UNC Technical Report TR-00-04

Face Fixer: Compressing Polygon Meshes with Properties

Martin Isenburg* Jack Snoeyink†

University of North Carolina at Chapel Hill

Abstract

Most schemes to compress the topology of a surface mesh have been developed for the lowest common denominator: triangulated meshes. We propose a scheme that handles the topology of arbitrary polygon meshes. It encodes meshes directly in their polygonal representation and extends to capture face groupings in a natural way. Avoiding the triangulation step we reduce the storage costs for typical polygon models that have group structures and property data.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surface, solid, and object representations;

Additional Keywords: Mesh compression, connectivity encoding

1 INTRODUCTION

Because bandwidth to the graphics pipeline is a limiting factor in a number of graphics applications, compression schemes for geometric data sets have recently been the subject of intense study. In particular, many efficient techniques have been proposed for encoding polygonal meshes [3, 24, 25, 7, 16, 18, 13, 15]; we survey these in the next section. Generally mesh compression techniques focus on encoding fully triangulated data sets—a natural candidate for the lowest common denominator. Triangle meshes are easily derived from other surface representations and are widely supported by today's graphics hardware. Especially for data sets whose only destination is the trip down the rendering pipeline, a compact triangle-based representation is a good choice.

However, many models are represented by polygonal meshes that contain a surprisingly small percentage of triangles. Two examples are the standard 'triceratops' and 'galleon' models shown in Figure 1, which are initially not triangulated. The 'Premier Collection' from Viewpoint Datalabs [28]—a well-known source of high quality 3D models—consists mostly of meshes with very low triangle counts. Likewise, few triangles are found in the output formats of many computer aided design (CAD) packages. The dominating element of these models is the quadrangle or quadrilateral, but pentagons, hexagons and higher degree faces are also common.

Especially for storage purposes it is beneficial to keep a mesh in its native polygonal representation and delay the conversion to triangles until this becomes necessary. King et al. [13] have shown that the connectivity information of meshes mostly composed of quadrangles can be represented with fewer bits than that of their triangulated counterparts. Furthermore, most meshes have associated properties such as normal, colour or texture information that account for a large portion of the storage costs. Triangulating a polygon mesh not only adds an extra processing step, but also increases the number of faces and corners—replicating their associated properties.

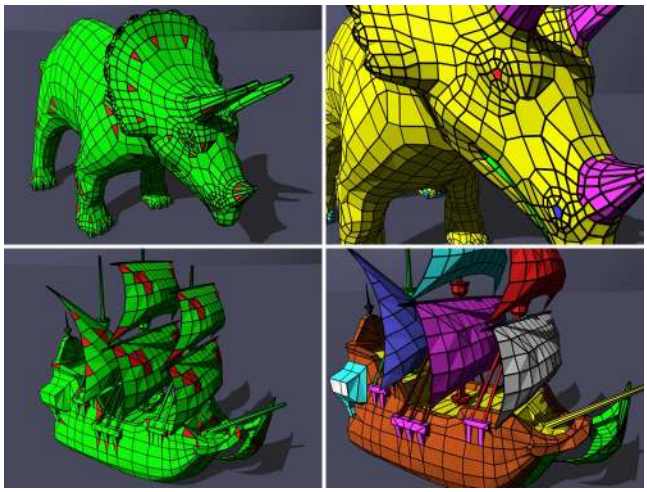

Figure 1: The triceratops and the galleon model contain only a small number of triangles shown in red (left). The group structures on these meshes are illustrated with arbitrary colours (right).

Reconstructing the original polygon mesh rather than a triangulated version can also lead to better results for subsequent triangle strip generation. Stripification algorithms, such as STRIPE [5], exploit the freedom to triangulate polygons on demand to generate triangle strips that use a minimal number of swaps and restarts.

Often polygon models also contain structural information that classifies groups of faces into logical units. The triceratops and the galleon model in Figure 1 both contain such group information. Such a structure can establish a mapping between meaningful parts of the real-world object and the faces in the model that represent them. This information can also be used to attach material properties to groups of faces. Encoding such group structures has not been addressed by previously reported compression schemes.

We propose a simple scheme for encoding the connectivity of a polygon mesh that is based on assigning a code to each mesh edge. Section 3 describes our scheme as it applies to polygon meshes with holes and handles, and extends it in Section 4 to efficiently associate property data and encode group structures. We report compression rates for a number of meshes that have been used in the literature.

2 PREVIOUS WORK

After reviewing the problem of encoding polygon meshes and the approaches typically taken, we give a detailed survey of the recent literature on connectivity compression. However, we limit this description to the case of simple meshes. For details on how these schemes encode meshes with boundary, with holes, or with handles, we refer the reader to the original reference.

*isenburg@cs.unc.edu http://www.cs.unc.edu/~isenburg/facefixer

†snoeyink@cs.unc.edu

2.1 Preliminaries

A *polygon mesh* is a collection of polygonal *faces* that intersect only along shared edges and vertices. Any edge is shared by at most two faces; unshared edges are *boundary edges*. Around each face we find a cycle of vertices and edges; around each vertex we find a cycle of edges and faces. Each appearance of a face in a vertex list or of a vertex in a face list is called a *corner*. In the mesh compression literature, a distinction is often made between three things: mesh *geometry*, which includes vertex coordinates, mesh *properties* such as normals, colours, and texture coordinates that are attached to vertices, faces, or corners, and mesh *connectivity*, which describes the incidences between vertices, edges, and faces. The mesh connectivity information is also referred to as mesh topology.

Topologically, a mesh is a graph embedded in a 2-manifold surface in which each point has a neighborhood that is homeomorphic to a disk or a half-disk. Points with half-disk neighborhoods are on the boundary. A mesh has genus g if one can remove up to g closed loops without disconnecting the underlying surface; such a surface is topologically equivalent to a sphere with g handles. A mesh is *simple* if it has no handles ($g = 0$) and no boundary edges. Euler's relation says that a graph embedded on a sphere having f faces, e edges, and v vertices satisfies $f - e + v = 2$. When all faces have at least three sides, we know that $f \leq 2v - 4$ and $e \leq 3v - 6$, with equality if an only if all faces are triangles. For a mesh with g handles (genus g) the relation becomes $f - e + v = 2 - 2g$ and the bounds on faces and edges increase correspondingly.

The standard representation for uncompressed polygon meshes uses a list of vertex coordinates to store geometry and a list of vertex indices for each face to store mesh connectivity. For triangle meshes of v vertices, this requires approximately $6v \log_2 v$ bits for the mesh connectivity. Note that this representation does not directly store face adjacency, which must be recovered by sorting around vertices if the mesh is to be checked for cracks or turned into triangle strips.

However, mesh connectivity can be encoded in a constant number of bits per vertex, while geometry and property data can be efficiently compressed with schemes that predict a position or a feature from previously decoded neighbours. Researchers in mesh compression have aimed for three different objectives: efficient rendering, progressive transmission, and maximum compression.

Efficient rendering: Encodings for rendering use partial information about mesh connectivity to reduce the work in the graphics pipeline. In the standard representation, each triangle of the mesh must be rendered individually by sending its three vertices to the graphics hardware. On average, every mesh vertex is processed six times. Processing a vertex involves passing its coordinates from the memory to and through the graphics pipeline. Typically, this also includes normal, colour, and texture information. The most common technique to reduce the number of times this data needs to be transmitted is to send long runs of adjacent triangles. Such *triangle strips* [5, 30] are widely supported by today's graphics hardware. Two vertices from a previous triangle are re-used for all but the first triangle of every strip. Depending on the quality of the strips, this can reduce the number of vertex repetitions by a factor of three.

In addition to specifying quantizations and codings for coordinates, normals, colors, and other mesh properties, Deering's pioneering paper [3] introduced a technique to further reduce the number of vertex repetitions. His *generalized triangle mesh* is designed for a geometry engine that can cache up to sixteen of the vertices that have previously passed through the transformation pipeline.

Progressive transmission: Encodings for progressive transmission use incremental refinements of mesh connectivity and geometry so that partial data already represents the entire mesh at a lower resolution. Hoppe's Progressive Mesh scheme [8] encodes a mesh by collapsing edges one by one. Decoding starts with a small base mesh and expands the collapsed edges in reverse order.

While the first progressive schemes were not designed for compression and used a large number of bits per vertex, recent schemes [22, 17, 1, 2] group the refinement operations into large batches and achieve bit-rates that come close to those of non-progressive methods. Even though more bits are used for the connectivity information, the progressive nature of the decoding allows more accurate geometry and property prediction.

For the special case of terrains models based on Delaunay triangulations, Snoeyink and van Kreveld [20] used ideas from Kirkpatrick's point location scheme [14] to encode all topology information in a permutation of the vertices, from which the mesh is progressively reconstructed. Denny and Sohler's work [4] extended this scheme to arbitrary *planar* triangulations. Although the cost of storing the topology is zero, the unstructured order in which the vertices are received and the absence of adjacency information during their decompression prohibits predictive geometry encoding. This makes these schemes overall more expensive. Moreover, it is not clear that it is possible to extend this idea to general surface meshes.

Maximum compression: Most schemes for maximum mesh compression encode mesh connectivity through a compact and often interwoven representation of two dual spanning trees: one tree spans the vertices, and its dual spans the triangles. Neither the triangle nor the vertex tree is sufficient by itself to capture the connectivity information. Typically such compression schemes [24, 25, 7, 16, 9, 18] use a pair of spanning trees obtained by traversing the vertices and the triangles of the mesh with a deterministic strategy (e.g. breadth or depth first search). The geometry data and the property data of the mesh are usually compressed using predictive encoding based on local neighbourhood information [24, 25].

2.2 Connectivity Compression Techniques

One of the nicest proofs of Euler's relation for planar graphs partitions the edges into two spanning trees [21]. One tree, spanning the vertices, has $v - 1$ edges and the other, spanning the faces, has $f - 1$ edges, so $e = (v - 1) + (f - 1)$. Turan [26] was the first to observe that this partition into two spanning trees could be used to encode planar graphs. He gave an encoding that used 12 bits per vertex (bpv). Keeler and Westbrook [11] improved Turan's method to guarantee 9 bpv for encoding planar graphs and 4.6 bpv for simple triangle meshes, but reported no extension for general meshes.

Taubin and Rossignac proposed a scheme that explicitly encodes both spanning trees. Their Topological Surgery method [24] cuts a mesh along a set of edges that corresponds to a spanning tree of vertices. This produces a simple mesh without internal vertices that can be represented by the dual triangle spanning tree. Run-length encoding both trees results in practice in bit-rates of around 4 bpv.

Touma and Gotsman's Triangle Mesh Compression [25] encodes the degree of each vertex along a spiraling vertex tree with an "add <degree>" code. For each branch in the tree they need an additional "split <offset>" code that specifies the start and the length of the branch. This technique implicitly encodes the triangle spanning tree. They compress the resulting sequence of "add" and "split" commands using a combination of run-length and entropy encoding. Especially for regular meshes they achieve lower bit-rates than other schemes. Results on standard meshes range from 0.2 to 3.0 bpv.

Gumhold and Strasser [7] introduce a compressed representation for triangle meshes that is similar to the Edgebreaker method [18]. Starting with the three edges of an arbitrary triangle as what they call the initial "cut-border," they traverse the triangles of the mesh and include them into this boundary loop using three connect and one split operation. The offset value associated with the split operation is used to re-play the split operation during decoding. This makes it possible to decode the mesh connectivity in a single forward traversal of all operations, which allows encoding and decoding to run in parallel—with a minimal delay of one operation.

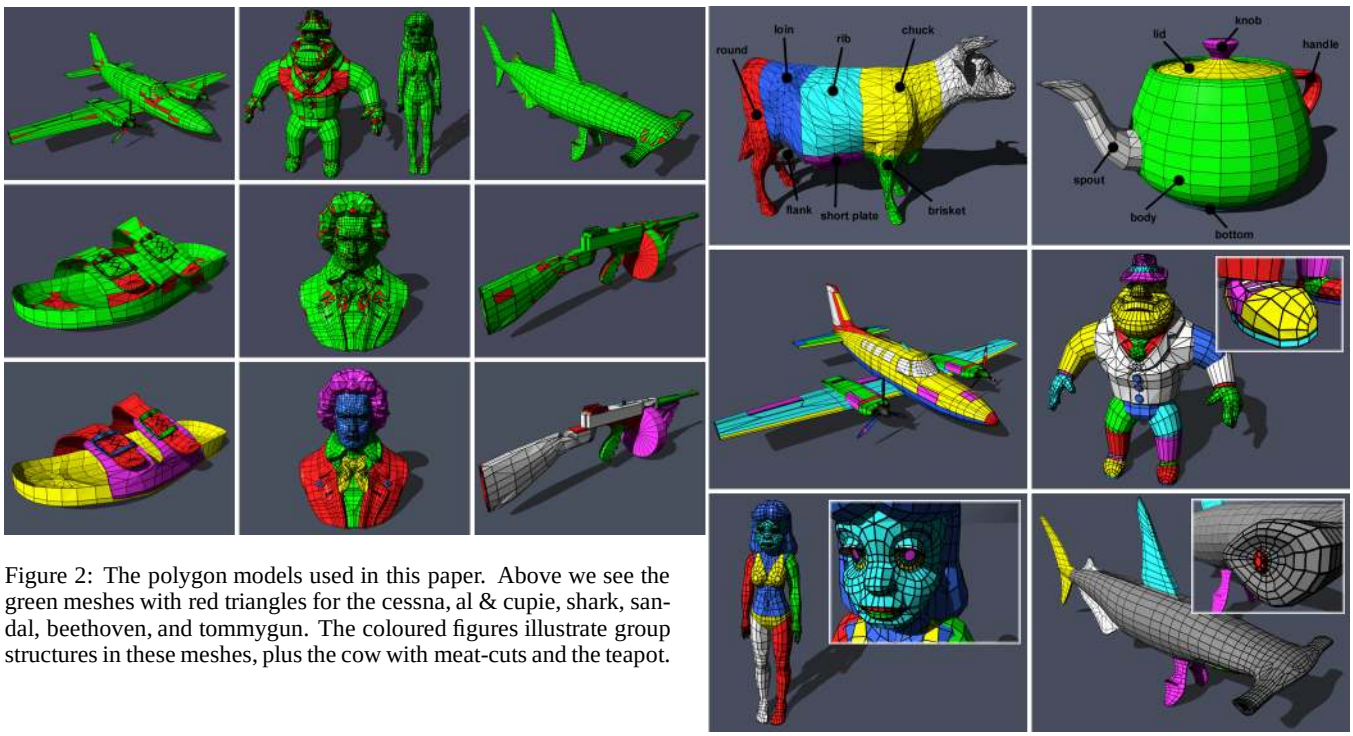

Figure 2: The polygon models used in this paper. Above we see the green meshes with red triangles for the cessna, al & cupie, shark, sandal, beethoven, and tommygun. The coloured figures illustrate group structures in these meshes, plus the cow with meat-cuts and the teapot.

name	groups	parts	vertices	faces		corners		△	▱	⬠	⬡	>⬡	holes	hndls	name	bpv
triceratops	6	1	2832	2834	(5660)	11328	(16980)	346	2266	140	63	19	–	–	triceratops	**2.115**
galleon	17	12	2372	2384	(4698)	9466	(14094)	336	1947	40	18	43	–	–	galleon	**2.595**
cessna	38	11	3745	3927	(7446)	15300	(22338)	900	2797	180	27	23	–	–	cessna	**2.841**
beethoven	10	8	2655	2812	(5030)	10654	(15090)	680	2078	44	4	6	10	–	beethoven	**2.890**
sandal	5	9	2636	2953	(4952)	10858	(14856)	961	1985	7	–	–	14	12	sandal	**2.602**
shark	7	1	2560	2562	(5116)	10240	(15348)	188	2253	83	29	9	–	–	shark	**1.670**
al	35	21	3618	4175	(7152)	15502	(21456)	1579	2505	44	11	36	–	–	al	**2.926**
cupie	15	6	2984	3032	(5944)	12008	(17832)	384	2506	114	10	18	–	–	cupie	**2.307**
tommygun	15	39	4171	3980	(8210)	16170	(24630)	992	2785	84	21	98	–	6	tommygun	**2.611**
cow	8	1	2904	5804	(5804)	17412	(17412)	5804	–	–	–	–	–	–	cow	**2.213**
teapot	6	1	1189	1290	(2378)	4958	(7134)	215	1070	3	1	1	–	1	teapot	**1.669**

Table 1: The statistics of mesh topology and polygon types for all example models together with the achieved connectivity compression in bits per vertex (bpv). The numbers in brackets give the face and corner counts for the triangulated version of each model.

Rossignac's Edgebreaker scheme [18] gives the best guaranteed bit-rates for triangle mesh connectivity. The compression algorithm uses five operations, C, L, E, R and S, to include triangles into a boundary, which is initially defined around an arbitrary triangle. The operations S and E replace the split operation used by the "cut-border machine" [7], thereby eliminating the need for explicitly encoding the associated offset value. Improvements on the original paper give linear decoding time [19, 10] and tighten the guaranteed bit-rate to 3.67 bpv [12]. This is currently the lowest worst-case bound and lies within 13% of the theoretical lower limit by Tutte [27].

All of the above schemes have been designed to compress the connectivity of purely triangular meshes. However, several authors have reported extensions to their schemes in order to handle polygonal input. A naive approach arbitrarily triangulates the polygon mesh and then uses one bit per edge to distinguish the original edges from those added during the triangulation process. Marking every edge can be avoided by triangulating the polygons systematically.

For the Topological Surgery method [23] the extension to polygonal meshes first cuts the mesh along a vertex spanning tree and then triangulates the dual polygon spanning tree. Only the edges interior to the resulting triangle spanning tree need to be marked. Similarly King et al. [13] describe how to let the Edgebreaker method guide the triangulation process. For simple polygon meshes without vertices of degree two they give an encoding that guarantees 5 bpv.

In fact, King et al. [13] are first to prove that quadrangular meshes can be compressed more efficient than their triangulated counterparts by avoiding the triangulation step. They give compact encodings for pure quadrangular meshes and for meshes containing mostly quadrangles and a few triangles. Similar results are reported by Kronrod and Gotsman [15]. Both papers suggest extensions to arbitrary polygons, but no experimental results are given.

We recently learned about the Dual Graph approach by Li et al. [16], which has similarities to the method presented here. Their scheme traverses the edges of the dual and records a stream of symbols and integer values, which is compressed with a carefully designed context based entropy coder. Decoding uses the recorded information to re-play this traversal, thereby reconstructing the mesh connectivity. Their use of split offsets resembles that of the "cut-border machine" [7], which is avoided by our method.

Inspired by Edgebreaker [18], we propose an edge-based compression scheme that encodes the connectivity of 2-manifold polygon meshes and extends to capture structural information as well.

3 FACE FIXER

The connectivity of the polygon mesh is encoded as a sequence of labels F_n, R, L, S, E, H_n, and $M_{i,k,l}$. The total number of labels equals the number of mesh edges. The sequence of labels represents an interwoven description of a polygon spanning tree and its complementary vertex spanning tree. For every face of n sides there is a label F_n and for every hole of size n there is a label H_n. Together they label the edges of the polygon spanning tree. For every handle there is a label $M_{i,k,l}$ that has three integer values associated. These specify the two edges of the polygon spanning tree that need to be 'fixed' together to re-create the handle. The remaining labels R, L, S, and E label the edges of the corresponding vertex spanning tree and describe how to 'fix' faces and holes together. Subsequently an entropy coder compresses the label sequence into a bit-stream.

3.1 Encoding and Decoding

Starting with a polygon mesh of v vertices, e edges, f faces, h holes, and g handles, the encoding process produces a sequence of e labels. This sequence contains f labels of type F_n, h labels of type H_n, g labels of type $M_{i,k,l}$, and $v-2+g$ labels of type R, L, S, or E. The connectivity of the polygon mesh can be reconstructed with a single reverse traversal of the label sequence.

The algorithm maintains one or more loops of edges that separate a single processed region of the mesh from the rest. Each of these *boundary loops* has a distinguished *gate* edge. The focus of the algorithm is on the *active boundary*; all others are temporarily buffered in a stack. The initial active boundary, defined clockwise around an arbitrary edge of the mesh, has two *boundary edges*. The gate of the active boundary is the *active gate*.

In every step of the encoding process the active gate is labeled with either F_n, R, L, S, E, H_n, or $M_{i,k,l}$. Which label the active gate is given depends on its adjacency relation to the boundary. After recording the label, the boundary is updated and a new active gate is selected. Depending on the label, the boundary expands (F_n and H_n), shrinks (R and L), splits (S), ends (E), or merges ($M_{i,k,l}$). Table 2 summarizes the changes to the processed region and its boundaries for each operation. The encoding process terminates after exactly e iterations, where e is the number of mesh edges.

	change to # of processed					# of boundary	
label	faces	holes	vertices	edges	handles	loops	edges
F_n	$+1$	$\cdots$	$\cdots$	$+1$	$\cdots$	$\cdots$	$+(n-2)$
H_n	$\cdots$	$+1$	$\cdots$	$+1$	$\cdots$	$\cdots$	$+(n-2)$
R, L	$\cdots$	$\cdots$	$+1$	$+1$	$\cdots$	$\cdots$	-2
S	$\cdots$	$\cdots$	$\cdots$	$+1$	$\cdots$	$+1$	-2
E	$\cdots$	$\cdots$	$+2$	$+1$	$\cdots$	-1	-2
$M_{i,k,l}$	$\cdots$	$\cdots$	$\cdots$	$+1$	$+1$	-1	-2
init	$=0$	$=0$	$=0$	$=0$	$=0$	$=1$	$=2$
final	$=f$	$=h$	$=v$	$=e$	$=g$	$=0$	$=0$

Table 2: The changes for each label in number of processed faces, holes, vertices, edges, handles, boundary components, and boundary edges. Initial and final counts are listed at the bottom.

In Figure 3 we illustrate for each label the situation in which it applies and the respective updates for gate and boundary. Both encoding and decoding are shown. The details for encoding are:

label F_n The active gate is not adjacent to any other boundary edge, but to an unprocessed face of degree n. The active boundary is extended around this face. The new active gate is the rightmost edge of the included face.

label R The active gate is adjacent to the next edge along the active boundary. The gate is 'fixed' together with this edge. The new active gate is the previous edge along the active boundary.

label L The active gate is adjacent to the previous edge along the active boundary. The gate is 'fixed' together with this edge. The new active gate is the next edge along the active boundary.

label S The active gate is adjacent to an edge of the active boundary which is neither the next nor the previous. The gate is 'fixed' together with this edge, which splits the active boundary. The previous edge and the next edge along the active boundary become gates for the two resulting boundaries. One is pushed on the stack and encoding continues on the other.

label E The active gate is adjacent to an edge of the active boundary which is both the next and the previous. Then the active boundary consists of only two edges which are 'fixed' together. The encoding process terminates if the boundary stack is empty. Otherwise it continues on the boundary popped from this stack.

label H_n The active gate is not adjacent to any other boundary edge, but to an unprocessed hole of size n. The active boundary is extended around this hole. The new active gate is the rightmost edge of the included hole.

label $M_{i,l,k}$ The active gate is adjacent to a boundary edge which is not from the active boundary, but from a boundary in the stack. 'Fixing' the two edges together merges the two boundaries. Consequently this boundary is removed from the stack. Its former position i in the stack and two offset values l and k (see Figure 3) are stored together with the label. The new active gate is the previous edge along the boundary from the stack.

We use a simple half-edge structure [6] during encoding and decoding to store the mesh connectivity and to maintain the boundaries. Besides pointers to the origin, to the next half-edge around the origin, and to the inverse half-edge, we have two pointers to reference a next and a previous boundary edge. This way we organize all edges of the same boundary into a cyclic doubly-linked list.

The decoding process reconstructs the connectivity of the polygon mesh with a single reverse traversal of the label sequence. Each label has a unique inverse operation (see Figure 3) that undoes the gate and boundary updates that happened during encoding. The time complexity for decoding is linear in the number of mesh edges. An exception is the inverse operation for label $M_{i,k,l}$ which requires the traversal of $k+l$ edges. However, labels of this type correspond to handles in the mesh, which are of rare occurence.

3.2 Compression

The label sequence produced by the encoding process is subsequently mapped into a bit-stream. The frequencies with which the different labels occur are highly non-uniform, which invites some kind of entropy encoding. There is also a strong correlation among subsequent labels, which can be exploited using a memory-sensitive encoding scheme. With a simple order-3 adaptive arithmetic coder [29] we achieve excellent compression ratios.

For an adaptive arithmetic encoder with three label memory the space requirement for the probability table grows as the cube of the number of symbols. Therefore we limit the number of labels in the input sequence to eight: F_3, F_4, F_5, F_c, R, L, S, and E. This allows the implementation of the arithmetic order-3 entropy coder to be both space and time efficient. The probability tables need only 4 KB of memory and we can use fast bit operations to manage them. Labels F_n with $n > 5$ are expressed through the combination of a label F_5 and $n-5$ subsequent labels of type F_c. We observe that labels of type F_n are never followed by label L or label E. We exploit this to express the typically infrequent appearing labels H_n and $M_{i,k,l}$ using the combinations F_4L and F_4E. The integer values associated with these labels are stored using a standard technique for encoding variable sized integers into bit-streams.

The compression scheme described above is extremely fast and produces very compact encodings for the label sequence. In Table 1 we give connectivity compression results in bits per vertex (bpv) for a set of popular example meshes.

Simple triangle or quadrangle meshes. A simple triangle (quadrangle) mesh is a polygon mesh without holes and handles whose faces are all triangles (quadrangles). In this case we can give encodings with guaranteed bit-rates that are theoretically interesting. A simple triangle mesh with v vertices has $3v-6$ edges and $2v-4$ triangles. Thus, $2v-4$ labels are F_3 while the remaining $v-2$

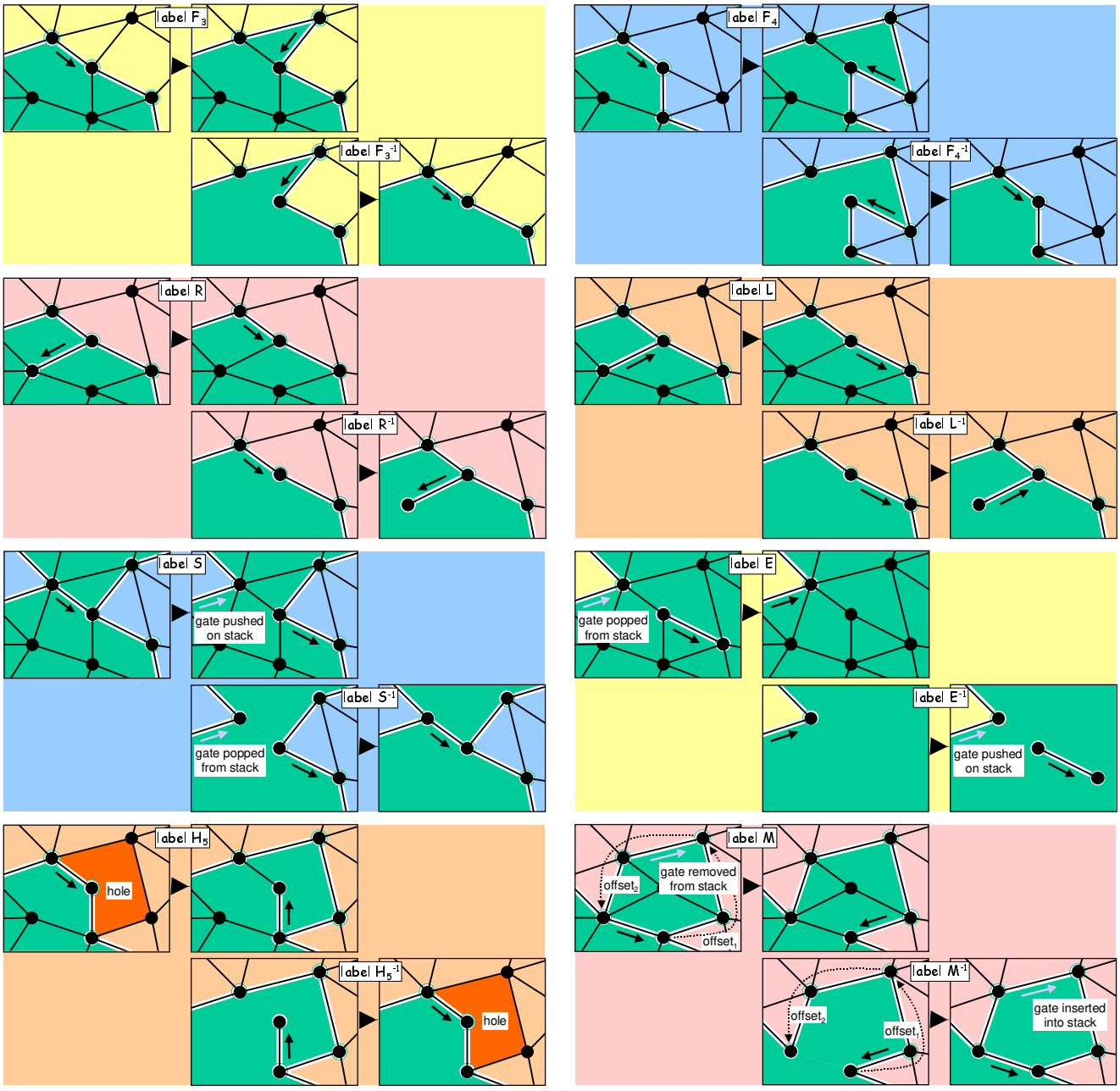

Figure 3: The labels of the Face Fixer scheme: when they apply and the corresponding updates during encoding and decoding.

labels are R, L, S, or E. An encoding that uses 1 bit for label F_3 and 3 bits each for the other labels guarantees a $5v - 10$ bit encoding.

Similarly, a simple quadrangle mesh with v vertices has $2v - 4$ edges and $v - 2$ quadrangles. Here $v - 2$ labels are of type F_4 while the remaining $v - 2$ labels are R, L, S, or E. An encoding that uses 1 bit for label F_4 and 3 bits each for the other labels guarantees a $4v - 8$ bit encoding.

3.3 Quadrilateral Grids

Instead of fixing together faces the Face Fixer scheme can also fix together patches of faces. Then we have to describe in addition the interior of these patches. If a patch is a rectangular quadrilateral grid this can be done very efficiently through the number of rows and columns in this grid. The beethoven bust and the shark model in Figure 4 for example, contain large patches of quadrilateral grids.

We introduce the label $QG_{r,l,h}$ to include such a *quad grid* into the active boundary. The associated integer values r, l, and h count the number of quadrangles that this grid extends to the right, to the left, and across as seen from the active gate (see Figure 4).

Optimal selection of a set of non-overlapping quad grids on the model is not only NP-hard, we also lack a well-defined optimality criterium. Including quad grids into the active boundary breaks up the regularity of the label stream, which in turn hampers subsequent arithmetic coding. However, first results using greedy methods are promising: The connectivity of the teapot, for example, compresses down to 1.069 bpv using 10 quad grids, for the shark 1.374 bpv, for the galleon 2.190 bpv, and for the beethoven bust 2.591 bpv.

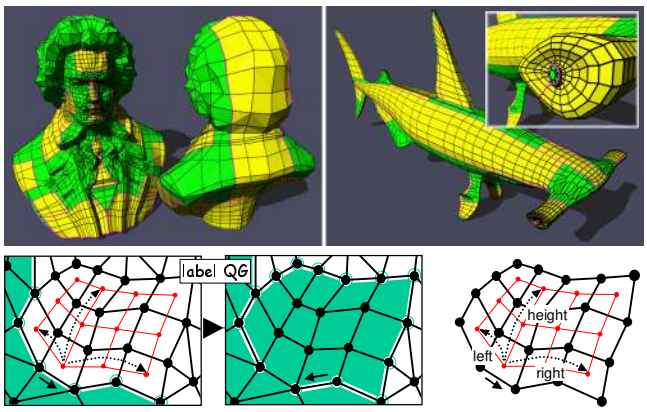

Figure 4: The beethoven bust and the shark model with quad grids marked in yellow (top). The label QG encodes a quad grid by specifying its left and right extend and its height (bottom).

4 PROPERTIES AND STRUCTURES

The Face Fixer scheme as presented so far allows to efficiently compress and uncompress the connectivity of a polygon mesh. However, polygonal models have geometry data associated with each vertex that specifies their physical location in 3D. Additional property data, such as normals, colours, and texture coordinates, is often attached to the vertices, the faces, or the corners of the mesh. To establish a connection between the geometry and property data and the vertex, face, or corner they are associated with, we define an implicit ordering on the occurrence of these mesh elements. Such an ordering can be derived using any deterministic mesh traversal that starts at a known point. Then the encoder stores the geometry or property values in the order in which the mesh features they are attached to are encountered during the traversal. Decoding performs the same traversal and re-assigns the data to the appropriate places.

Compression schemes that use the traversal order induced by the connectivity encoder to attach geometry and property data to the mesh are called *one-pass* coders. Carefully designed [7], they can combine connectivity, geometry and property information into a single bit-stream, which makes it possible to *stream* a mesh across a network. Then decompression can reconstruct the mesh incrementally as the bits are received. Time-critical applications benefit from such a scheme as transmission and reconstruction of the mesh can run in parallel. Other one-pass encoders [25] keep connectivity data separate from the rest in order to compress each more efficiently.

Reconstructing the polygon mesh in a single pass forces the predictive encoding for geometry and property data to make its estimations with incomplete neighbourhood information. A *multi-pass* coder stores the connectivity data separately from property and geometry data and traverses the mesh two or more times during encoding and decoding. The decoder first reconstructs the complete connectivity information before re-attaching geometry and property data to their appropriate location. In this case the mesh traversal used to establish the implicit ordering of geometry and property data can be different from the one used by the connectivity encoder.

4.1 Predictive Compression

The Face Fixer scheme can be combined with previously proposed techniques for predictive compression of geometry and property data [3, 24, 25]. Since the prediction rules of these schemes assume meshes with triangle connectivity, we could simply triangulate the polygons using a deterministic strategy that is solely based on the connectivity. However, even though this paper does not address predictive compression, we believe that the recovered polygon information can be utilized for more accurate geometry prediction.

For high-quality polygonal models like those in the Viewpoint Premier collection [28], faces are nearly planar and convex. Although a face may not be not perfectly planar, major discontinuities are improbable to occur across it—otherwise it would likely have been triangulated when the model was designed. This can lead to an improvement in predictive geometry encoding: After the positions of three vertices of a planar face are known, the 3D problem of predicting the coordinates for the remaining vertices around the face reduces to 2D. The embedding planes of multiple neighbouring faces around a vertex give additional hints for predicting its location.

The convexity constraint can lead to further improvements in the accuracy of the prediction. The *parallelogram* rule introduced by Touma and Gotsman [25] uses the assumption that adjacent triangles form a parallelogram for predictive coding. While two adjacent triangles can violate this assumption quite drastically, a convex quadrangle can not. Their approach could be extended to define a *pentagon* or a *hexagon* rule for higher degree faces.

4.2 Vertex and Face Properties

A vertex-based property assignment is commonly used to achieve visually smooth transitions across face boundaries. In the same way the geometry data is shared by all faces around each vertex to avoid cracks in the surface, a common normal, colour, or texture coordinate eliminates discontinuities when interpolated shading (e.g. Gouraud shading) is applied. Geometry data and property data associated with a vertex are stored in the order the vertices are encountered during the traversal of the mesh.

A typical example for a face-based property assignment is a precomputed radiosity solution. Each face has assigned a colour that corresponds to the amount of light it emits or transmits. The property data associated with a face is stored in the order the faces are encountered during the traversal of the mesh. Obviously this is independent from the degree of the face. Here lies another advantage of the Face Fixer method over encoding schemes that first triangulate the input mesh. Splitting a face of degree n into $n-2$ triangles creates $n-2$ copies of its properties. Instead of encoding these properties once, they need to be encoded $n-2$ times.

4.3 Corner Properties

A corner-based property assignment becomes necessary to reflect physical discontinuities in the underlying 3D model. Vertices that lie along such a discontinuity have usually more than one associated normal, colour or texture coordinate, each of which they share with a disjoint set of adjacent corners. Five of our example models have vertices with multiple normals (see Table 3).

We need to establish a mapping between a property value and the set of corners it is associated with. Our approach is a simple but effective improvement on work by Taubin et al. [23]. They store a *discontinuity bit* with every corner that is "0" when this corner uses the same property as the previous corner in counterclockwise order and a "1" otherwise. Then the property data associated with a set of corners is stored in the order in which the corresponding corners marked with "1" are encountered during the traversal of the mesh. This approach requires as many bits as the mesh has corners.

Based on the observation that not all vertices have multiple properties, we propose a similar marking scheme that uses *vertex bits* and *corner bits*. We use one bit per vertex to distinguish vertices with a single property ("1") from those with multiple properties ("0"). The corners around every vertex with multiple properties are marked as described above (see Figure 5). We store the property data in the same order as the corresponding "1" bits appear in the bit sequence. The results in Table 3 show that this encoding gives savings of 20 % to 70 % over the method proposed by Taubin et al. [23].

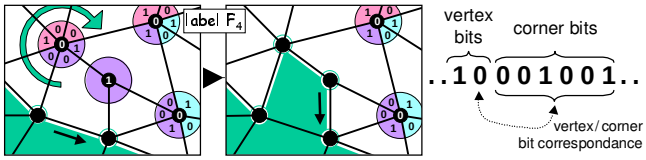

Figure 5: Encoding the mapping from properties to sets of corners.

name	vertices with n normals						vertex bits	corner bits
	n = 1	n = 2	n = 3	n = 4	n = 5	n = 6		
triceratops	2585	232	14	1	–	–	2832	980
galleon	1146	894	308	9	16	–	2372	4756
beethoven	1838	681	118	14	2	2	2655	3235
sandal	1120	1227	274	15	–	–	2636	6150
shark	1985	575	34	1	–	–	2560	2300

Table 3: Example results for encoding multiple vertex normals. The number of vertex bits and corner bits are reported that need to be recorded during the mesh traversal to establish a mapping between each normal and the set of corners sharing it.

4.4 Group Structures

Structural information that classifies groups of faces of a polygonal model into logical units is present in many file formats. Such face groupings allow to assign qualitative information to otherwise nameless polygons. Typically they establish a mapping between a meaningful part of the real world object and the set of faces of the polygonal model representing this part. We created and colour-coded such structural information for the popular teapot mesh and the cow mesh (see Figure 2). Many other well-known polygonal models, such as the triceratops, the cessna, the beethoven bust, and the galleon mesh contain similar group structures.

The triceratops mesh for example has six groups that classify each face as either skin, horn, toe, mouth, eye, or nose. The 58 faces that belong to the mouth-group form a single connected patch on the triceratops mesh. The 149 faces of the horn-group form three such patches and the 205 faces of the toe-group form fifteen.

The galleon model has a total of 17 groups. But this model consists of 12 unconnected components, which capture some of the group structure. The six sails, the three masts, the rig, and the lamp are separate components and form a group each. The body of the galleon however is one component with six groups: the hull, the keel, the deck, the aft, the windows, and the rig.

Encoding such structural information present in a polygon mesh has not been addressed by previously reported compression schemes. In a naive approach, we have a list of groups and assign a group index to every face. These group indices can then be treated like any other face property. For models with k groups and f faces such an encoding requires at least $f \log k$ bits.

When a model consists of several mesh parts we can improve on the above by specifying for each component the number of groupings it contains. For mesh parts whose faces belong all to the same group no additional information needs to be recorded. For mesh parts with group structures we need only as many bits per face as necessary to distinguish among the groups of this component. This is the approach we will compare our results against.

The concept of a *super face* is an natural extension of the Face Fixer scheme that leads to more compact and elegant encodings of face groupings. A super face is a collection of faces that is contained inside a single closed boundary loop. The representation power of super faces is illustrated in Figure 6: A simple super face composed of 9 faces (case A), a super face with a non-manifold vertex (case B), a super face with a non-manifold edge (case C), and a super face that contains another while being adjacent to a third (case D).

We introduce a new label SF to encode super face structure on the mesh. When the active gate is adjacent to a super face the active boundary is extended around the entire super face and the new gate is pushed on the stack. The super face is cut out of the polygon mesh and its boundary becomes the active boundary with the gate being the same as before (see Figure 7). The super face, which itself may contain other super faces, is first processed in its entirety before the encoding continues on the boundary that was pushed on the stack. The inverse operation used for decoding simply 'fixes' the super face back into the mesh. The length of the super face boundary is not encoded explicitly, but is directly related to the cost of encoding a super face. In addition to the label SF there is one additional label of types R, L, S, or E per super face boundary edge.

We use super faces to encode the group structure of a polygon mesh by declaring each group boundary a super face boundary. This results in super faces corresponding to cases A, B, and D. Ideally we would like a one to one mapping from groups to super faces. But when the faces of a group are not all adjacent, like the three horns of the triceratops, a group is represented by more than one super face. We could connect the three horns using non-manifold super face edges (case C) along a shortest path across the mesh. However, this requires additional computation and is expensive for distant patches because of the increasing length of the super face boundary.

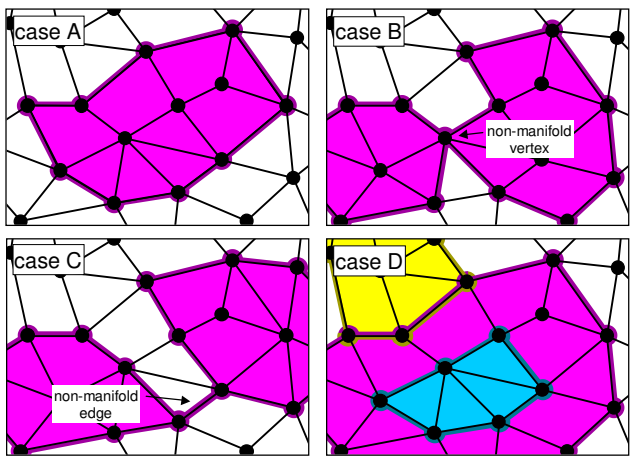

Figure 6: The representation power of super faces.

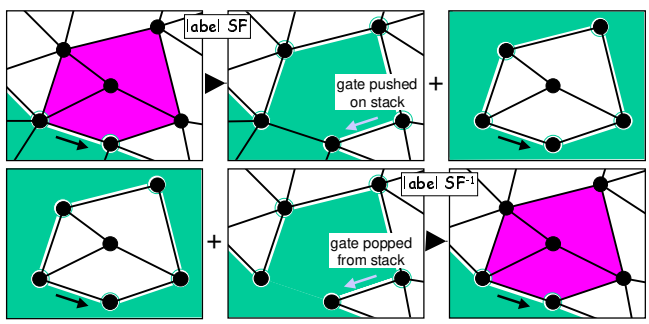

Figure 7: The label SF for encoding and decoding super faces.

Instead of storing one group index per face, we store one group index per super face and one group index per mesh component. Table 4 lists results for our pool of example meshes. Although more bits are needed to include the super face structure into the connectivity encoding, the savings in the number of necessary group references lead to superior compression rates overall.

Hierarchical Super Faces. The group structures that we have discussed so far are flat structures without a hierarchy. Suppose the

	bpv without super faces			bpv with super faces		
name	conn.	grouping	total	conn.	grouping	total
triceratops	2.115	8502	5.117	2.484	(23 + 1) * 3	2.509
galleon	2.595	2640	3.708	2.701	(10 + 12) * 5	2.747
cessna	2.841	15538	6.990	3.457	(137 + 11) * 6	3.694
beethoven	2.890	5470	4.950	3.081	(12 + 8) * 4	3.111
sandal	2.602	27	2.612	2.602	(0 + 9) * 3	2.612
shark	1.670	7686	4.672	1.962	(11 + 1) * 3	1.976
al	2.926	6039	4.595	3.051	(23 + 21) * 6	3.124
cupie	2.307	5060	4.003	2.387	(17 + 6) * 4	2.418
tommygun	2.611	5036	3.818	2.823	(12 + 39) * 4	2.872
cow	2.213	17412	8.209	2.346	(7 + 1) * 3	2.354
teapot	1.669	3870	4.924	1.853	(7 + 1) * 3	1.874

Table 4: Example results for encoding the grouping structure on a mesh together with the connectivity. The middle columns reflect the number of bits required for all indices into the group list. Both bit counts exploit the fact that the group structure is partly captured through the component structure of the models.

hierarchical structure of a world map with the oceans being the lowest level containing continents and islands. The continents themselves are subdivided into countries, some of which are composed of states, then counties, etc. Super faces can be used to efficiently encode such hierarchical structures on a polygon mesh.

Corner Properties Revisited. Group boundaries in a polygonal model often reflect discontinuities in the represented real-world object, which in turn is reason to associate multiple properties to sets of corners around a vertex. A substantial number of polygon meshes from the Viewpoint Premier collection [28] have vertices with multiple properties only along group boundaries. These meshes share a single normal, colour, or texture coordinate among all corners around a vertex that belong to the same group. In this case the super face structure is sufficient to establish the mapping between the property data and the appropriate set of corners. No additional information like corner or vertex bits is required.

5 SUMMARY

We have presented a new compression algorithm that encodes the connectivity of surface meshes directly in their polygonal representation. This has several benefits compared to methods that compress pre-triangulated polygon meshes: The original connectivity is preserved. Properties associated with faces and corners need not to be replicated. Subsequent stripification algorithms can generate better triangle strips. Predictive coding for geometry and property data can exploit additional convexity and planarity constraints.

Our method also improves on approaches that triangulate meshes prior to compression, but recover the polygons by marking edges. Although the freedom to triangulate polygons on demand can lead to compact encodings for the triangulated mesh, for example using Touma and Gotsman's scheme [25], the number of bits required to mark the edges would be as high as 3 bpv. This holds true especially for meshes with low triangle counts, as there is less connectivity information to compress. For the case of quadrangular meshes the Edgebreaker extensions [13, 15] give better encodings.

We have also introduced a method to encode structures on a polygon mesh. Such structures can specify classifications of faces, hierarchies, smoothing groups, or mesh partitions. The concept of super faces naturally extends our algorithm and leads to elegant and efficient encodings of such structural information—this has not been addressed by previous compression schemes. The edge-based nature of Face-Fixer combined with reverse decoding make the implementation of super faces extremely simple.

The techniques presented here lead to more compact and complete representations of polygonal models. The model collection from Viewpoint [28], for example, consists of data sets that contain few triangles, but are rich in structural information and associated properties. Especially with such 3D content moving towards networked environments, our methods will find many applications.

We owe a debt of gratitude to many people for their thoughts and time. We especially thank Davis King and Jarek Rossignac for discussions, Mike Maniscalco and Stefan Gumhold for tips on arithmetic coding, and Viewpoint Datalabs for the polygon models.

References

[1] C. Bajaj, V. Pascucci, and G. Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Visualization 99*, pages 307–316, 1999.

[2] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *Visualization 99 Conference Proceedings*, pages 67–72, 1999.

[3] M. Deering. Geometry compression. In *SIGGRAPH 95*, pages 13–20, 1995.

[4] M. Denny and C. Sohler. Encoding a triangulation as a permutation of its point set. In *Proc. of 9th Canadian Conf. on Comp. Geom.*, pages 39–43, 1997.

[5] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Visualization 96 Conference Proceedings*, pages 319–326, 1996.

[6] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi Diagrams. *ACM ToG*, 4(2):74–123, 1985.

[7] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH 98 Conference Proceedings*, pages 133–140, 1998.

[8] H. Hoppe. Progressive meshes. In *SIGGRAPH 96*, pages 99–108, 1996.

[9] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *Proceedings of SIBGRAPI 99*, Campinas, Brazil, pages 27–28, 1999.

[10] M. Isenburg and J. Snoeyink. Spirale reversi: Reverse decoding of the Edgebreaker encoding. Technical Report TR–99–08, Computer Science, UBC, 1999.

[11] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. In *Discrete Applied Mathematics*, pages 239–252, 1995.

[12] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proc. of 11th Canadian Conf. on Comp. Geom.*, pages 146–149, 1999.

[13] D. King, J. Rossignac, and A. Szymczak. Connectivity compression for irregular quadrilateral meshes. Technical Report TR–99–36, GVU, Georgia Tech, 1999.

[14] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12(1):28–35, 1983.

[15] B. Kronrod and C. Gotsman. Efficient coding of non-triangular meshes. In *Proc. of 16th Europ. Workshop on Computational Geometry*, pages 24–26, 2000.

[16] J. Li, C. C. Kuo, and H. Chen. Mesh connectivity coding by dual graph approach. Contribution Document MPEG98/m3530 Tokyo, mar 1998.

[17] R. Parajola and Rossignac. Compressed progressive meshes. Technical Report TR–99–05, GVU, Georgia Tech, 1999.

[18] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[19] J. Rossignac and A. Szymczak. Wrap&zip: Linear decoding of planar triangle graphs. *The Journal of Computational Geometry, Theory and Applications*, 1999.

[20] J. Snoeyink and M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications. In *Proc. of Europ. Symp. Alg.*, pages 459–471, 1997.

[21] D. M. Y. Sommerville. *An Introduction to the Geometry of N Dimensions*. Dutton Publications, New York, 1929.

[22] G. Taubin, A. Guéziec, W.P. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH 98 Conference Proceedings*, pages 123–132, 1998.

[23] G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proceedings of the IEEE*, 86(6):1228–1243, 1998.

[24] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[25] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface 98 Conference Proceedings*, pages 26–34, 1998.

[26] G. Turan. Succinct representations of graphs. *Dis. Apl. Math.*, 8:289–294, 1984.

[27] W.T. Tutte. A census of planar triangulations. *Cnd. Jrn. Math.*, 14:21–38, 1962.

[28] Viewpoint. *Premier Catalog (2000 Edition) www.viewpoint.com*.

[29] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[30] M. Woo, J. Neider, and T. Davis. *Open GL Programming Guide*. A.W., 1996.