Triangle Strip Compression

Martin Isenburg

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

Abstract

In this paper we introduce a simple and efficient scheme for encoding the connectivity and the stripification of a triangle mesh. 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 triangle mesh. However, no previously reported mesh encoding scheme is designed to include triangle strip information into the compressed representation. Our algorithm encodes the stripification and the connectivity in an interwoven fashion, that exploits the correlation existing between the two.

Key words: Mesh compression, connectivity encoding, triangle strips, triangle fans, stripification.

1 Introduction

Encoding the connectivity of triangle meshes has recently been the subject of intense study and many representations have been proposed [10, 11, 3, 7, 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 needed.

For interactive visualization not only the speed at which a triangle mesh can be received is important, but also the speed at which it can be displayed. Here the bottleneck is the rate at which this data can be sent to the rendering engine. Each triangle of the mesh can be rendered individually by sending its three vertices to the graphics hardware. Then every mesh vertex is processed about six times, which involves passing its three coordinates and optional normal, colour, and texture information from the memory to and through the graphics pipeline.

A 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 [2, 15] are widely supported by today's graphics hardware. Here two vertices from a previous triangle are re-used for all but the first triangle of every strip. Depending on the quality of the triangle strips this can potentially reduce the number of vertex repetitions by a factor of three.

For rendering purposes, an optimal stripification covers the mesh with as few strips using as few swaps [2] as possible. Computing an optimal set of triangle strips is NP-complete [1]. Various heuristics for generating good triangle strips have been proposed by Evans et al. [2], Speckmann and Snoeyink [8], and Xiang et al. [16].

Given the difficulty of generating good triangle strips it would be desirable to do this just once and store the computed stripification together with the mesh. Especially for data sets with a large distribution (such as the models from the Viewpoint Datalabs collection [13]) it is worthwhile to provide a good pre-computed stripification.

Currently available mesh compression techniques do not support the encoding of stripified meshes. Obviously one can enhance any existing compression method by encoding the stripification separately and concatenating the results. However, such a two-pass technique adds unnecessary overhead—it does not exploit the correlation between the connectivity and the stripification of a mesh.

In this paper we introduce a simple and efficient scheme for encoding the connectivity and the stripification of a triangle mesh. Enhancing Triangle Fixer, our edge-based connectivity compression algorithm [4], we compress this information in an interwoven fashion, that fully exploits the existing correlation.

2 Connectivity Compression Techniques

Most efficient connectivity compression schemes for triangle meshes [10, 11, 3, 7] 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. This is based on Turan's observation [12] that planar graphs can be encoded with a constant number of bits per vertex (bpv) when represented as a pair of spanning trees. Indexed triangle sets— the standard representation for triangle meshes— use at least $6 \log n$ bpv for the connectivity.

The Topological Surgery method [10] traverses the vertices of a mesh in a deterministic fashion (e.g. breadth or depth first search) and encodes the corresponding vertex and its dual triangle spanning tree separately. Run-length encoding both trees results in bit-rates around 4 bpv.

Touma and Gotsman's Triangle Mesh Compression scheme [11] records the degree of each vertex along a spiraling vertex tree. For branches in the tree they need an

additional split code. This technique implicitly 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 between 2 and 3 bpv otherwise.

Both the Cut-Border Machine [3] and the Edgebreaker scheme [7] include triangle after triangle into a boundary while traversing a spiraling triangle spanning tree. At each step they record the adjacency relation between the included triangle and the boundary, which implicitly encodes the vertex spanning tree. Follow-up work by King and Rossignac [6] establishes the currently lowest known worst case bound of 3.67 bpv.

Inspired by Rossignac's Edgebreaker scheme [7], we propose an edge-based approach for connectivity compression, which—as we will see later—has a simple and natural extension towards the compression of triangle strips. This is accomplished by a design choice that is the crucial difference between our Triangle Fixer scheme and previous approaches [3, 7]. Our method slightly uncouples the traversal of the triangle spanning tree from the traversal of its corresponding vertex spanning tree. Triangles are included into a boundary without immediately specifying their adjacency relation.

3 Triangle Fixer

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 can be compressed into a compact bit-stream by assigning a unique bit-pattern to every label. The correlation among subsequent labels can be exploited for more compact encodings with a simple order-3 adaptive arithmetic coder [14].

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 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

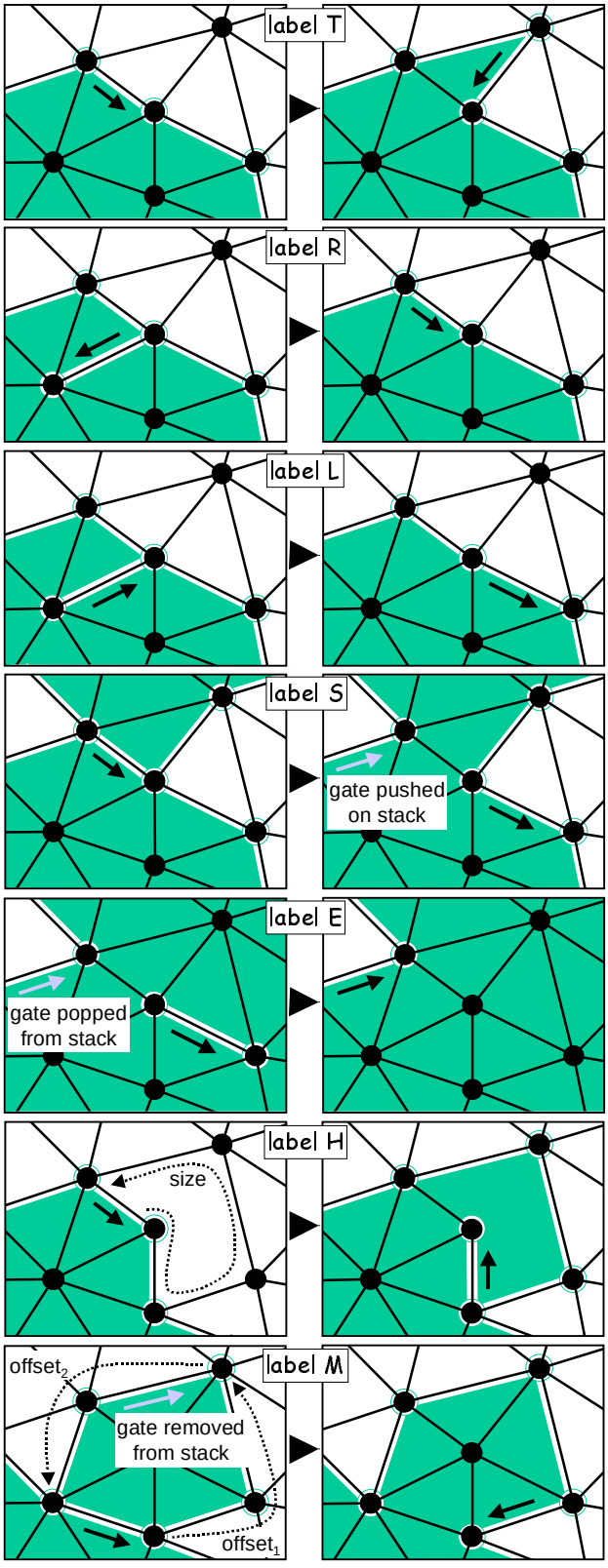

Figure 1: The labels T, R, L, S, E, H, and M. The black arrow denotes the active gate, the grey arrows denote gates in the stack.

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 L), splits (S), ends (E), or merges (M). 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.

In Figure 1 we illustrate 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 with which it is 'fixed' together. 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 with which it is 'fixed' together. 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. 'Fixing' the two edges together splits the active boundary. The previous and the next edge along the active boundary become gates for the two resulting boundaries. The first is pushed on the stack and encoding continues on the latter.

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 gate of the boundary that is popped from the stack.

label $\mathbf{H}_n$ The active gate is not adjacent to any other boundary edge, but to an unprocessed hole. The active boundary is extended around this hole. Its size n (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 $\mathbf{M}_{i,k,l}$ 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. The boundary is removed from the stack. Its former position i in the stack and two offset values k and l (see Figure 1) are stored with the label. The new active gate is the previous edge along the stack boundary.

We use a simple half-edge data structure 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.

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. 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 $k + l$ edges. However, labels of type M correspond to handles in the mesh, which are of rare occurrence.

mesh characteristics					bits per vertex	
name	vertices	triangles	holes	hndls	fixed	aac-3
bishop	250	496	-	-	4.00	1.86
shape	2562	5120	-	-	3.99	0.77
triceratops	2832	5660	-	-	4.00	2.52
fandisk	6475	12946	-	-	4.00	1.67
eight	766	1536	-	2	4.09	1.43
femur	3897	7798	-	2	4.16	3.05
skull	10952	22104	-	51	4.22	2.96
bunny	34834	69451	5	-	4.00	1.73
phone	33204	66287	3	-	4.05	2.70
terrainSM	13057	25818	1	-	4.02	2.53
terrainLG	42943	85290	1	-	4.01	2.43

Table 1: Compressing connectivity with a fixed bit assignment scheme (*fixed*) and an order-3 adaptive arithmetic coder (*acc-3*).

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 and $v - 2$ labels of 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.

We notice a correlation among subsequent labels that is consistent across our wide range of test models. Label R for instance is likely to be followed by label R, whereas label L is likely to be followed by another label of type L. We 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 above describes the bit assignment we use.

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

The number of holes and handles of a mesh is generally small and so is the number of labels H and M. Since label T can never be followed by labels L or E, we 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.

However, the correlation among subsequent labels also invites arithmetic encoding [14]. Experimental results for various meshes using a simple order-3 adaptive arithmetic coder are listed in Table 1. Since the input sequence to the arithmetic coder contains only five different symbols, it can be efficiently implemented using less than 4 KB of memory for the probability tables.

4 Triangle Strips

Supported in software and hardware, triangle strips are used for efficient rendering of triangle meshes. They reduce the data transfer rate between the main memory and the graphics engine by allowing the re-use of vertices for up to three consecutive triangles. This requires the graphics hardware to have a built-in buffer for two vertices, which is very common in today's graphic boards.

An OpenGL-style triangle strip is a sequence of m vertices $(v_0, ..., v_{m-1})$ that represents the sets of triangles $\{(v_i, v_{i+1}, v_{i+2})\}$ for even i and $\{(v_{i+1}, v_i, v_{i+2})\}$ for odd i with $0 \leq i < m-2$. The distinction between odd and even assures a consistent orientation of all triangles.

Two triangle strips and the vertex sequences that represent them are shown in Figure 2. The strip on the left is called *sequential*, because it turns alternating to the right and to the left. The sequence of 9 vertices describes 7 consistently oriented triangles. The strip on the right is not sequential, because it contains consecutive turns in the same direction. Such a strip is is called *generalized*. Here 10 vertices are necessary to describe the 7 triangles. In order to use vertex v_2 in 4 consecutive triangles the degenerate zero-area triangle (v_2, v_3, v_2) needs to be inserted into the strip. The cost for such a *swap* operation is one vertex, which is cheaper than a restart that costs two vertices.

The problem of constructing good triangle strips has been considered in several papers [2, 8, 16]. The objective is to minimize the number of swaps and restarts, thereby minimizing the total number of required vertices. Since computing the optimal solution is an NP-complete problem [1], heuristic search strategies are employed. For polygon models that are not fully triangulated the *patchification* method by Evans et al. [2] gives good results. This technique lets the triangle strips dictate the way the polygons are triangulated so that swaps are avoided.

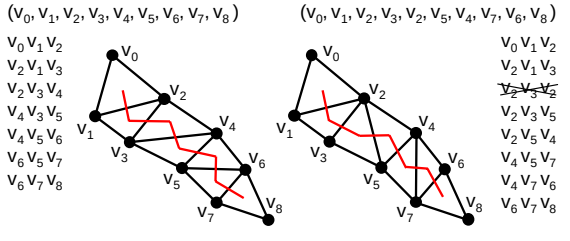

Figure 2: A sequential triangle strip (left) and a generalized triangle strip (right) with their corresponding vertex sequence.

Although the use of indexed triangle strips reduces the amount of data needed to represent the mesh connectivity by a factor between two and three compared to indexed triangle sets, it still needs at least $2 \log n$ bpv. For storage and transmission purposes it is often necessary to have a more compact representation of a mesh. However, current mesh compression techniques are not designed to encode stripified triangle meshes.

5 Triangle Strip Compression

The following is based on the observation that the stripification of a triangle mesh is uniquely defined by the set of *strip-internal* edges. These are edges that are shared by subsequent triangles in a strip. The set of strip-internal edges marks either none, one, or two edges of every triangle. A triangle without a strip-internal edge is a triangle strip by itself. A triangle with one strip-internal edge is either the start or the end of a strip. A triangle with two strip-internal edges is in the middle of a triangle strip.

It is necessary to distinguish the start from the end of a generalized triangle strip, because one direction sometime needs one fewer swap operation than the other. This can be computed in a single traversal of the triangle strip by counting the number of necessary swap operations.

Using one bit per edge (3 bpv) is sufficient to mark all strip-internal edges. Any previously reported mesh compression scheme could be combined with such an encoding of the stripification. However, this two-pass approach fails to exploit the redundancy between the connectivity and the stripification of a mesh: Every strip expresses the edge adjacency for each pair of subsequent triangles it contains. This local connectivity information also needs to be captured by the mesh compression scheme. Triangle Strip Compression specifies this information only once.

Our compression scheme follows the concept of encoding mesh connectivity through an interwoven representation of a triangle spanning tree and its dual vertex spanning tree. 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.

5.1 Encoding

As in Section 3.1, the encoding process initially defines the active gate and the active boundary around some edge of the mesh. However, now this choice is not completely arbitrary. The edge must not be strip-internal.

Again the active gate is labeled at each step of the encoding algorithm. Instead of label T we use the four labels T_R, T_L, T_B, and T_E. This subclassification captures the

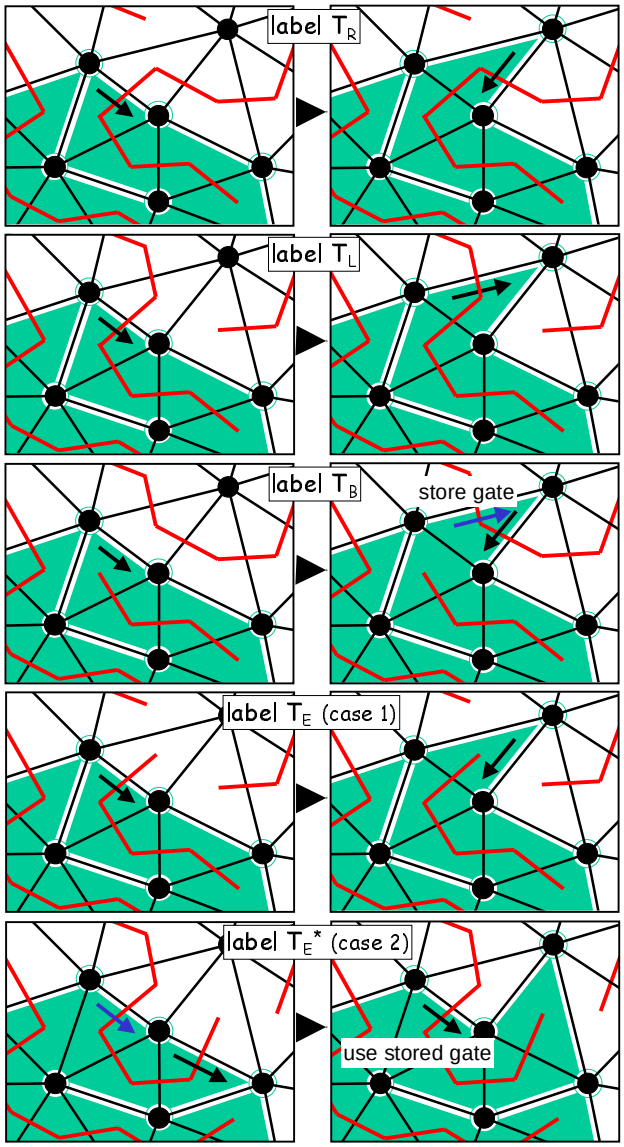

Figure 3: The labels T_R, T_L T_B, and T_E. The black arrow denotes the active gate, the dark-grey arrow denotes a stored gate.

stripification of the mesh. The four labels direct the way the encoding process traverses the mesh triangles so that it follows the underlying strips. Once a triangle strip is entered, it is processed in its entirety using these labels. The total number of edges that receive labels T_R, T_L, T_B, or T_E is equal to the number of mesh triangles. The labels R, L, S, E, H, and M are used and assigned as before.

Each of the four new label updates the boundary just like label T. The difference—illustrated in Figure 3—lies the way the active gate is updated. They are as follows:

label T_R The triangle strip leaves the included triangle through the right edge. The new active gate is this right edge.

label T_L The triangle strip leaves the included triangle through the left edge. The new active gate is this left edge.

label T_B The triangle strip leaves the included triangle through the right and the left edge, which means we just entered this triangle strip somewhere in its middle. Both directions need to be considered. Therefore the left edge is stored and the right edge is the new active gate.

label T_E (case 1) The triangle strip leaves the included triangle neither through the right nor through the left edge *and* this is the last triangle of this strip. The new active gate is the right edge.

label T_E^* (case 2) The triangle strip leaves the included triangle neither through the right nor through the left edge, *but* this is not the last triangle of this strip. Then there was a preceding label T_B. The edge that was stored with label T_B is the new active gate.

5.2 Decoding

As before, the labels are processed in reverse order and the inverse of each label operation is performed. However, one initial traversal of the labels in forward order is necessary. For every label T_B we count the number of encountered T_R labels before the first occurrence of a label T_E. We add 2 to the count and associate this value with the respective label T_E, marking it with a *.

When during the decoding process a label T_E^* with associated value w is encountered, we walk from the active gate w edges along the active boundary. The edge we arrive at is the new active gate and we continue normally.

This little variation becomes necessary to invert what happens during encoding: The first occurrence of a label T_E after a label T_B marks the completion of one end of a triangle strip. The active gate jumps to the edge that was stored with the preceding label T_B. The computed value expresses how many boundary edges were between the active gate and the stored edge at the time this jump occurred. The time complexity for decoding remains linear, since every triangle strip is traversed at most once.

The example in Figure 4 and 5 leads step by step through the encoding and decoding process of a small mesh with two triangle strips.

5.3 Compressing and Results

There is a very strong correlation among subsequent labels. We can observe long runs of labels R and L, and long sequences of alternating labels T_R and T_L. The simple bit assignment scheme that is described in the table above exploits these dependencies and achieves bit-rates between 3.0 and 5.0 bpv. This bit allocation scheme is geared towards

after	T_R	T_L	T_B	T_E	RLSE
T_R,T_E^*	2	1	–	2	----
T_L,T_B	1	2	–	2	----
T_E	4	5	3	6	2 7 1 7
R, E	7	6	5	7	1 4 3 2
L, S	6	7	5	7	4 1 3 2

mesh characteristics		corners of		bits per vertex	
name	strips	triangles	strips	fixed	aac-3
bishop	1	1488	498	2.98	1.78
shape	2	15360	5124	3.09	0.62
triceratops	144	16980	5948	4.12	3.49
fandisk	224	38838	13394	3.61	2.25
eight	24	4608	1584	3.46	1.78
femur	237	23394	8272	4.48	4.02
skull	600	66312	23304	4.74	4.18
bunny	1229	208353	71909	3.69	2.40
phone	1946	198861	70179	4.42	3.88
terrainSM	707	77454	27232	4.31	3.76
terrainLG	2404	255870	90098	4.41	3.83

Table 2: Compressing connectivity and stripification with a fixed bit scheme (*fixed*) and an arithmetic coder (*acc-3*).

long triangle strips with alternating left-right turns. The encodings become more compact with higher quality stripifications (e.g. fewer strips, fewer swaps).

The resulting compression rates (see Table 2) increase by at most 0.6 bpv for the fixed and 1.3 bpv for the arithmetic coder compared to those from Table 1. For very regular meshes the encodings are even more compact than before because such meshes can be decomposed into long sequential strips. Overall, the achieved compression rates for connectivity and stripification are significantly better than those of previously reported compression schemes for connectivity combined with an one bit per edge (3 bpv) encoding of the stripification.

We used version 2.0 of STRIPE [2] to stripify our example meshes. This software is designed for fast generation of triangle strips, hence the generated strips are not optimal. We would now like software that gives us higher quality strips at the expense of longer computation time.

6 Normals, Colours, and Texture Coordinates

Additional care needs to be taken when stripifying polygonal meshes that have multiple corner attributes per vertex. A corner is a vertex/triangle pair and corner attributes are typically vertex normals, colours, or texture coordinates. The idea of triangle strips is based on re-using the vertex data, which includes these corner attributes. Discontinuities in the model like a crease or a material change result in discontinuities in the corner attributes around a vertex. Good stripification software must assure that the triangle strips do not run across such discontinuities. Vertices have on average a set of three adjacent corners in a triangle strip and their attributes need to be consistent.

The above restricts the stripification process, but can be exploited for compressing the number of bits needed for mapping attributes to corners, which uses one bit per corner in the method by Taubin et al. [9]. Since the attributes of all adjacent triangle corners within a strip are consistent, we need to specify them only once for such a *strip corner*. The number of different triangle corners for meshes with t triangles is $3t$. However, decomposed into s strips we need to distinguish only the $t+2s$ strip corners for the mapping from attributes to corners (see Table 2).

7 Summary and Acknowledgments

Our main contribution is the compression of stripified meshes. We have extended Triangle Fixer to include information about a pre-computed set of triangle strips into the compressed representation of a mesh. Our algorithm fully exploits the existing correlation between connectivity and stripification of a mesh.

The new compressed format is especially useful for models with a large distribution. The computation of high quality stripifications is very expensive and, in particular for triangle meshes with corner attributes, not trivial. Once a good set of triangle strips has been computed, our technique allows to store and distribute it together with the model at little additional storage or processing cost.

Many thanks to Bettina Speckmann for discussions on triangle strips, to Xinyu Xiang for triangulating various models, and to Jack Snoeyink for reviewing the paper.

8 References

[1] F. Evans, S. S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical report, Department of Computer Science, State University of New York at Stony Brook, 1996.

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

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

[4] M. Isenburg. Triangle Fixer: Edge-based connectivity encoding. In *16th European Workshop on Comp. Geom.*, pages 18–23, 2000.

[5] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *Proceedings of 12th SIBGRAPI*, Brazil, pages 27–28, October 1999.

[6] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proc. of 11th CCCG*, pages 146–149, 1999.

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

[8] B. Speckmann and J. Snoeyink. Easy triangle strips for TIN terrain models. In *Proceedings of 9th CCCG*, pages 239–244, 1997.

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

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

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

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

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

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

[15] M. Woo, J. Neider, and T. Davis. *Open GL Programming Guide*. Addison Wesley, 1996.

[16] X. Xiang, M. Held, and J. Mitchell. Fast and efficient stripification of polygonal surface models. In *I3DG*, pages 71–78, 1999.

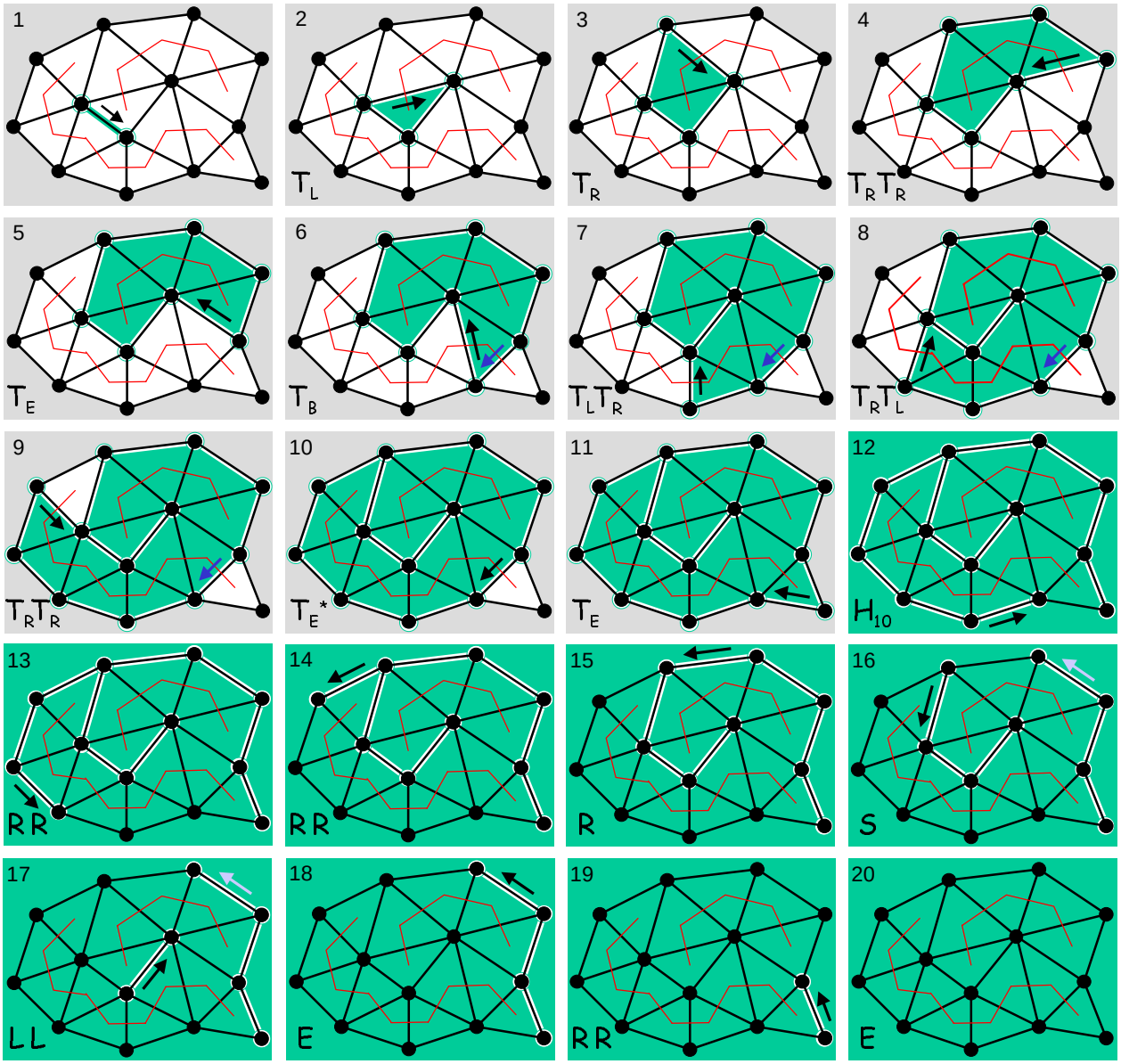

Figure 4: An example run of the encoding algorithm on a small mesh with two triangle strips. The interior of the active boundary is shaded dark, the active gate is denoted by a black arrow, a gate in the stack by a light-grey arrow, and a stored gate by a dark-grey arrow. The label(s) in the lower left corner of each frame express the performed update(s) since the previous frame. (1) Initial active boundary. (2-4) Boundary is expanded along the first triangle strip. (5) Reaching the last triangle of this strip. (6) Entering the second triangle strip in its middle. (7-9) Expanding this strip into one direction. (10) Finishing one side, the active gate jumps to expand other direction. (11) Finishing the other side. (12) Including a hole of ten edges. (13-15) Fixing the boundary with five R labels. (16) Splitting the boundary, one part is pushed on stack, continuing on other part. (17) Fixing the boundary with two L labels. (18) Ending this boundary, popping a boundary from stack. (19) Fixing the boundary with two R labels. (20) Ending this boundary, stack is empty, terminate.

Note: Instead of defining the initial active boundary around an edge we can also define it around a hole. In this example this would save us the label H_{10}. In general we want to define the initial active boundary around the largest hole of the mesh, which is also referred to as the boundary of the mesh.

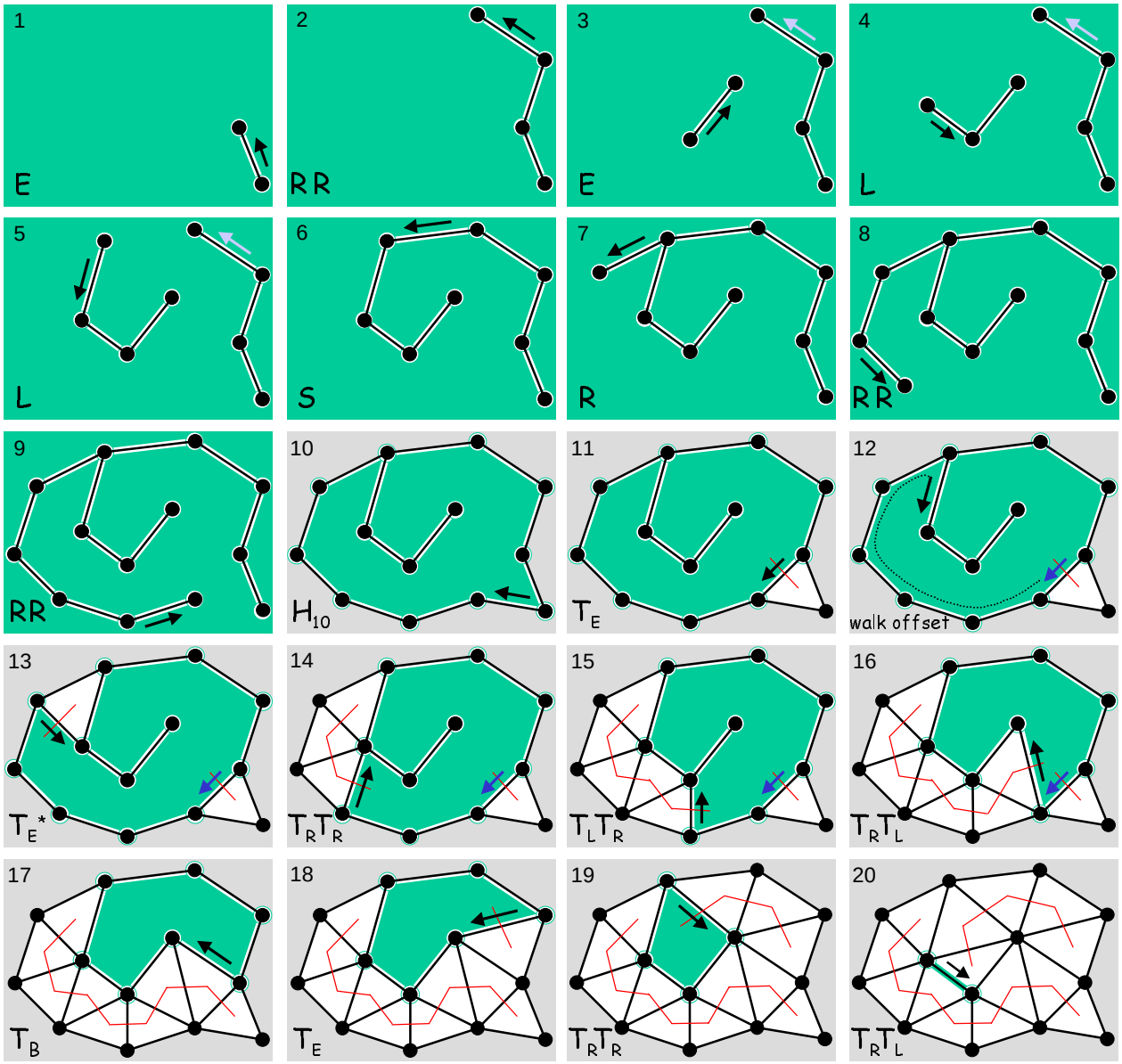

Figure 5: The decoding process that reconstructs connectivity and stripification of a mesh from the label sequence generated in Figure 4. The label(s) in the lower left corner of each frame indicate the inverted label operation(s) since the previous frame. In an initial forward traversal of the label sequence we mark the first occurrence of a T_E label after a T_B label with a *. Adding two to the number of T_R labels between T_B and T_E makes six, which is associated with the (now marked) label T_E^* (1) Creating a boundary of length two, undoing the last label E operation. (2) Expanding this boundary, undoing two label R operations. (3) Pushing the current boundary on the stack and creating a new boundary undoes another label E operation. (4-5) Expanding the boundary. (6) Merging the boundaries that were split by the S label. (7-9) Further expansion of the boundary. (10) Recreating a hole of size ten. (11) Recreating a triangle that starts the first strip. (12) Walking the offset associated with the marked label. (13) Recreating the first triangle at the other end of the strip. (14-16) Recreating six more triangles of this strip. (17) Finishing the first strip, by gluing its two sides together. (18) Recreating a triangle that starts the next strip. (19-20) Recreating four more triangles of this strip, terminate.