Triangle Fixer: Edge-based Connectivity Compression

Martin Isenburg

University of North Carolina at Chapel Hill
isenburg@cs.unc.edu

1 Introduction

Encoding the connectivity of triangle meshes has recently been the subject of intense study and many representations have been proposed [9, 10, 4, 8, 2, 5]. The sudden interest in this area is fueled by the emerging demand for interactive visualization of 3D data sets in a networked environment (e.g. VRML over the Internet). Since transmission bandwidth across wide-area networks is a scarce resource, compact encodings for 3D models are of great advantage.

Common representations for triangle meshes use two lists: a list of vertices and a list of triangles. The list of vertices contains coordinates that specify a physical location for each mesh vertex. This is referred to as the geometry of the triangle mesh. The list of triangles contains triplets of indices into the vertex list that specify the three vertices of each triangle. This is referred to as the connectivity of the triangle mesh. For triangle meshes with v vertices, the triangle list uses at least $3 \log_2 v$ bits per triangle. Euler's relation implies that there are approximately twice as many triangles as vertices, giving a total of $6v \log_2 v$ bits for the mesh connectivity.

In this paper we introduce a simple and efficient scheme for encoding the connectivity of a triangle mesh. Using seven operations our algorithm traverses all edges of the mesh and records a sequence of corresponding labels T, R, L, S, E, H, and M. For every triangle there is a label of type T, for every hole there is a label of type H, and for every handle there is a label of type M. The remaining labels R, L, S, and E correspond to the edges of a vertex spanning tree that 'fixes' triangles and holes together.

2 Connectivity Compression Techniques

Most efficient connectivity compression schemes for triangle meshes [9, 10, 4, 8, 2, 5] follow the same pattern: They encode the mesh through a compact and often interwoven representation of a vertex spanning tree and its corresponding dual, a triangle spanning tree. Neither the triangle nor the vertex spanning tree are by themself sufficient to capture this connectivity information. All schemes start at an arbitrary edge and traverse the vertices and the triangles of the mesh using a deterministic strategy (e.g. breadth or depth first search).

Turan [11] was one of the first to observe that the fact that planar graphs could be decomposed into two spanning trees implied that they could be encoded in a constant number of bits per vertex. He gave an encoding that uses 12 bits per vertex (bpv).

Keeler and Westbrook improved Turan's scheme for encoding planar graphs. They specialize their encoding of planar graphs and maps [6] and achieve a guaranteed 4.6 bpv encoding for simple triangle meshes.

Taubin and Rossignac have the only scheme that explicitly encodes both the vertex and the triangle spanning tree. Their Topological Surgery method [9] 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 represents the dual triangle spanning tree. A rather complicated decoding algorithm can reconstruct the connectivity from these two trees. Run-length encoding both trees results in bit-rates of around 4 bpv.

Touma and Gotsman's Triangle Mesh Compression scheme [10] encodes the degree of each vertex along a spiraling vertex tree. For branches in the tree they need an additional split code. This technique implicitely encodes the triangle spanning tree. They compress the resulting code sequence using a combination of run-length and entropy encoding and achieve bit-rates as low as 0.2 bvp for very regular meshes and 2 ˜ 3 bpv otherwise.

Isenburg and Snoeyink's Mesh Collapse Compression [5] encodes the mesh connectivity with an invertible sequence of edge contract and edge divide operations. They record a vertex degree for each contract operation and a start and an end symbol for each divide operation. Entropy encoding this code sequence results in bit-rates that range from 1.1 to 3.6 bpv.

Gumhold and Strasser [4] introduce a compressed representation for triangle meshes that traverses the triangles of the mesh and includes them into a *cut-border* using six different operations. The reported compression rates vary from 3.5 to 5 bpv, but no upper bound is given.

Independently developed but similar is Rossignac's Edgebreaker scheme [8]. Seven operations are used to include triangle after triangle into an *active boundary*. For meshes without holes and handles a guaranteed 4 bpv encoding exists. The work by King and Rossignac [7] improves this bound to 3.67 bpv. This is currently the lowest worst case bound known and lies within 13% of the theoretical lower limit by Tutte [12].

De Floriani et al. [2] presented a technique that is closely related to the two methods above. Their compression algorithm is simpler as it uses only four operations to traverse the triangles of the mesh. However, this limits their scheme to the class of extendably shellable triangle meshes [1]. For those a bit-rate of 6 bpv is guaranteed and experimental bitrates of 4.1 to 4.5 bpv are reported. Triangle meshes with holes and handles can be compressed by partitioning them into shellable patches. This increases the observed bitrate to 5 bpv and higher. Furthermore it requires the replication of all vertices shared by more than one patch (up to 30%).

3 Triangle Fixer

Inspired by Rossignac's Edgebreaker method [8], we propose a novel edge-based approach for connectivity compression. The Triangle Fixer scheme expects the input mesh to be a 2-manifold surface with boundary composed of consistently oriented triangles. This means that the neighbourhood of each vertex can be mapped to a disk or a half-disk. The input mesh might consist of several connected components and can have multiple holes or handles.

The connectivity of the input mesh is encoded as a sequence of labels T, R, L, S, E, H, and M. The total number of labels equals the number of mesh edges. For every triangle there is a label of type T, for every hole there is a label of type H, and for every handle there is a label of type M. The remaining labels R, L, S, and E describe how to 'fix' triangles and holes together.

Subsequently this sequence of labels is compressed into a compact bit-stream by assigning a unique bit-pattern to every label. We observe that dependencies among subsequent labels can be exploited for more compact encodings. Therefore we compress the label sequence with a simple order-3 adaptive arithmetic coder [13] and achieve high compression ratios.

3.1 Encoding

The encoding process defines an *active boundary* in clockwise orientation around an arbitrary edge of the mesh. This initial boundary has two *boundary vertices* and two *boundary edges*. One of them becomes the *gate* of the boundary. The gate of the active boundary is the *active gate*.

In every step of the encoding process the active gate is labeled with either T, R, L, S, E, H, or M. 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 (T and H), shrinks (R and

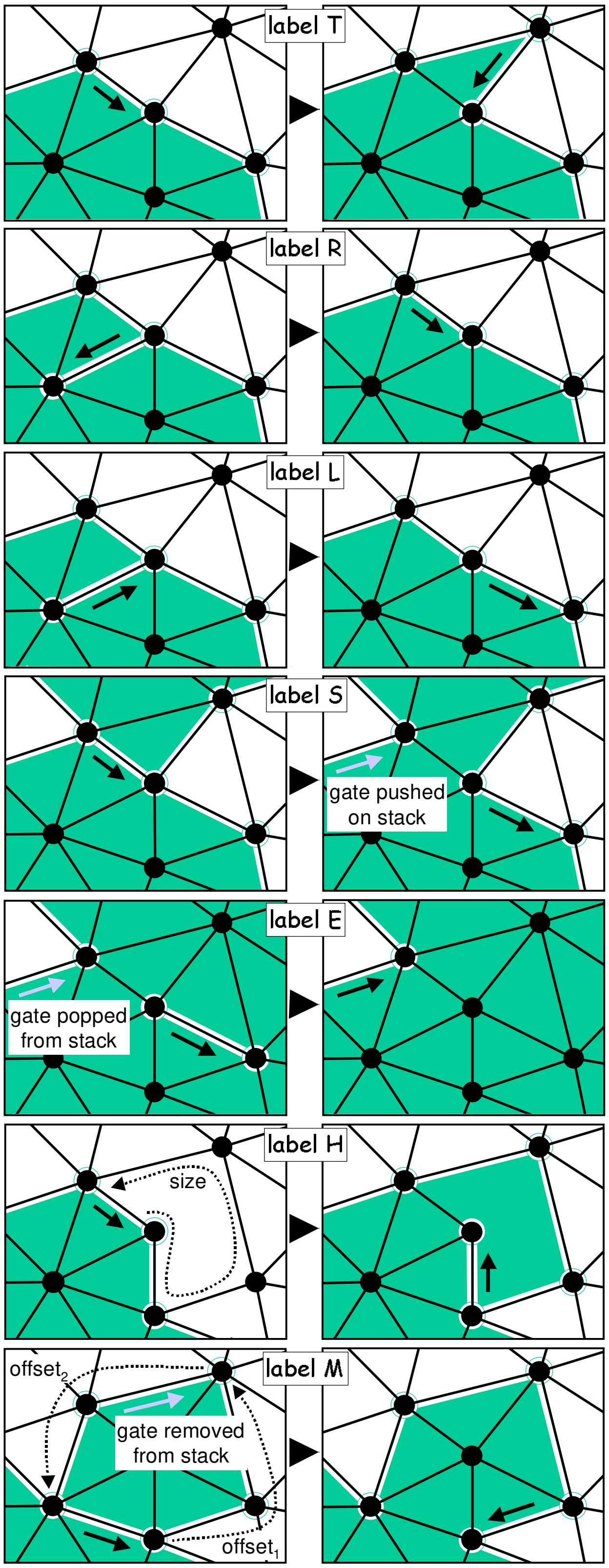

Figure 1: The labels T, R, L, S, E, H, and M of the Triangle Fixer encoding scheme.

L), splits (S), ends (E), or merges (M). See Table 1 for a summary. An initially empty stack of boundaries is used to temporarily buffer boundaries. The encoding process terminates after exactly e iterations where e is the number of mesh edges.

label	boundaries	edges	vertices
init	1	2	0
T	–	+1	–
R	–	-2	+1
L	–	-2	+1
S	+1	–	–
E	-1	-2	+2
H_{size}	–	+(size-2)	–
M_{idx,off_1,off_2}	-1	-2	–

Table 1: The relative changes in number of boundaries, boundary edges, and abandoned vertices for each label.

Figure 1 illustrates for all seven labels the situation in which they apply and the respective updates for gate and boundary that they imply. They are as follows:

label T The active gate is not adjacent to any other boundary edge, but to an unprocessed triangle. The active boundary is extended around this triangle. The new active gate is the right edge of the included triangle.

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 edge and the previous edge. Then the active boundary consists of only two edges which are 'fixed' together. If the boundary stack is empty the encoding process terminates. Otherwise it continues on the boundary that is popped from the stack.

label $\mathbf{H}_{size}$ The active gate is not adjacent to any other boundary edge, but to an unprocessed hole. The active boundary is extended around this hole. The *size* of the hole (e.g. the number of edges around the hole) is stored with the label. The new active gate is the rightmost edge of the included hole.

label $\mathbf{M}_{idx,off_1,off_2}$ 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 idx in the stack and two offset values (see Figure 1) are stored with the label. The new active gate is the previous edge along the boundary from the stack.

An example run for the encoding and the decoding process is given in Figure 2 and 3. We use an enhanced half-edge structure [3] during encoding and decoding to store the mesh connectivity and to maintain the boundaries. Besides pointers to the origin, to the next and the previous edge around the origin, and to the invers half-edge, we have two pointers to reference a next and a previous boundary edge. This way we organize all half-edges of the same boundary into a cyclic doubly-linked list.

The Triangle Fixer encoding scheme as presented so far captures the connectivity of an unlabeled mesh. Together with a permutation of vertex data it captures the connectivity of a labeled mesh. The vertex data, such as coordinates, texture information, and normals, are stored in the order in which the vertices are abandoned by the encoding process. Vertices are abandoned in the moment they are removed from the active boundary. The operations R and L remove one boundary vertex, the operation E removes two boundary vertices.

3.2 Decoding

The recorded information (e.g. the sequence of labels) is sufficient to uniquely invert each boundary and gate update that was performed during encoding. Thus, we decode the mesh connectivity by processing the labels in reverse order, while performing the inverse of every label operation. Every update can be performed in constant time, which gives us linear time complexity. An exception is the inverse operation for label M, which requires the traversal of $offset_1$ + $offset_2$ edges. However, labels of type M correspond to handles in the mesh, which are of rare occurrance.

This reconstructs the connectivity of the unlabeled mesh. Using the reverse of the of vertex data permutation produced by the encoding process, the mesh labeling is reconstructed. The vertex data is assigned in the order in which the vertices are encountered by the decoding process. Vertices are encountered in the moment they are inserted into the active boundary. The operations R and L insert one boundary vertex, the operation E inserts two boundary vertices.

mesh characteristics				bits per vertex	
name	vertices	holes	handles	fixed	aac-3
marcy-U1	42943	1	-	4.008	2.432
marcy-U2	28510	1	-	4.015	2.495
marcy-U3	13057	1	-	4.017	2.534
marcy-U4	6221	1	-	4.013	2.561
marcy-A1	15389	1	-	4.002	2.382
marcy-A2	15233	1	-	4.001	2.368
marcy-A3	15515	1	-	4.003	2.369
marcy-A4	15624	1	-	4.001	2.357
shape	2562	-	-	3.998	0.769
fandisk	6475	-	-	4.007	1.671
eight	766	-	2	4.090	1.434
cow	3078	22	-	3.987	2.364
femur	3897	-	2	4.161	3.047
pieta	3476	-	7	4.147	2.932
skull	10952	-	51	4.221	2.957
bunny	34834	5	-	4.001	1.732
phone	33204	3	-	4.054	2.700

Table 2: Example results: The two compression ratio are achieved using a fixed bit assignment scheme (*fixed*) and an order-3 adaptive arithmetic coder (*acc-3*). Marcy U1-U4 are terrains at uniform accuracy, Marcy A1-A4 are terrains at variable accuracy, the remaining meshes represent three dimensional objects.

3.3 Compression and Results

Triangle meshes of v vertices without holes or handles have $3v-6$ edges and $2v-4$ triangles. This means that $2v-4$ labels are of type T, while the remaining $v-2$ labels have type R, L, S, or E. An encoding that uses 1 bit for label T and 3 bits each for the other labels guarantees a $5v-10$ bit encoding.

However, we notice a correlation among subsequent labels that is consistent across our wide range test models. For instance label R is more likely to be followed by label R than by label L, whereas label L is more likely to be followed by another label of type L.

after	TRLSE
T, R	1 2 4 3 4
L	1 4 2 4 3
S	1 4 3 4 2
E	1 2 4 4 3

We can exploit this correlation for compression by making the bit assignment dependent on the last label. Using 1 bit for label T and a varying assignment of 2, 3, 4 and 4 bits for labels R, L, S, and E guarantees a $6v-12$ bit encoding, while being in practice close to $4v$ bits. The table on the right describes the bit assignment we use.

The number of holes and handles of a mesh is generally small and so is the number of labels H and M. We observe that label T is never followed by labels L or E. So we can encode label H with the label combination TL and label M with the combination TE. The associated integer values are compressed subsequently using a standard technique for encoding variable sized integers into bit-streams.

The correlation among subsequent labels invites arithmetic encoding [13]. Using a simple order-3 adaptive arithmetic coder achieves excellent compression rates. Since the input sequence to the arithmetic coder contains only five different symbols, it can be efficiently implemented using less than 8 KB of memory for the probability tables. Experimental results for various meshes are listed in Table 2.

4 Summary and Acknowledgments

We have presented a new edge-based algorithm for compressing the connectivity information of triangle meshes. The Triangle Fixer scheme is very simple to implement and the achieved compression ratio compare well with those of previously reported schemes.

Many thanks to Paola Magillo for providing her set of example meshes, to Michael Maniscalco and Frederick Wheeler for technical support on arithmetic encoding, and to Jack Snoeyink for reviewing the paper.

5 References

[1] H. Bruggesser and P. Mani. Shellable decompositions of cells and spheres. *Math. Scand.*, 29:197–205, 1971.

[2] L. de Floriani, P. Magillo, and E. Puppo. A simple and efficient sequential encoding for triangle meshes. In *Proceedings of 15th European Workshop on Computational Geometry*, pages 129–133, 1999.

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

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

[5] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *Proceedings of SIBGRAPI'99 - 12th Brazilian Symposium on Computer Graphics and Image Processing*, pages 27–28, 1999.

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

[7] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proceedings of 11th Canadian Conference on Computational Geometry*, pages 146–149, 1999.

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

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

[10] C. Touma and C. Gotsman. Triangle mesh compression. In *GI'98 Conference Proceedings*, pages 26–34, 1998.

[11] G. Turan. Succinct representations of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.

[12] W.T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.

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

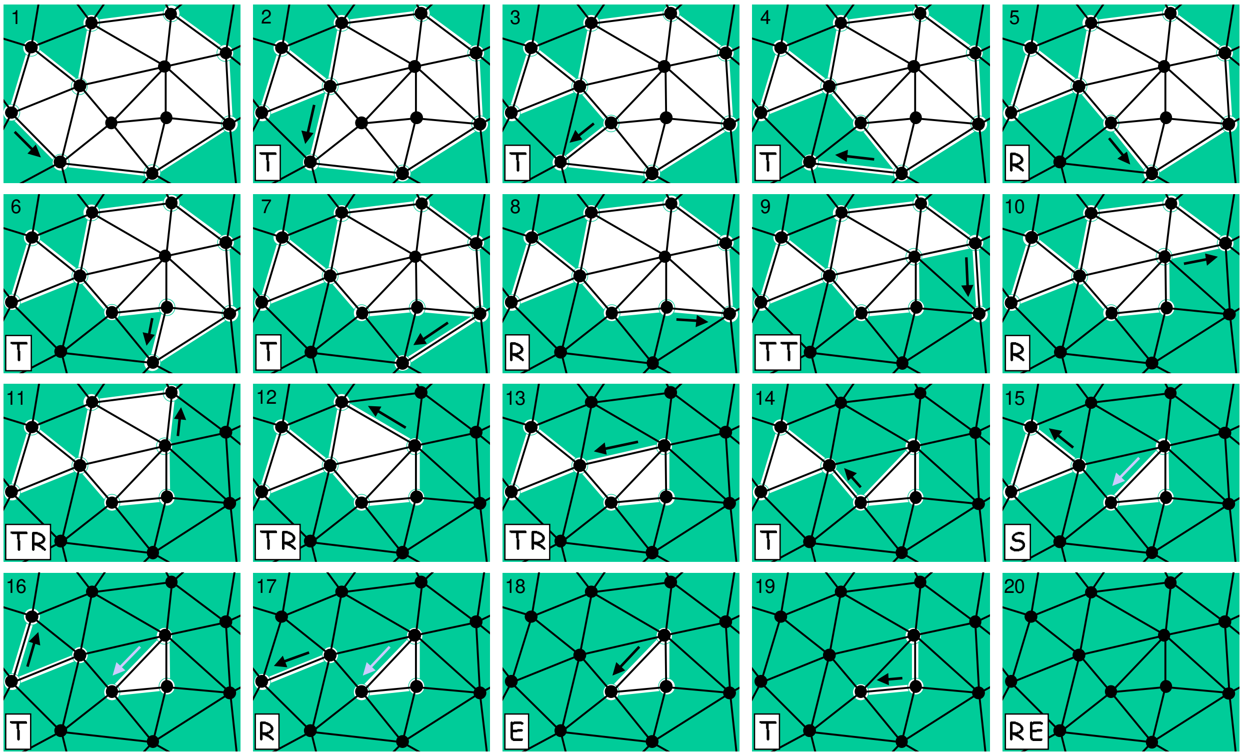

Figure 2: An example run showing the final 24 iterations of the encoding algorithm. Almost the entire mesh has already been processed except for the thirteen triangles shown in white. The interior of the active boundary is shaded dark, the active gate is denoted by a black arrow, and gates in the stack are shown as grey arrows. The label(s) in the lower left corner of each frame express the performed update(s) since the previous frame (compare to Figure 1).

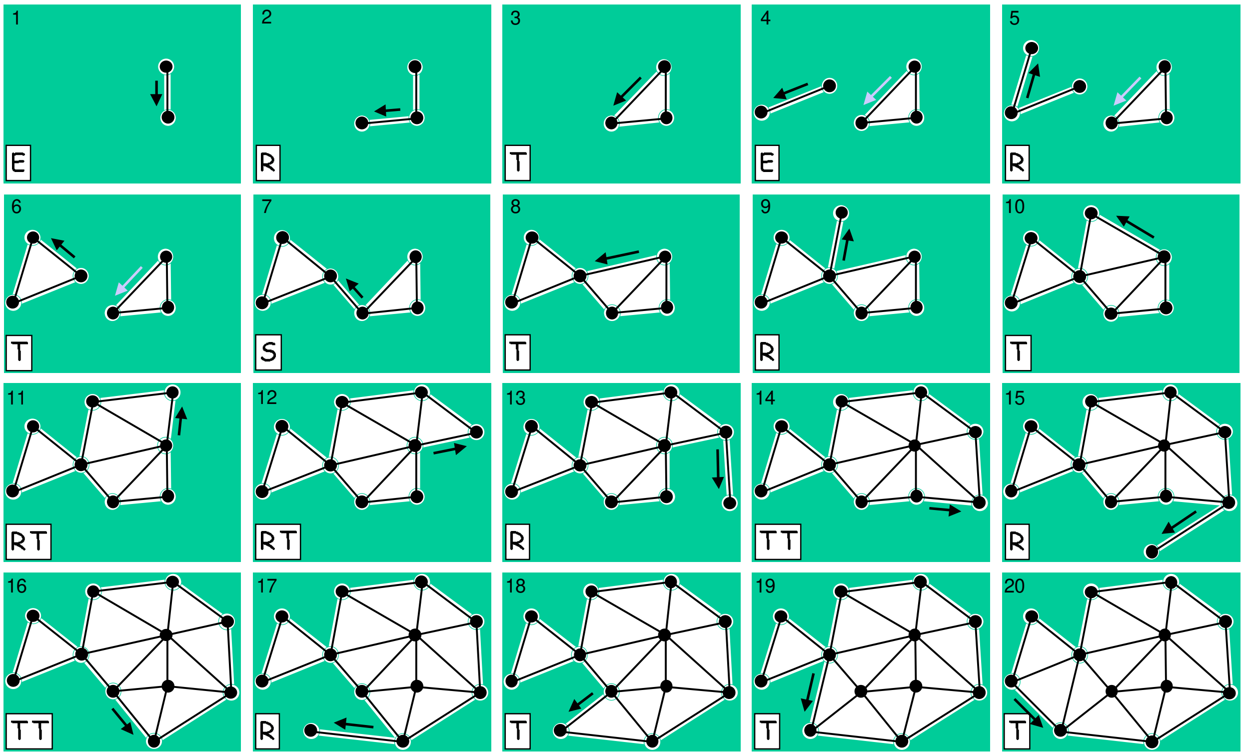

Figure 3: Here the first 24 iterations of the correspoding decoding process that reconstructs the connectivity of the mesh from the sequence of labels produced in Figure 2. Note that the labels are processed in reverse order.

APPENDIX (sneak preview)

The motivation for our edge-based approach to connectivity encoding was the design of a compression scheme for more general polygon meshes. That is, meshes composed of a mix of triangles, quadrangles, pentagons, hexagons, and higher degree faces. It proved to be a versatile technique with natural extensions towards efficient encodings for stripified triangle meshes and polygon meshes that contain structural information.

This is accomplished by a design choice that is the crucial difference between our scheme and similar approaches [4, 8, 2]. Our method slightly uncouples the traversal of the triangle spanning tree from the traversal of its corresponding vertex spanning tree. Triangles are included into the active boundary without immediately specifying their adjacency relation to all previously processed mesh components. This happens delayed and explicit through the labels R, L, S, E, and M.

A Face Fixer

The Triangle Fixer method can be seen as a specialization of our Face Fixer scheme for the case of fully triangulated input meshes. The Face Fixer scheme uses the labels F_3, F_4, F_5, F_6, F_7, etc. to encode polygonal faces in the mesh (see Figure 4). Label F_3 corresponds to label T and encodes triangles in the mesh. The other labels encode quadrangles (F_4), pentagons (F_5), hexagons (F_6), and so on. Some example results for various well-known polygon models are given in Table 3.

Face Fixer is the first compression method that encodes polygon meshes directly. Previous approaches compress a triangulated version of the input mesh and mark edges with bits to recover the polygon information.

name	vrts	△	▭	⬠	⬡	>⬡	aac-3
triceratops	2832	346	2266	140	63	19	2.355
galleon	2372	336	1947	40	18	43	2.595
chessna	3745	900	2797	180	27	23	2.841
shark	2560	188	2253	83	29	9	1.670
cupie	2984	384	2506	114	10	18	2.307

Table 3: Popular polygonal models, statistics about the faces they contain, and the achieved connectivity compression in bits per vertex with Face Fixer.

B Triangle Strip Compression

Supported in software and hardware, triangle strips are used for efficient rendering of triangle meshes. Since generating a good set of triangle strips is a hard problem, it is desirable to do this just once and store the computed strips with the mesh. However, no previously reported mesh encoding scheme is designed to include triangle strip information into the compressed representation.

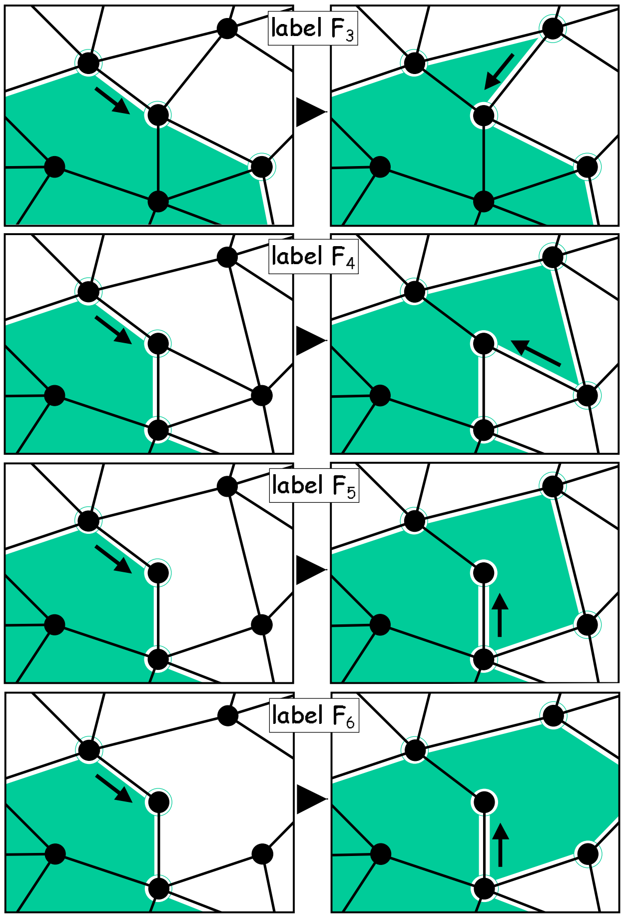

Figure 4: Face Fixer uses labels F_3, F_4, F_5, F_6, ... to encode polygonal faces in a mesh.

Our algorithm encodes the stripification and the connectivity of a triangular mesh in an interwoven fashion, that exploits the correlation between the two. It follows the concept of encoding mesh connectivity through an interwoven representation of a triangle spanning tree and its dual vertex spanning tree. But instead of traversing a triangle spanning tree using a deterministic search strategy we let the underlying stripification be the guide. The adjacency information that is encoded while walking along a strip means progress for both, the compression of connectivity and the compression of stripification.

name	vertices	strips	fixed	aac-3
marcy-U3	13057	707	4.313	3.755
shape	2562	2	3.092	0.617
cow	3078	152	4.002	3.238
femur	3897	237	4.484	4.021
bunny	34834	1229	3.692	2.399

Table 4: Compressing connectivity and stripification with a fixed bit scheme (*fixed*) and an arithmetic coder (*acc-3*).