinement operation of the PFS scheme, is illustrated in Figure 11. It is specified by a forest in the graph of vertices and edges of the mesh, a sequence of simple polygons (triangulated with no internal vertices), and a sequence of vertex displacements. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting *tree boundary loops* with one of the simple polygons, and finally displacing the new vertices.

A multi-resolution mesh represented in the PFS format is composed of an initial low resolution level of detail followed by a sequence of forest split operations. The TS method is used to represent the lowest resolution level of detail because the PFS representation is a natural extension of the representation used in this scheme.

13.1. The Forest Split Operation

A forest split operation, illustrated in Figure 11, is represented by: a forest in the graph of vertices and edges of a mesh; a sequence of simple polygons; and a sequence of vertex displacements. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices.

Applying a forest split operation involves: 1) cutting the mesh through the forest edges; 2) triangulating each tree loop according to the corresponding simple polygon; and 3) displacing the new vertices to their new positions. As will be explained in the next section, some of the information required to perform these steps, such as the correspondence between trees of the forest and simple polygons, and between tree boundary loop edges and polygon boundary loop edges of each corresponding tree-polygon pair, is not given explicitly, but is based on an implicit convention for enumerating mesh elements.

13.2. Enumeration of mesh elements

Given a triangular mesh with V vertices and T triangles, we assume that the vertices have consecutive *vertex indices* in the range $0,\ldots,V-1$, and the triangles have consecutive *triangle indices* in the range $0,\ldots,T-1$. The edges of the mesh, which are represented by pairs of vertex indices (i,j) with $i<j$, are ordered lexicographically and assigned consecutive *edge indices* in the range $0,\ldots,E-1$. The trees in the forest are ordered according to the minimum vertex index of each tree. The *root vertex* v_{rt} of each tree in the forest is the leaf of the tree with the minimum index. Starting at the root, the boundary loop created by cutting along the tree can be traversed in cyclic fashion in one of the two directions. The *root edge* e_{rt} of the tree is the only edge of the tree which has the root vertex as an endpoint. Of the two triangles incident to the root edge of the tree, the *root triangle* t_{rt} of the tree is the one with the minimum triangle index. The root triangle of the tree determines the direction of traversal of the tree boundary loop. Of the two edges of the tree boundary loop corresponding to the root edge of the tree, the *root edge* e_{rt} of the tree boundary loop is the one incident to the root triangle. Figures 12-A,B illustrate these concepts.

Each simple polygon has a boundary edge identified as

© Gabriel Taubin 1999.

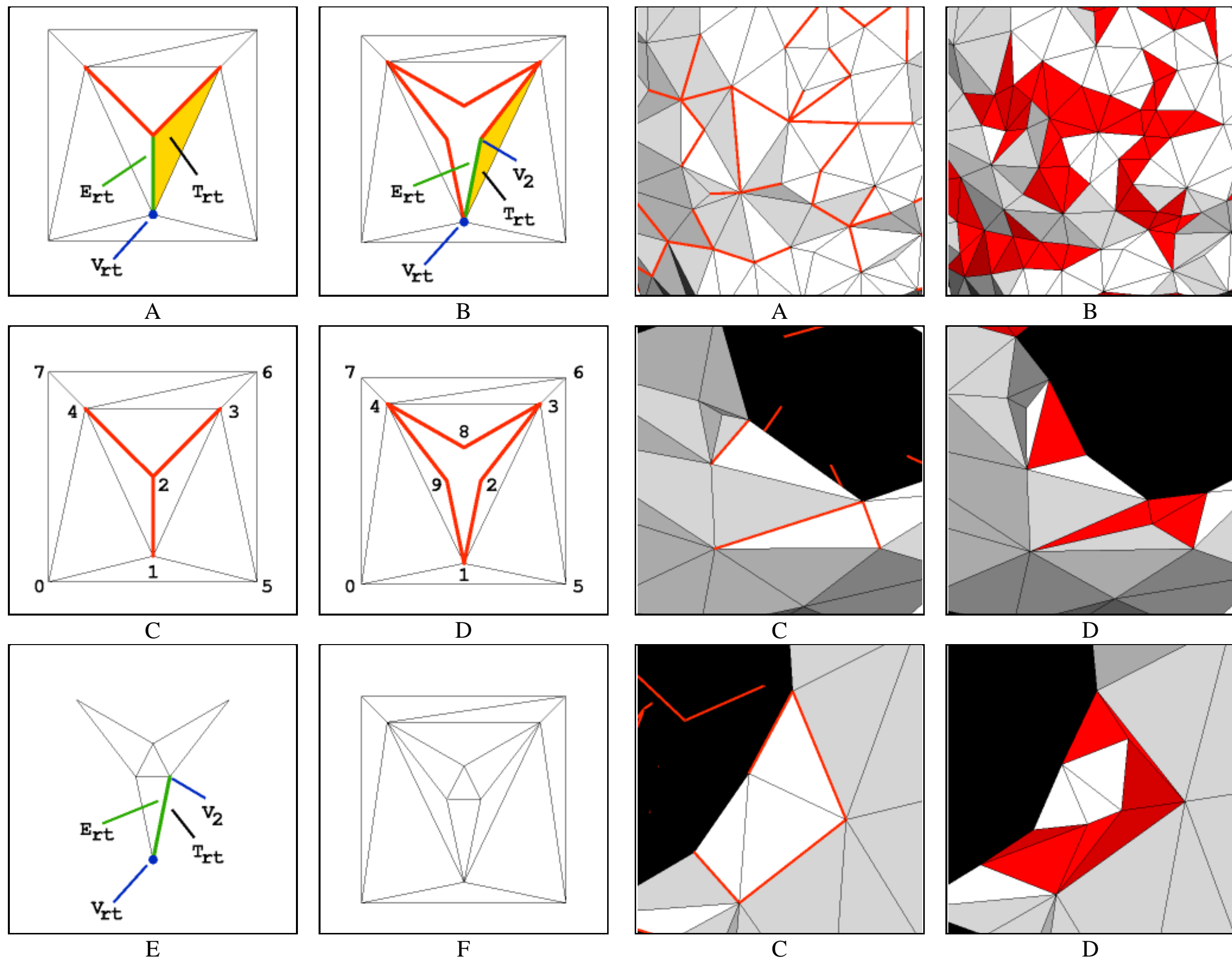


Figure 12: *When a mesh is cut through a tree of edges (red and green edges in A), a tree boundary loop (red and green edges in B) is created with each edge of the tree corresponding to two edges of the boundary loop. Some vertex indices are assigned before cutting (C) to new tree boundary loop vertices, others are assigned subsequent indices (D). The hole created by the cutting operation is filled by triangulating the boundary loop using a simple polygon (E) resulting in a refined mesh (F) with the same topological type as the initial mesh.*

Figure 13: *Construction and triangulation of tree boundary loops. A,B: No tree vertices in the mesh boundary. C,D: A tree vertex isolated in the mesh boundary requires an extra tree loop edge. E,F: A tree edge on the mesh boundary edges does not require an extra tree loop edge, but some of the new vertex indices may only be used by new triangles. Note that the tree may have several contacts with the mesh boundary.*

the *root edge* e_{rt}, with one of the two endpoints labeled as the *root vertex* v_{rt}, and the other endpoint labeled as the *second vertex* v_2. Figure 12-E illustrates these concepts. The cyclical direction of traversal of the polygon boundary loop is determined by visiting the root vertex first, followed by the second vertex. The correspondence between vertices and edges in a tree boundary loop and the polygon boundary loop is defined by their directions of cyclical traversal and by the matching of their root vertices.

13.3. Cutting through forest edges

Cutting through a forest of edges can be performed sequentially, cutting through one tree at time. Each cut is typically a local operation, affecting only the triangles incident to vertices and edges of the tree. However, as in the TS method, a single cut could involve all the triangles of the mesh. Cutting requires duplicating some tree vertices, assigning additional indices to the new vertices and fixing the specification of the affected triangles.

As illustrated in Figure 13-A,B, if no tree vertex is a boundary vertex of the mesh, then the tree is completely surrounded by triangles. Starting at the root triangle, all the cor-

© Gabriel Taubin 1999.

ners of affected triangles can be visited in the order of traversal of the tree boundary loop, by jumping from triangle to neighboring triangle, while always keeping contact with the tree. This process produces a list of triangle corners, called the *corner loop*, whose values need to be updated with the new vertex indices. While traversing this list, we encounter runs of corners corresponding to the same vertex index before the cut. A new vertex index must be assigned to each one of these runs. To prevent gaps in the list of vertex indices we first need to reuse the vertex indices of the tree vertices, which otherwise would not be corner values of any triangles. The first visited run corresponding to one of these vertices is assigned that vertex index. The next visited run corresponding to the same vertex index is assigned the first vertex index not yet assigned above the number of vertices of the mesh before the cut. This procedure performs the topological cut. For example, in Figure 12-C, the vertex index values of the corners in the corner loop list are:

$$[1233333244444421111] \ .$$

The list can be decomposed into 6 runs [11111], [2], [33333], [2], [444444], and [2]. As shown in Figure 12-D, the vertex indices assigned to these runs are 1, 2, 3, 8, 4, 9.

A tree with m edges containing no mesh boundary vertices creates a tree boundary loop of $2m$ edges. This may not be the case when one or more tree vertices are also part of the mesh boundary. As illustrated in Figures 13-C,D,E,F, several special cases, must be considered. These special cases treat collapsed edges incident to or on the mesh boundary produced by the PFS generation algorithms.

13.4. Triangulating tree boundary loops

By replacing each run of corners in the corner loop with the assigned vertex index, we construct a new list representing the tree boundary loop, If the tree boundary loop has m vertices, so does the corresponding polygon boundary loop. Each triangle $t = \{i, j, k\}$ of the simple polygon defines a new triangle of the refined mesh by replacing the polygon boundary loop indices i, j, k with their corresponding tree boundary loop indices. This is done using the list representing the tree boundary loop as a lookup table. The triangles of the simple polygon are visited in the order of a depth first traversal of its dual tree. The traversal starts with the triangle opposite to the root triangle and always traverses the left branch of a branching triangle first.

13.5. Displacing vertices

To satisfy the smooth transition property, vertex coordinates corresponding to new vertices are first assigned the same coordinates as the corresponding tree vertices before the cut. To prevent the appearance of holes, these vertices are displaced after the boundary loops are triangulated. Optionally, all affected vertices may be repositioned.

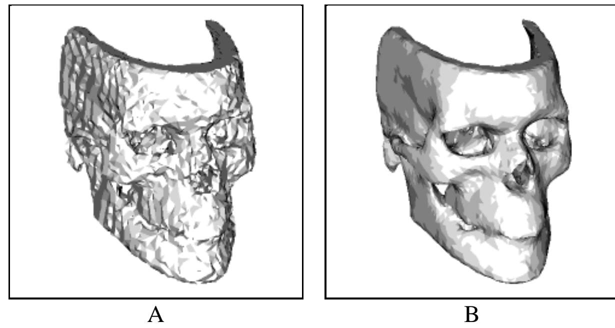


Figure 14: *Effect of post-smoothing. A: Coordinates quantized to 6 bits per coordinate. B: Result of applying the smoothing algorithm of Taubin [39] with parameters $n = 16$ $\lambda = 0.60$ $\mu = -0.64$.*

13.6. Pre and post smoothing

The differences between vertex positions before and after each forest split operation can be made smaller by representing these errors as the sum of a global predictor plus a correction. We use the smoothing method of Taubin [39] as a global predictor. The method requires only three global parameters which are included in the compressed data stream. After the connectivity refinement step of a forest split operation is applied, the new vertices are positioned where their corresponding vertices in the previous level of detail were positioned and the mesh has many edges of zero length (all the new triangles have zero surface area). The smoothing method of Taubin, which tends to equalize the size of neighboring triangles, brings the endpoints of most such edges apart, most often reducing the distance to the desired vertex positions. The corrections, the differences between the vertex positions after the split operation and the result of smoothing the positions before the split operation, are then quantized according to the global quantization grid and entropy encoded. To make sure that the resulting vertex positions have values on the quantization grid, the smoothed coordinates must be quantized before computing the corrections. In our experiments, this procedure reduces the total length of the entropy encoded corrections by up to 20-25%.

14. Progressive Forest Split in MPEG-4

In MPEG-4 the hierarchical mode representation is based on PFS. The base mesh is encoded according to the enhanced TS scheme, and followed by one or more *refinement layers*. The encoding of each refinement layer is composed of the entropy encoded sequence of forest edges, a sequence of simple polygons with no vertex coordinate data, but with per face and per corner property data associated with the new triangles, and finally displacements for the vertex coordinate and properties associated with loop vertices, and faces and corners incident to loops. Optionally, updates for the remain-

© Gabriel Taubin 1999.

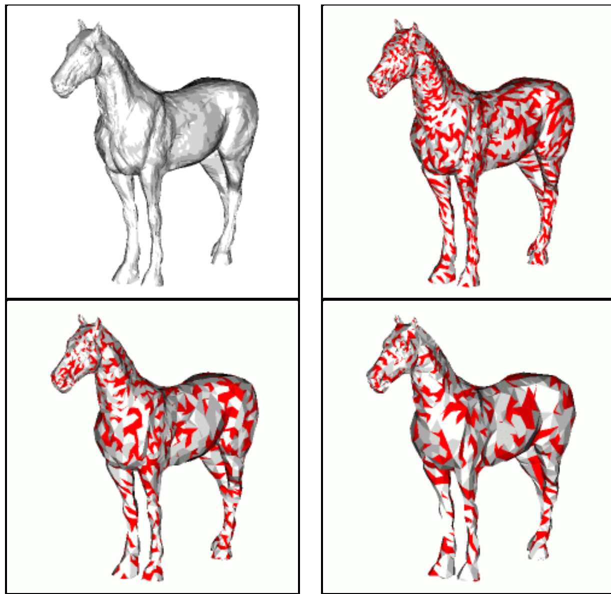


Figure 15: *Progressive forest split example.*

ing vertex coordinates and properties can also be encoded to allow for deformations to be applied.

15. Conclusions

Geometry Compression has developed very rapidly during the last four years. Starting from methods to reduce the amount of data transmitted from the CPU to the graphics adapter, the technology evolved into sophisticated schemes to transmit compressed mesh data efficiently and progressively. Some of these methods are even considered mature enough for standardization. But there is still work to be done. The limits of the current technology have not been reached. Because the associated optimization problems are too hard to attack in any other way, all the existing compression algorithm are based on heuristics. What is the *minimum* number of bits necessary to encode a mesh ? Efficient multi-resolution compression schemes with flexible and controlled changes in topology are still missing. In my view, the next challenge is to add a new dimension to the problem: animation, particularly when the topology needs to change.

References

1. V. Abadjev, M. del Rosario, A. Lebedev, A. Migdal, and V. Paskhaver. Metastream. In *VRML'99 Conference Proceedings*, 1999.

2. C. Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *IEEE Data Compression Conference Conference Proceedings*, May 1999.

3. R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, April 1996.

4. P. Borrel and J. Rossignac. Multi-resolution graphic representation eploying at least one simplified model for interactive visualization applications, September 1995. US Patent Number 5,448,686.

5. F. Bossen. *On The Art Of Compressing Three-Dimensional Polygonal Meshes And Their Associated Properties*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), June 1999.

6. L. Chiariglione. Mpeg home page. `http://www.cselt.it/mpeg`.

7. M.M. Chow. Optimized geometry compression for real-time rendering. In *IEEE Visualization'97 Conference Proceedings*, pages 347–354, 1997.

8. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Siggraph'96 Conference Proceedings*, pages 119–128, August 1996.

9. M. Deering. Geometric Compression. In *Siggraph'95 Conference Proceedings*, pages 13–20, August 1995.

10. F. Evans, S.S. Skiena, and A. Varshney. Optimized triangle strips for fast rendering. In *Proceedings, IEEE Conference on Visualization'96*, 1996.

11. M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Siggraph'97 Conference Proceedings*, pages 209–216, August 1997.

12. A. Guéziec. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, Baltimore, MD, November 1995.

13. A. Guéziec, G. Bossen, F. abd Taubin, and C. Silva. Efficient compression of non-manifold meshes. In *IEEE Visualization'99 Conference Proceedings*, October 1999. (to appear).

14. A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *VRML 98*. ACM, February 1998.

15. A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. A framework for streaming geometry in vrml. *IEEE Computer Graphics and Applications*, 19(2):68–78, March-April 1999.

16. A. Guéziec, G. Taubin, F. Lazarus, and W.P. Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. In *IEEE Visualization'98 Conference Proceedings*, pages 383–390, October 1998.

17. S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *Siggraph'98 Conference Proceedings*, pages 133–140, July 1998.

18. B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

19. H. Hoppe. Progressive meshes. In *Siggraph'96 Conference Proceedings*, pages 99–108, August 1996.

© Gabriel Taubin 1999.

20. H. Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 1998.

21. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Siggraph'93 Conference Proceedings*, pages 19–25, July 1993.

22. Silicon Graphics Inc. Gl programming guide, 1991.

23. A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, (17):215–219, 1982.

24. E.S. Jang, S.J. Kim, M. Song, M. Han, S.Y. Jung, and Y.S. Seo. Results of ce m5 error resilient 3d mesh coding, December 1998. ISO/IEC JTC 1/SC 29/WG 11 Input Document No. M4251.

25. K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. *Discrete and Applied Mathematics*, (58):239–252, 1995.

26. R. Koenen. Mpeg-4 : Multimedia for our time. *IEEE Spectrum*, 36(2):26–33, February 1999.

27. J. Li and C.C. Kuo. Progressive Coding of 3D Graphics Models. *Proceedings of the IEEE*, 86(6):1052–1063, June 1998.

28. Mpeg-4 applications (seoul revision), March 1999. ISO/IEC JTC1/SC29/WG11 Document No. W2724.

29. Mpeg-4 overview (seoul revision), March 1999. ISO/IEC JTC1/SC29/WG11 Document No. W2725.

30. Mpeg-4 requirements (seoul revision), March 1999. ISO/IEC JTC1/SC29/WG11 Document No. W2723.

31. ISO/IEC 14496-1 Information technology - Coding of audio-visual objects, Part 1: Systems (MPEG-4 v.1) , December 1998. ISO/IEC JTC 1/SC 29/WG 11 Document No. W2501.

32. ISO/IEC 14496-1 Information technology - Coding of audio-visual objects, Part 1: Systems / PDAM1 (MPEG-4 v.2), March 1999. ISO/IEC JTC 1/SC 29/WG 11 Document No. W2739.

33. J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison-Wesley, MA, USA, 1997.

34. J. Popović and H. Hoppe. Progressive simplicial complexes. In *Siggraph'97 Conference Proceedings*, pages 217–224, August 1997.

35. R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3), 1996. Proc. Eurographics'96.

36. J. Rossignac. Edgebreaker: Connectivity compression for triangular meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January-March 1999.

37. J. Rossignac and P. Borrel. *Geometric Modeling in Computer Graphics*, chapter Multi-resolution 3D approximations for rendering complex scenes, pages 455–465. Springer Verlag, 1993.

38. R.E. Tarjan. *Data Structures and Network Algorithms*. Number 44 in CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1983.

39. G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.

40. G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Siggraph'98 Conference Proceedings*, pages 123–132, July 1998.

41. G. Taubin, W. Horn, and P. Borrel. Compression and transmission of multi-resolution clustered meshes. Technical Report RC-21398, IBM Research, February 1999.

42. G. Taubin, W.P. Horn, and F. Lazarus. The VRML Compressed Binary Format, June 1997. `http://www.research.ibm.com/vrml/binary`.

43. G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac. Geometric Coding and VRML. *Proceedings of the IEEE*, 86(6):1228–1243, June 1998.

44. G. Taubin and J. Rossignac, editors. *3D Geometry Compression*, Siggraph'98 Course Notes 21, July 1998.

45. G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

46. G. Taubin and J. Rossignac, editors. *3D Geometry Compression*, Siggraph'99 Course Notes 22, August 1999.

47. C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface Conference Proceedings*, Vancouver, June 1998.

48. G. Turán. On the succint representation of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.

49. The Virtual Reality Modeling Language. `http://www.web3d.org`, September 1997. ISO/IEC 14772-1.

© Gabriel Taubin 1999.