Wrap&zip: Linear decoding of planar triangle graphs

Jarek Rossignac and Andrzej Szymczak
GVU Center, Georgia Institute of Technology

Abstract

The Edgebreaker compression technique, introduced in [11], encodes any unlabeled triangulated planar graph of t triangles using a string of 2t bits. The string contains a sequence of t letters from the set {C, L, E, R, S} and 50% of these letters are C. Exploiting constraints on the sequence, we show that the string may in practice be further compressed to 1.6t bits using model independent codes and even more using model specific entropy codes. These results improve over the 2.3t bits needed by Keeler and Westbrook [6] and over the various 3D triangle mesh compression techniques published recently, which all exhibit larger constants or non-linear worst case storage costs. As in [11], we compress the mesh using a spiraling triangle-spanning tree and generate the same sequence of letters. Edgebreaker's decompression uses a look-ahead procedure to identify the third vertex of split triangles (S letter) by counting letter occurrences in the remaining part of the sequences. We introduce here a new decompression technique, which eliminates this look-ahead and thus exhibits a linear asymptotic time complexity. Wrap&zip converts the string into the corresponding triangle-spanning tree and assigns orientations to each one of its free edges. During that "wrapping" process, whenever two consecutive edges point to the same vertex, it glues them together, possibly continuing the "zip" along the next pair of edges that just became adjacent. By labeling the vertices according to the order in which they first appear in the triangle-spanning tree, this compression approach may be used to encode the connectivity (incidence of labeled graphs) of three-dimensional triangle meshes that are homeomorphic to a sphere. Being able to decompress connectivity prior to vertex locations is essential for the most advanced geometry compression schemes, which use connectivity to predict the location of a vertex from the location of its previously decoded neighbors.

Introduction

Planar graphs

We consider here planar triangle graphs of t triangles and v vertices. These are topologically equivalent to the connectivity graph of a triangulated surface that is homeomorphic to a sphere. Note that t=2v-4 for such graphs. In this paper, we use the term *mesh* to refer to such a graph.

Previously reported compression schemes

The connectivity of a mesh may be stored as a sequence of t triangle descriptors, each triangle been represented by 3 integer labels. Each labels identifies one amongst the v vertices and requires $\lceil \log_2(v) \rceil$ bits. Organizing triangles into strips [3], where each new triangle shares an edge with the previous one, reduces in practice the above storage by half. The use of a buffer to cache a small number of labels [1] may further reduce the expected cost.

The structure of a labeled planar graph may be encoded using slightly less than $12v$ bits [18]. Turan builds a vertex-spanning tree which represents the boundary of a topological polygon of $2v-2$ edges. The structure of the vertex-spanning tree is encoded with $4v-4$ bits. There are at most $2v-5$ edges that do not belong to the vertex-spanning tree. These may be encoded using 4 bits each. The overall connectivity cost is thus, 12v−24 bits.

Inspired by Tutte [19], Itai and Rodeh [5] show that any unlabeled rooted non-separable triangulated planar graphs of v vertices may be represented by 4v bits. They also propose a linear algorithm for constructing a representation of any labeled planar graph using at most $1.5n\log_2(v)+6v+O(\log_2(v))$ bits, while the theoretical minimum is $v\log_2(v)+O(v)$. They use a triangle as the initial outer loop and then shrink that loop by removing one triangle at a time. They always delete the triangle that is incident to the smallest vertex v_1 in the outer loop and is bounded by the outer loop edge that starts at v_1. They distinguish four cases: (1) The third vertex precedes v_1 in the outer loop; (2) It follows the successor of v_1; (3) It is somewhere else in the outer loop; and (4) it is not on the outer loop. Operations (3) and (4) each require $\log_2(n)$ bits to identify a vertex in the not yet processed part of the mesh. Several improvements over Itai and Rodeh's method were reported recently [14, 15, 12, 4, 11].

Cutting through the edges of the vertex-spanning tree produces a triangulated surface that is a simply connected mesh without internal vertices and thus may be completely represented by the triangle-spanning tree. A triangle-spanning tree of a mesh is a binary tree, whose nodes correspond to the triangles and whose edges correspond to some of the edges of the mesh. A depth-first traversal of such a spanning tree corresponds to a walk on the entire mesh that starts at the root triangle and recursively visits the neighboring triangles that have not been previously visited. The triangle-spanning tree may be encoded using 2 bits per triangle, but does not contain sufficient information to recover the mesh.

The Topological Surgery method of Taubin and Rossignac [14, 15] compress both a triangle-spanning tree and its dual vertex-spanning tree be encoding the lengths of consecutive single-child nodes. For complex and reasonably regular meshes, the expected cost of encoding both trees may amount to about two bits per triangle. However, the overhead of the run length encoding may result in a significantly higher average cost for irregular or small meshes.

Rossignac [12] has proposed a variation, which uses 2 bits per vertex to encode the vertex-spanning tree (one bit indicates the presence of a child while the other bit indicates the presence of a right sibling) and 2 bits per triangle to encode the triangle-spanning tree (one bit indicates the presence of a right child, while the other bit indicates the presence of a left child). With twice more triangles than vertices, the guaranteed worst case connectivity cost of this representation is 3 bits per triangle.

Gumhold and Strasser's technique [4] and Rossignac's Edgebreaker scheme [11], although developed independently, are closely related. Both schemes perform the same traversal of the mesh as in [14]. At each step, they remove a triangle and encode the necessary information to reconstruct the triangle by distinguishing several cases that include the four cases of Itai and Rodeh. Edgebreaker uses the letters L, R, S, and C to identify cases 1 through 4 of Itai and Rodeh. Gumhold and Strasser add the case where a boundary edge is reached. Edgebreaker does not need to distinguish this case, since it encodes the bounding loop at the beginning of the vertex array. However, Edgebreaker adds the case E, which corresponds to the situation where the current triangle is not adjacent to any other remaining triangle. Both these approaches avoid the $\log_2(v)$ bits cost associated with case (4) of Itai and Rodeh by encoding the vertices in the order in which they are visited by case (4). With each case (3) operation, Gumhold and Strasser must encode the reference to a vertex in the current boundary, which requires $\log_2(v)$ bits and makes their storage costs a non-linear function of v. Instead, Edgebreaker uses a decompression preprocessing step to compute these vertex-references from the sequence of symbols, and therefore exhibits a linear storage cost, although a non-linear time complexity. For typical meshes, Gumhold and Strasser report compression results between 1.7 and 2.15 bits per triangle using Huffman encoding of the bit stream.

Keeler and Westbrook [6] improve on Turan's results and propose a technique for encoding planar graphs with a guaranteed 4.6v bits. They also build a triangle-spanning tree. Each triangle of the tree, except the root, shares an edge with its parent and may have zero, one, or two children and thus two, one, or zero free edges. They append free edges to the leaves of the triangle-spanning tree and label them. Encoding the graph and the labels requires an average of $1+\log_2(3)/3$ bits per edge. They suggest a coding scheme based on a series of graph transformations.

Touma and Gotsman [17] also encode the vertices along the vertex-spanning tree in the same spiraling order as [14,4,11]. They distinguish only two cases, which correspond to the cases (3) and (4) of Itai and Rodeh and to the Edgebreaker's cases S and C. Other cases are not encoded. Instead, Touma and Gotsman encode the degree of each vertex, i.e., the number of incident edges and use it to automatically identify the other cases. During decompression, they keep track of the number of already decoded triangles that are incident upon each vertex and are thus capable of identifying the R, L, and E triangles automatically. For highly tessellated regular models, where the degree of the vertices follows almost regular patterns, they report compression results of less than a bit per triangle using Huffman encoding. However, for smaller or less regular meshes, the required storage may easily exceed 2 bits per triangle. As Itai and Rodeh and as Gumhold and Strasser, they require that with each S operation be associated a vertex reference, which requires $\log_2(v)$ bits, prior to Huffman compression.

Inspired by [7] and improving on [9, 13], Denny and Sohler have proposed a technique for encoding the incidence of *planar* triangulations of sufficiently large size as a permutation of the vertices [2]. They show that there are less than $2^{8.2v\ +O(\log v\ )}$ valid triangulations of a planar set of v points, and that for sufficiently large v, each triangulation may be associated with a different permutations of these points (there are approximately $2^{v\ \log(v)}$ such permutations). They transmit the suitably ordered vertices in batches. The decompression sorts them lexicographically, computes a permutation number by comparing the order in which the vertices were received with their lexicographic order, then sweeps over the current triangulation from left to right and refines it by inserting the new vertices. At each vertex of the current batch, it identifies the enclosing triangle [8] and the vertex is inserted according to the incidence relation derived from the bit string that encodes the permutation number. Unfortunately, the unstructured order in which the vertices are received and the absence of the incidence graph during their decompression makes it difficult to combine this approach with the predictive techniques for vertex encoding discussed earlier. Furthermore, the vertex-triangle association is based on geometric comparisons that only work in two dimensions.

The Edgebreaker compression

For completeness, we summarize here the compression process proposed in [11].

Spiraling triangle-spanning tree

The Edgebreaker compression algorithm visits the triangle of the mesh in the order in which they appear in a spiraling triangle-spanning tree. Such a tree may be built by a recursive procedure as follows.

During the recursive traversal, we leave some triangle P to enter an adjacent triangle X that has not been previously visited. P is the parent of X in the triangle-spanning tree. The other two triangles adjacent to X may be consistently identified as X.left and X.right, using a convention and a consistent orientation of the triangles throughout the mesh. If X.right has not been previously visited, it is appended as the right child of X and is visited by the recursive procedure in a depth-first order. Then, if X.left has not been visited, it is appended as the left child of X and is also visited by the recursion. We mark all the vertices of visited triangles.

The procedure starts with any triangle X of the mesh and identifies one of its edges as the starting gate, which suffices to define X.right and X.left.

Let **v** be the only vertex of X that is not the vertex of its parent in the tree. The visited/not-visited status of **v**, of X.left,

and of X.right unambiguously identifies one out of the five possible situations depicted Fig. 1 and associated with the letters: C, L, E, R, and S. Compression simply encodes the corresponding letters—or more precisely their binary op-codes—in the order in which the corresponding triangles are visited. The resulting string of t symbols represents a compact encoding of the connectivity of the mesh.

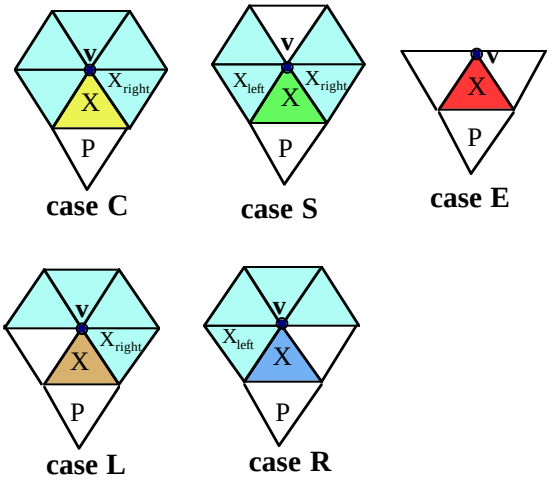

Figure 1: *Previously visited triangles are not shaded.*

- *Case C: **v** has not been visited.*
- *Case S: Only **v** has been visited.*
- *Case E: X.left and X.right have been visited.*
- *Case L: Only X.right was not visited.*
- *Case R: Only X.left was not visited.*

The selection of the appropriate case may be performed by the following sequence of tests:

```
if not visited(v) then C
 else if visited(X.left)
   then if visited(X.right) then E else L
   else if visited(X.right) then R else S
```

If the vertices are labeled in the order in which they are first visited by this traversal and then encoded in the order of increasing labels, this approach may be used to compress labeled graphs and 3D triangle meshes homeomorphic to a sphere [16].

Efficient encoding of the op-codes

Guaranteed 2t bit code

Except for the first two vertices, there is a one-to-one association between the vertices of the mesh and the triangles processed by C operations. Therefore, the number of Cs is v–2, which equals t/2, given that v=(t+4)/2. The total number of non-C operations, t–v+2, also equals t/2. Hence, if we use a1-bit code for C and 3-bit codes for the other four operations, the total cost for storing the string with the above scheme would be exactly 2t.

Because the first two operations are Cs, they can be omitted from the string, yielding a total storage cost of 2t–2 bits for any triangular planar graph.

Expected 1.7t bit code not exceeding 2t bits

The CL and CE sequences of operations correspond to impossible situations. We exploit this constraint to increase the expected compression ratio of Edgebreaker by using a slightly more complex coding scheme. We use two different code sets: the general code set proposed above is used for all operations except for S and R's that follow a C. These may each be encoded using only 2 bits. For example, we may have the following code. C is 0, L is 110, R that does not follow a C is 101, R that follow a C is 11, E is 100. S that does not follow a C is 111, and S that follows a C is 10.

In the worst case, with long sequences of consecutive Cs, this encoding method has no effect on the bit-count. At best however, when all Cs are separated, it reduces the bit-count to an average of 1.5 bit per triangle (because there are as many Cs as other operations). Our experiments on a variety of meshes show a ratio of 36% of R operations, almost half of which follow a C. The code proposed here results on the average storage of 1.7t bits.

Expected 1.6t bit code

By exploiting the relative frequencies of these operations in large meshes, we have devised a code (below) that works better in practice, but no longer guarantees never to exceed 2t bits. We encode CC, CS, and CR pairs as single symbols. The resulting file size is 1.6t bits for the large models that we have explored.

As indicated in the table below: E is 1100, L is 1110, R and S that follow even number of consecutive C's are respectively 10 and 1111, pairs of op-codes CR, CC and CS preceded by an even number of C's are respectively 01, 00 and 1101. The table also shows the frequencies of symbols or groupings in the string and their total storage cost, multiplied by 100/t.

Sequence	code	frequency	Cost for 100t
CR	01	15.5%	31 bits
CC	00	16.1%	32.2 bits
CS	1101	2.3%	9.2 bits
R (after even Cs)	10	20.8%	41.6 bits
E	1100	5.6%	22.4 bits
S (after even Cs)	1111	3.3%	13.2 bits
L	1110	2.5%	10 bits
TOTAL			159.6 bits

Custom 1.26b bit entropy codes

To further reduce the storage cost for large meshes, we introduce a space in the sequence of letters at each locations where a C follows a non-C operation. Doing so, we decompose it into words that have two parts: the first one contains one or more Cs and the second one contains one or more non-C operations.

We have constructed dictionaries for meshes with 200000 triangles and found only about 1400 words. A Huffman encoding of these sequences of words yields 1.26t bits of storage cost for those meshes. The table of codes takes up about 32000 bits (in this case, about 0.16 bits per triangle), but a part of it corresponding to most frequent words can be preloaded and kept constant for all large meshes. Words missing from the dictionary can be encoded as an escape sequence followed by their encoding using a fixed code scheme.

Progressive codes

An alternative is to use progressive coding schemes [22, 21, 10]. A number of general-purpose data compression schemes may be used for this purpose and will not be further discussed here. They may yield very high compression rations for large regular meshes, but often perform poorly for large, irregular meshes and for small meshes.

Wrap&Zip decompression

The decompression algorithm receives a binary encoding of the string of C, L, E, R, or S letters. It reproduces a labeled planar triangle graph that is homeomorphic to the original one with its vertices labeled as discussed in the compression section. The process is very simple.

Wrapping a triangle spanning tree

The wrap&zip decompression starts with the two initial triangles that correspond to the two initial C operations that have been omitted from the string. One of their external edges is identified as the gate. We read the string and for each operation, attach a new triangle to the gate. Depending on the op-code, we select zero, one or two of the free edges of the new triangle as the gates (Fig. 2). A stack is used to keep track of left branches of S operations, where the tree bifurcates. On an E operation, the current branch of the triangle tree terminates and we pop a new gate from the stack.

Orienting the free edges

During the above triangle-tree building process, edges that have not been gates are called the bounding edges. They have only one incident triangle. An E triangle has two bounding edges. S has none. C and L have a left bounding edge and R has a right bounding edge. These are oriented as shown with blue arrows on Fig. 2.

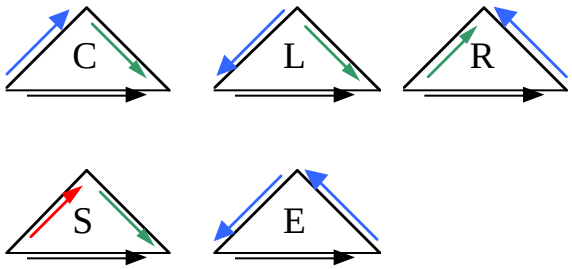

Figure 2: *A new triangle is attached to the gate (black arrow) and the new gate is indicated by a green arrow. The red arrow indicates a gate pushed onto the stack for the S operation. The blue arrows indicate the orientations of the bounding edges.*

Zipping the wrap

Each time two adjacent bounding edges point towards their common vertex (i.e. their blue-arrow orientations point towards that vertex), we zip them together by identifying their other ends. If the resulting, combined vertex exhibits the same property (i.e., its two incident bounding edges point towards it), we repeat this zip operation, and so recursively (Fig. 3).

Note that no zipping is possible for C, R, and S operations and that recursive zipping occurs only for E operations, starting from the left vertex (the starting vertex of the gate).

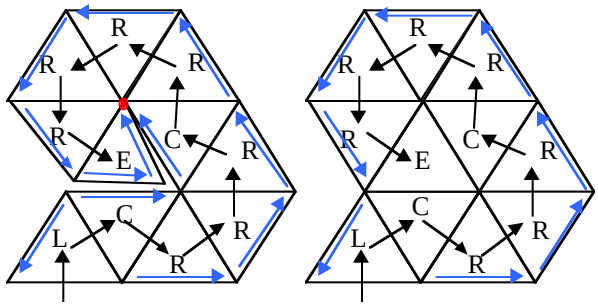

Figure 3: *The black arrows show the sequence in which the triangles were constructed from the subsequence LCRRRCRRRRE. The blue arrows (left) indicate the bounding edges and their orientation prior to the zipping for the last triangle in the sequence. The zipping operation starts at the red vertex and zips two pairs of edges. The result is shown right.*

Justification of the approach

Triangle spanning tree is the same as with other approaches [4,14,17,11]. It may always be converted into the correct connectivity by zipping pairs of edges. In fact, the vertex-spanning tree of [14] encodes precisely this zipping information. C operations are the only ones to add vertices to the vertex-spanning tree. If we orient the edges of the vertex spanning tree downwards (away from the root), then C operations create triangles that lie on the right of such edges. R, E, and L operations create triangles that lie on the left of such edges. Wrap&zip zips up the vertex spanning tree starting from its leaves, which are vertices with two incident bounding edges that point to them. Zipping an edge corresponds to removing it from the vertex spanning tree, and our recursive procedure ensures that we zip up all the children before we zip up past a branching node of the vertex spanning tree.

Time complexity

We start the recursive zipping procedure at most t times and, more precisely, once for each L and each E operation. Consequently, we also stop the zipping procedure the same number of times. (The zipping procedure goes up the tree and does not bifurcate.) We conclude that the number of times we test a vertex and decide not to zip is bounded by t. The number of successful zip operations equals the number of edges in the vertex-spanning tree, which is precisely v–1. Therefore, the decompression algorithm has linear time complexity.

Conclusion

The wrap&zip technique reported here improves the decompression algorithms for models compressed with the Edgebreaker approach. We have provided a simpler algorithm with linear complexity and very little overhead over the straightforward construction of the triangle-spanning tree.

By analyzing the statistics of the op-codes generated by the compression process for a variety of models, we have developed a model-independent coding scheme, which compresses each triangle to an average of 1.6 bits. This cost may be further reduced for large models down to 1.26 bits per triangle by using precomputed or adaptive entropy codes.

A 3D variation of this approach has been used by the authors to compress the connectivity graph of tetrahedral meshes [20].

Acknowledgement

Rossignac research on this project was partly supported by NSF grant 9721358. Szymczak was supported by KBN grant 0449/P3/94/06.

Bibliography

[1] M. Deering, Geometry Compression, Computer Graphics, Proceedings Siggraph'95, 13-20, August 1995.

[2] M. Denny and C. Sohler, Encoding a triangulation as a permutation of its point set, Proc. of the Ninth Canadian Conference on Computational Geometry, pp. 39-43, Ontario, August 11-14, 1997.

[3] F. Evans, S. Skiena, and A. Varshney, Optimizing Triangle Strips for Fast Rendering, Proceedings, IEEE Vizualization'96, pp. 319--326, 1996.

[4] S. Gumhold and W. Strasser, Real Time Compression of Triangle Mesh Connectivity. Proc. ACM Siggraph 98, pp. 133-140, July 1998.

[5] A.Itai and M. Rodeh, Representation of Graphs, Acta Informatica, No. 17, pp. 215-219. 1982.

[6] K. Keeler and J. Westbrook, Short Encodings of Planar Graphs and Maps, Discrete Applied Mathematics, No. 58, pp. 239-252, 1995.

[7] D. Kirkpatrick, Optimal search in planar subdivisions, SIAM Journal on Computing, vol **12**, pp. :28-35, 1983.

[8] D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its applications. SIAM J. on Computers, 6:594-606, 1977.

[9] M. Naor, Succinct representation of general unlabeled graphs, Discrete Applied Mathematics, vol. 29, pp. 303-307, North Holland, 1990.

[10] M. R. Nelson, LZW Data Compression, Dr. Dobb's Journal, October 1989.

[11] J. Rossignac, Edgebreaker: Compressing the incidence graph of triangle meshes, GVU Technical Report GIT-GVU-98-35, Georgia Institute of Technology, http://www.cc.gatech.edu/gvu/reports/1998.

[12] J. Rossignac, 3D Geometry Compression: Just-in-time upgrades for triangle meshes, in *3D Geometry Compression,* Course Notes 21, Siggraph 98, Orlando, Florida, July 18-24, 1998.

[13] J. Snoeyink and M. van Kerveld, Good orders for incremental (re)construction, Proc. ACM Symposium on Computational Geometry, pp. 400-402, Nice, France, June 1997.

[14] G. Taubin and J. Rossignac, Geometric Compression through Topological Surgery, ACM Transactions on Graphics, Volume 17, Number 2, pp. 84-115, April 1998.

[15] G. Taubin, W. Horn, F. Lazarus, and J. Rossignac, Geometry Coding and VRML, Proceedings of the IEEE, pp. 1228-1243, vol. 96, no. 6, June 1998.

[16] G. Taubin and J. Rossignac, *3D Geometry Compression*, Course Notes 21, Siggraph 98, Orlando, Florida, July 18-24, 1998.

[17] C. Touma and C. Gotsman, Triangle Mesh Compression, Proceedings Graphics Interface 98, pp. 26-34, 1998.

[18] G. Turan, Succinct representations of graphs, Discrete Applied Math, 8: 289-294, 1984.

[19] W.T. Tutte, The Enumerative Theory of Planar Graphs. In *A Survey of Computational Theory*, J.N. Srinivasan et al. (Eds.). North-Holland, 1973.

[20] A. Szymczak and J. Rossignac, Grow&Fold: Compression of Tetrahedral Meshes, GVU Technical Report GIT-GVU-99-02, Georgia Institute of Technology, February 99. http://www.cc.gatech.edu/gvu/reports/1999.

[21] T. Welch, A Technique for High-Performance Data Compression, Computer, June 1984.

[22] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, May 1977.