Graphical Models **63,** 263–275 (2001)
doi:10.1006/gmod.2001.0555, available online at http://www.idealibrary.com on IDEAL®

Efficient Coding of Nontriangular Mesh Connectivity

Boris Kronrod and Craig Gotsman

Computer Science Department, Technion-Israel Institute of Technology, Haifa 32000, Israel
E-mail: kronrod@cs.technion.ac.il, gotsman@cs.technion.ac.il

Received September 9, 2001

We describe an efficient algorithm for coding the connectivity information of general polygon meshes. In contrast to most existing algorithms which are suitable only for triangular meshes, and pay a penalty for treatment of nontriangular faces, this algorithm codes the connectivity information in a direct manner. Our treatment of the special case of triangular meshes is shown to be equivalent to the Edgebreaker algorithm. Using our methods, any triangle mesh may be coded in no more than 2 bits/triangle (approximately 4 bits/vertex), a quadrilateral mesh in no more than 3.5 bits/quad (approximately 3.5 bits/vertex), and the most common case of a quad mesh with few triangles in no more than 4 bits/polygon.

Key Words: mesh compression; coding.

1. INTRODUCTION

The subject of efficient coding of polyhedral meshes has attracted much interest during recent years, mainly due to the increasing popularity of 3D content on the Web. Without efficient coding methods, it is difficult to transmit these data in reasonable time frames.

Typical 3D meshes consist of connectivity information and geometric information. Many algorithms have been designed to code the connectivity information, e.g., the Topological Surgery algorithm of Taubin and Rossignac [11], the algorithm of Touma and Gotsman [12], the Edgebreaker algorithm of Rossignac [9], its refinement by King and Rossignac [6], and others by Gumhold and Strasser [2] and Bajaj *et al*. [1]. The common denominator of these algorithms is that they all deal strictly with inputs which are triangulated meshes, i.e., all mesh faces are triangles. This is crucial for the correctness of the algorithms, and their extension to handle nontriangulated meshes without explicitly triangulating them is not obvious. Hence, in practice, nontriangular meshes are coded by triangulating them, coding the result using one of the methods mentioned above, and storing additional information describing the extra edges introduced during the triangulation stage. These edges

1524-0703/01 $35.00

are discarded after decoding. Ironically, this means that the code of a nontriangular mesh might be larger than that of the triangulated version, instead of being shorter, as less connectivity information is present. An exception is the algorithm of King *et al.* [7] for coding quadrilateral meshes, which is based on the Edgebreaker algorithm [9] for triangle meshes. This method implicitly triangulates each quadrilateral to two triangles and uses sequences of the five basic Edgebreaker symbols (CLERS) to code the different possibilities which then arise. The only algorithm which extends naturally to nontriangular meshes containing polygons with more than four edges is that of Li and Kuo [8], using dual-graph methods, but this yields codes whose length could be unbounded.

This paper describes a general and direct method for coding the connectivity of any nontriangular mesh with an upper bound on the resulting code length. For the special case of a triangular mesh, it reduces to the Edgebreaker algorithm [9] and its improvement [6], which bound the code length from above by 4 and 3.67 bits/vertex, respectively. For the special case of a quadrilateral mesh, it bears some similarity to that of King *et al.* [7].

Our algorithm works, similarly to other mesh connectivity coding algorithms, by maintaining a *cut-border* of edges, which is extended by removing one polygon at a time from the mesh until the mesh is empty. The resulting code consists of information identifying the manner in which the removed polygon lies with respect to the cut-border and the number of edges (the *degree*) of the polygon. As we shall see, this information is sufficient to reconstruct the mesh by generating one polygon at a time. A key to the correctness of our algorithm is the observation that the number of ways that a k-gon can interact with the cut-border is finite and depends only on k, *and not on the size of the cut-border*, and there exists a simple algorithm to translate the interaction type into a unique index. We show that another important piece of information, namely the offset (location) of the polygon along the cut-border, is implicit in the other information, and so does not need to be part of the code.

While this work was in progress, Isenburg [3] proposed an algorithm for coding triangular mesh connectivity and later extended it with Snoeyink [4] to direct coding of nontriangular mesh connectivity. Their Face-Fixer algorithm is an *edge-based* approach, generating a symbol for each mesh edge, in contrast to ours, which is a *face-based* approach, generating a symbol only for each mesh face. This means that the main entity coded in their scheme is the edge and its relation to the rest of the mesh, whereas we deal with the mesh faces (polygons). For a mesh consisting of k-gons containing v vertices, it is possible to show that our algorithm generates in the *worst* case $2\lceil \log_2 P_k \rceil/(k-2)$ bits per vertex, and theirs generates in the *worst* case $(3k-4)/(k-2)$ bits per vertex for large meshes. P_k is $O[((1+\sqrt{5})/2)^{2k}]$, as will be shown in Section 2.2. For small values of k (e.g., $k = 3, 4, 5$), our algorithm is more efficient in the worst case. For large, but less common values of k, our algorithm still has a slight advantage, generating 2.77 bits per vertex in the worst case, while Face-Fixer generates 3 bits per vertex in the worst case.

We present a one-pass decoding algorithm inspired by the linear-time Spirale Reversi decoding algorithm [5] for the Edgebreaker coder [9]. Our coder is also able to handle meshes with topological iregularities, such as holes and handles.

This paper is organized as follows: Section 2 introduces some basic terminology and observations. Section 3 describes the encoding procedure and Section 4 the decoding procedure. Section 5 details how to encode and decode the relationship between a mesh polygon and the cut-border. We show how to bound the length of the codes generated by this algorithm using carefully designed codebooks in Section 6. We extend the basic algorithm

to handle more topologically complex meshes in Section 7. Experimental results from an implementation of our algorithm are presented in Section 8, and we conclude in Section 9.

2. BASICS

2.1. *Definitions*

Before we proceed, a number of terms used by our algorithms must be defined. See Fig. 1 for an illustration.

Cut-border: A cycle of manifold edges (edges with exactly two neighboring faces) in a mesh. For the encoder, unprocessed faces are inside the cut-border. For the decoder, unprocessed faces are outside.

Gate: The edge on the cut-border incident on the next polygon to be processed.

Touching edge: An edge of the processed face which coincides with some edge on the cut-border.

Free edge: The opposite of a touching edge.

Touching vertex: A vertex of the processed face which coincides with some vertex on the cut-border.

Free vertex: The opposite of a touching vertex.

Polygon interaction type: Label sequence for a polygon, indicating whether each vertex and edge (of the polygon) is free or touching a given cut-border.

Gap: The sequence of free edges between two adjacent touching vertices of a polygon.

2.2. *Some Topological Observations*

A key component of our coding algorithms is the observation that a k-gon may interact (in terms of its vertex–edge free–touching label sequence) with a cut-border in a finite number of ways and these may be enumerated. We will call this number P_k. By enumeration, it turns out that $P_3 = 5$ and $P_4 = 13$. Figure 2 illustrates these different possibilities (for a triangle or quadrilateral mesh). Note that P_k depends only on k.

The precise number of interaction types—P_k—is important for our coding and may be calculated using the following theorem:

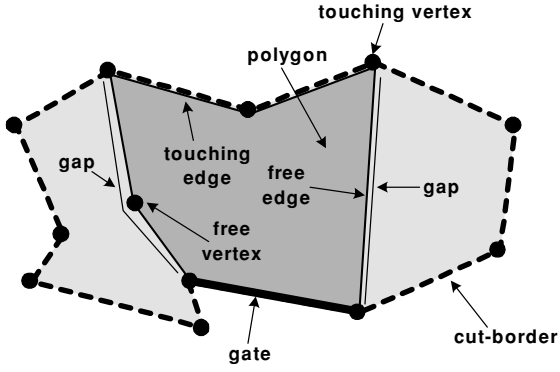

FIG. 1. Terminology of a polygonal face interacting with a mesh cut-border.

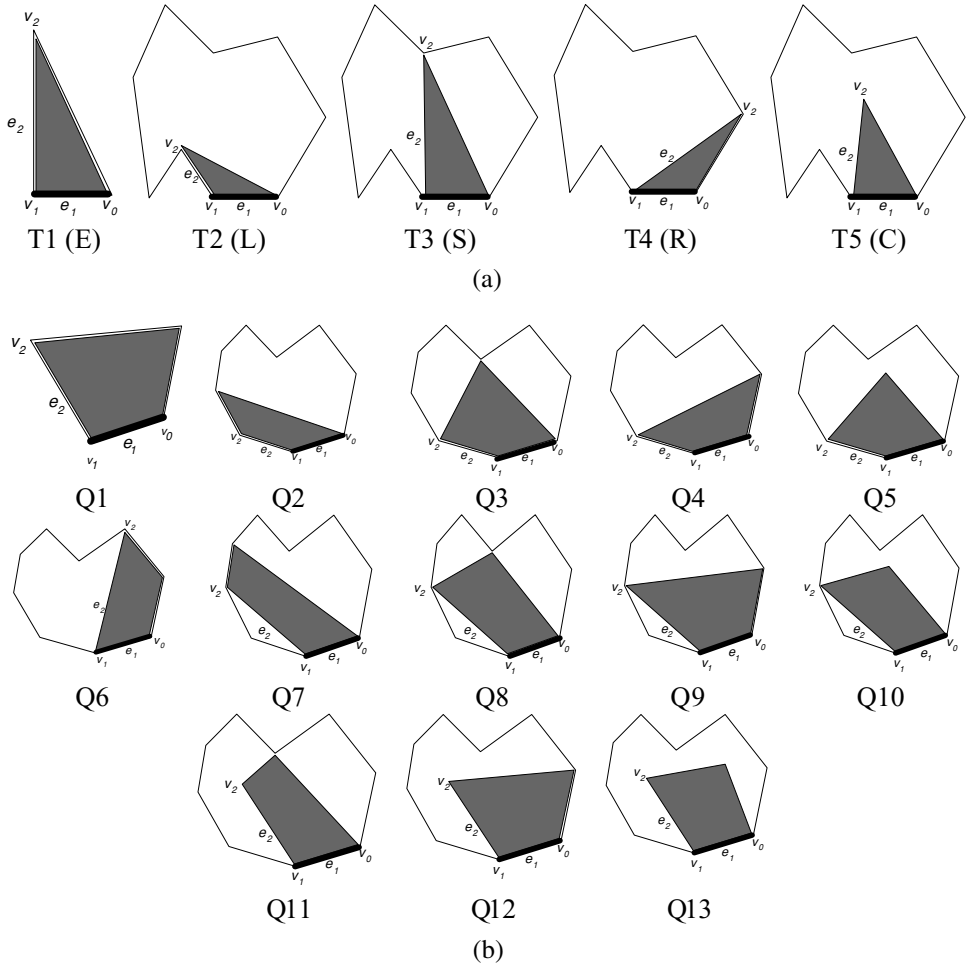

FIG. 2. (a) All possible interactions between a triangle and a cut border. The thick edge is the gate. Case T3 (having 2 gaps) generates a new cut-border during coding. Marked in parentheses are the analogous symbols of the Edgebreaker code. (b) All possible interactions between a quad and a cut-border. The thick edge is the gate. Cases Q3, Q7, Q9, Q10, and Q11 (having two gaps) generate a new cut-border, and case Q8 (having three gaps) generates two new cut-borders.

THEOREM. *For* $k > 4$, $P_k = 3P_{k-1} - P_{k-2}$.

Proof. Consider a k-gon interacting with a cut-border such that e_1—the gate—is a touching edge. There are exactly three distinct possibilities to relate P_k to P_{k-1} (see Fig. 3):

1. The edge e_2 is touching (Fig. 3a), and hence also v_2, but e_3 may or may not be touching. In any case, we may chop off the "ear" consisting of the triangle v_0–v_1–v_2 and remain with a polygon of $k-1$ edges. Doing this, we find that the original k-gon interacts with the original cut-border in precisely the same number of ways that the new (chopped) $(k-1)$-gon interacts with the new (chopped) cut-border. Note that the sizes of the cut-borders are not important. Thus the contribution of this case to P_k is P_{k-1} possibilities.

2. The edge e_2 is free, but the vertex v_2 is touching (Fig. 3b) and again e_3 may or may not be touching. The same argument as in the previous case shows that here too the contribution of this case to P_k is P_{k-1} possibilities.

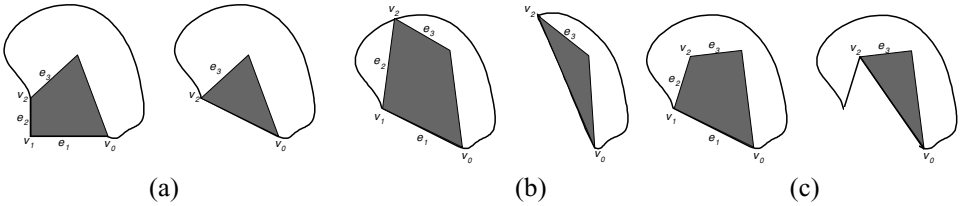

FIG. 3. Three cases reducing P_k to P_{k-1}. Left is the k-gon, and right is the reduction to a $(k-1)$-gon. (a) e_2 is a touching edge. (b) e_2 is a free edge and v_2 a touching vertex. (c) e_2 is a free edge and v_2 a free vertex.

3. The edge e_2 is free, and so is the vertex v_2 (Fig. 3c). This implies that the edge e_3 is free (otherwise v_2 would not be free), as opposed to the previous two cases. v_3 may or may not be touching. Thus the number of interaction types between the rest of the polygon edges and the cut-border is P_{k-1} less those for a polygon of degree $k-1$ with a touching edge (as enumerated in case 1), because of the constraint on e_3. So the contribution of this case to P_k is $P_{k-1} - P_{k-2}$ possibilities.

Hence $P_k = P_{k-1} + P_{k-1} + P_{k-1} - P_{k-2} = 3P_{k-1} - P_{k-2}$. ■

Similar observations were independently made by King *et al.* [7] in the context of their method of coding quadrilateral meshes by triangulation. They further showed that $P_k = F(2k-1)$, where F is the Fibonacci sequence.

The well-known relation between the Fibonacci sequence and the golden ratio

$$F(n) \sim \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

implies that the number of bits required to code these possibilities for large k is $2\log_2((1+\sqrt{5})/2)^k = 2.77\,k$.

Our observations indicate that every type of interaction between a k-gon and a cut-border can be associated with an index in the range $[1..P_k]$. In Section 5 we describe a deterministic algorithm which finds this association and can index any interaction type in an invertible manner. The algorithm uses the inductive nature of the calculation of P_k.

3. ENCODING

Assume the mesh has a boundary. If it does not (i.e., it is a closed mesh), it is possible to artificially create a boundary by removing one polygon. This boundary will function as the initial cut-border from which polygons will be removed one by one. Removing polygons will extend the cut-border. Encoding the mesh is a simple matter of traversing the polygons of the mesh around the cut-border and recording a code of the interaction type of the current polygon with the cut-border and also the polygon's degree (number of edges). The polygon is then removed from the cut-border, the cut-border updated, and the procedure repeated. If the interaction type of the removed polygon contains gaps (i.e., the polygon has at least one touching vertex), the cut-border will split into two or more cut-borders.

An important feature of the algorithm implementation is that a cut-border is not stored in an explicit manner, since at any particular moment only the gate of the cut-border is relevant. The cut-borders are maintained implicitly as a stack of gate edges. Every traversed polygon

creates one or more cut-borders (including the one it participated in) and pushes on the stack exactly one gate for each cut-border generated. This way the current gate of the active cut-border is always at the top of the stack, until that cut-border shrinks to nothing. The procedure terminates when the entire mesh is reduced to nothing. The following summarizes the encoding algorithm:

1. Create a cut-border by identifying the mesh boundary or by removing one mesh polygon.
2. Initialize an empty stack SG of gates, and push onto it some edge of the initial cut-border.
3. While SG is not empty:
4. Pop an edge AG from SG. AG is now the active gate. The cut-border to which AG belongs is called the active cut-border (ACB). The single face P connected to AG (and not yet coded) will be the currently processed polygon.
5. Write the degree (number of edges) of P and its interaction type with ACB onto the encoder output stream (see Section 5).
6. For every gap G on P (as defined in Section 2.1) in counterclockwise order from AG, push the edge of G furthest from AG onto SG. If there is more than one gap (i.e., there is at least one touching vertex), this effectively splits the ACB into a number of smaller cut-borders,
7. Remove P from the mesh and modify ACB accordingly.
8. Endwhile

A sample run of the encoding procedure appears in Fig. 4.

As we will see in the next section, the output of the encoder is sufficient to reconstruct the connectivity data of the mesh. The time complexity of each coding iteration is O(degree of P); hence the complexity of the entire runtime is linear in the size of the mesh.

4. DECODING

It is possible to decode the compressed connectivity information in a two-pass manner, similar to the original Edgebreaker decoder [9]. This, however, has superlinear complexity. Instead, we use another method, modeled after the Spirale Reversi decoder [5], which is both simpler and has linear-time complexity. This process is the reverse of the encoding process; i.e., mesh polygons will be reconstructed in an order opposite to which they were encoded. This contrasts with the Wrap and Zip decoder [10], which also has linear runtime, but operates in the same order as the encoder.

In the encoding process, every traversed face created one or more cut-borders (including the one it participated in) and pushed on the stack exactly one gate for each cut-border generated. The interaction type defines exactly how this polygon is connected to all generated cut-borders. Hence, if the decoder knows the connectivity in the interior of the cut-borders *after* the encoding step and the nature of the interaction between the encoded polygon and the active cut-border, it is easy to reconstruct the connectivity of the interior of the active cut-border as it was *before* this encoding step.

Another way of saying this is: it is possible to decode a polygon after all its gates to cut-borders used during encoding (and polygons based on them) are decoded. This is ideal for postfix recursive implementation. The recursion terminates when the interaction between the coded polygon and the cut-border is such that all polygon edges are touching; i.e., no new cut-border gates are generated (the T1, Q1, etc. symbols).

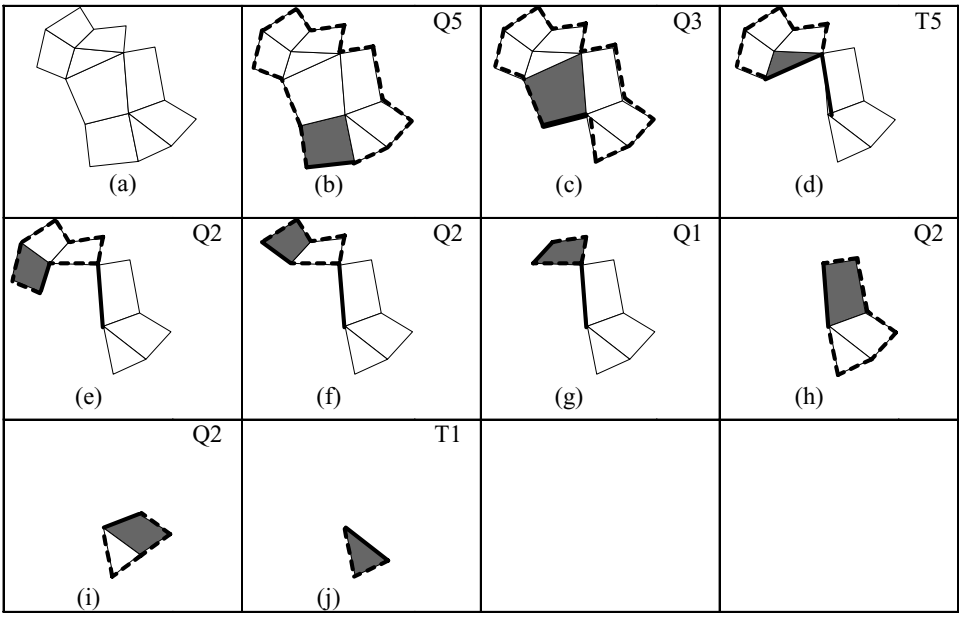

FIG. 4. Code of mesh containing both triangles and quadrilaterals. Dashed edges are active cut-borders. Solid edges are gates in stack. Gray polygon is that being coded. (a) Input mesh. (b) An arbitrary edge of the cut-border is selected as a gate, and the quad it defines has interaction type Q5 with the cut-border; hence Q5 is written to the code output. (c) Two free edges and no free vertices in a quad define two gaps, which results in a split into two cut-borders. The gate of the second (nonactive) one is pushed first onto the stack. (d), (e), (f) Similar to (b) with interaction types T5, Q2, and Q2. (g) No free edges to push onto stack—active cut-border exhausted. (h) Gate pushed in (c) popped from stack and new active cut-border initiated. (i) Similar to (b) with interaction type Q2. (j) No free edges and stack is empty—terminate. Final code is Q5Q3T5Q2Q2Q1Q2Q2T1.

The complexity of each step is O(degree of decoded polygon); thus the overall time complexity is linear in the mesh size. Figure 5 shows how to decode the mesh of Fig. 4 in this manner.

5. CODING INTERACTION TYPES

In our coding algorithm we assumed it is possible to index the interaction between a polygon and a cut-border. This implies that the interactions have some orderly structure that is easy to capture. In Section 2.2 we showed that this is indeed the case and here we show how to exploit this structure in order to code. We now describe an invertible mapping of an interaction type onto the range $\{1..P_k\}$, where P_k is the number of all possible interactions between a k-gon and a cut-border, as defined in Section 2.2.

5.1. *Mapping Interaction Type to Index*

Assume we have a k-gon and its interaction type, i.e., a labeling of all k vertices and edges in the polygon relative to the cut-border: "free" or "touching" in the VertexLabel $(1..k)$ and EdgeLabel $(1..k)$ arrays, respectively.

The following pseudo-code computes the interaction index associated with the interaction type, assuming the integer function (table) $P(k)$ is known (where $P(1) = 1$ and $P(2) = 2$):

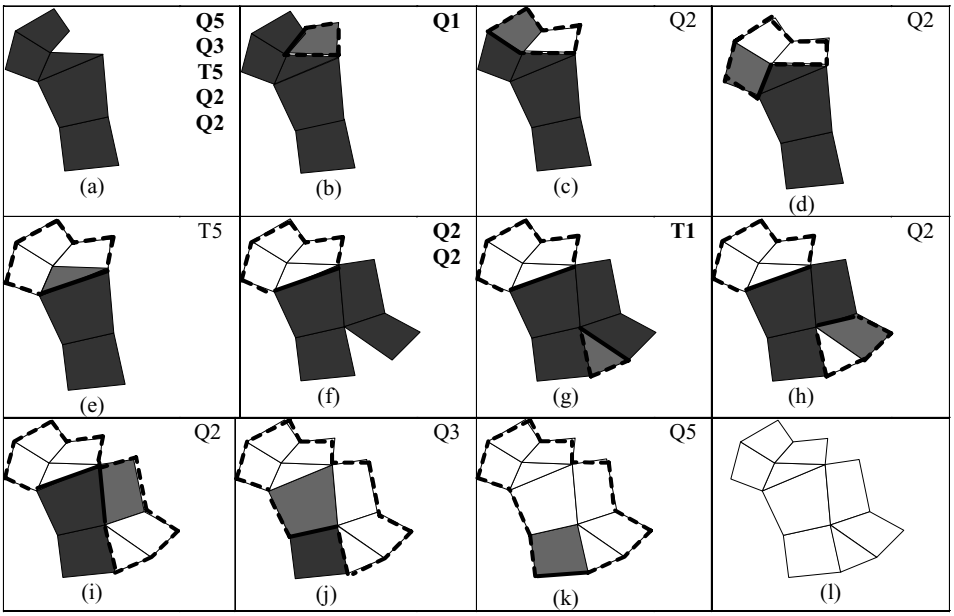

FIG. 5. Decoding the mesh of Fig. 4 using the one-pass decoder. White polygons have already been decoded. The gray light polygon is the one being decoded. Dark polygons are those whose decoding has started because of the postfix recursion, but not yet completed. The dashed edges are cut border(s). The thick edge is the gate. Codes in boldface are those read from input into stack. Codes in regular font are those popped from the stack. (a) Codes read from input into stack, resulting in strip of polygon templates. (b) Q1 read from input, adding another quad to strip. No free edges means that codes will now be popped from stack, and polygons (including this) generated. Note that the final connectivity of this polygon has not yet been resolved (it is not yet connected to the triangle). (c), (d) Q2 popped from stack, and one polygon generated per code. (e) T5 popped from stack. Two free edges mean that this is a cut-border split, so more codes must be read from the input. Triangle is now connected to neighboring quad. (f) Q2, Q2 read from input, and two polygon templates generated, continuing first template strip. (g) T1 read from input. No free edges means that codes will now be popped from stack and polygons generated. (h), (i) Q2, Q2 popped from stack and two polygons formed. (j) Q3 popped from stack. Two free edges means that it connects this cut-border to previous one. (k) Q5 popped from stack. Free vertex means it is connected to both neighboring quads. (l) Stack empty. Final mesh.

```
function index(k,VertexLabel,EdgeLabel)
  if EdgeLabel(k)=="free" then index := 2
    else index := 1;
  for (i=k-1;i>1;i--)
    if EdgeLabel(i)=="free" & VertexLabel(i)=="touching"
        then index += P(k-i+1);
      if EdgeLabel(i)=="free" and VertexLabel(i)=="free"
        then index += 2*P(k-i+1) - P(k-i);
  end for
  return index;
```

5.2. *Mapping Index to Interaction Type*

The inverse of the algorithm given above proceeds as follows. Assume we know the interaction index and the degree k of the k-gon. The procedure should label each edge and each vertex of the k-gon with "free" or "touching."

By definition, the edge e_1 is touching, as well as vertices v_0 and v_1. The following pseudocode labels all remaining edges and vertices (i.e., generates the VertexLabel and EdgeLabel arrays):

```
function Label(k,index,VertexLabel,EdgeLabel)
  EdgeLabel(1) := "touching";
  VertexLabel(1) := "touching";
  VertexLabel(2) := "touching";
    for (i=1;i<k−1; i++)
      if index <= P(k−i) then
        EdgeLabel(i+1) := "touching";
        VertexLabel(i+1) := "touching";
      endif
      if index > P(k−i) and index <= 2*P(k−i) then
        EdgeLabel(i+1) := "free";
        VertexLabel(i+1) := "touching";
        index −= P(k−i);
      endif
      if index >2*P(k−i) then
        EdgeLabel(i+1) := "free";
        VertexLabel(i+1) := "free";
        index −= 2*P(k−i)+P(k−i−1);
      endif
    endfor
    if index == 1 then
      EdgeLabel(k) := "touching"
    else
      EdgeLabel(k) := "free";
```

6. CODEBOOKS AND BOUNDS

Although it is possible to code polygon meshes using our methods, and then apply entropy coding to take advantage of the different frequencies of the resulting symbols, it is also possible to use a predefined codebook and, due to the topological properties of manifold meshes, achieve an upper bound on the total length of the mesh code. This optimization for a triangle mesh is similar to that described in [6], so we will elaborate on quad meshes, and a quad mesh with a minority of triangles, which is a case frequently encountered in real-world models.

6.1. *Quad Meshes*

A mesh containing only quads admits 13 different interaction types, which, when coded naively using a fixed-length code, require $\lceil \log_2 13 \rceil = 4$ bits/quad. Fortunately, not all the interactions occur with equal frequencies, so a more efficient variable-length prefix code may be used, reducing the total code length to less than 3.5 bits/quad. We propose that summarized in Table 1, with code lengths of 2, 3, and 5 bits. To see why the total code length is as claimed, note that any quad from group C—those which do *not* introduce a new

TABLE 1
Codes for Quad Mesh

Code group	Description	Interaction	Code
A	Quad with two free vertices	Q13	00
B	Quad with one free vertex	Q5	010
		Q10	011
		Q11	100
		Q12	101
C	Quad with no free vertices	Q1	11000
		Q2	11001
		Q3	11010
		Q4	11011
		Q6	11100
		Q7	11101
		Q8	11110
		Q9	11111

vertex into the mesh—is present in the mesh iff a quad from group A—those introducing *two* new vertices into the mesh—is present; thus the total code length for that pair of quads is 7 bits, or 3.5 bits/quad. When a quad from group B is present, it introduces one new vertex and requires only 3 bits/quad.

6.2. *Quad Meshes with Few Triangles*

A mesh containing both quads and triangles admits $5 + 13 = 18$ different interaction types; hence naïve coding will require $\lceil \log_2 18 \rceil = 5$ bits/poly. Similarly to the case of a pure quad mesh, a more efficient variable-length code may be used, reducing the total code length to less than 4 bits/poly. We propose that summarized in Table 2, with code lengths of 3, 4, or 5 bits. To see why the total code length is as claimed, note that a quad from group A is present only if a quad from group C is also present or two triangles from group E are present. Hence the average code length per polygon is $(3 + 5)/2 = 4$ bits in the first case and $(3 + 2 \times 4)/3 = 3.667$ bits in the second case. Similarly, a triangle from group D is present only if there is also a quad from group C or a triangle from group E. Here the average code length per polygon is $(2 \times 3 + 5)/3 = 3.667$ bits in the first case and $(3 + 4)/2 = 3.5$ bits in the second case.

7. TOPOLOGICAL IRREGULARITIES

Our coding algorithm, as described in the previous sections, is capable of handling only closed manifold meshes of zero genus, i.e., those topologically equivalent to a sphere. This section describes simple extensions of our basic algorithms in order to handle more complex topologies.

7.1. *Holes*

Holes occur in a mesh with boundaries. For example, a manifold mesh with one boundary is topologically equivalent to a disk. A simple way to encode this information is to treat

TABLE 2
Codes for Mixed Tri/Quad Mesh

Code group	Description	Interaction	Code
A	Quad with two free vertices	Q13	000
B	Quad with one free vertex	Q5	0100
		Q10	0110
		Q11	1000
		Q12	1010
C	Quad with no free vertices	Q1	11000
		Q2	11001
		Q3	11010
		Q4	11011
		Q6	11100
		Q7	11101
		Q8	11110
		Q9	11111
D	Tri with one free vertex	T5	001
E	Tri with no free vertices	T1	0101
		T2	0111
		T3	1001
		T4	1011

the hole as a missing mesh polygon with a relatively large number of edges. The index of this missing polygon in the mesh must also be coded in order to remove it from the mesh during decoding.

A more compact, but elaborate, scheme to encode boundary information is to maintain a separate binary code stream—the *BoundaryStream*. When a polygon containing a boundary edge is first encountered during encoding, the following information is written to that stream:

1. The index of the coding step at which this event occurred.
2. The number of edges in the boundary.
3. For each free vertex and free edge on the polygon (relative to the active cut-border), the index of the vertex or edge on the boundary, if they coincide, or -1 if they are not incident on the boundary.

After this, the hole boundary is considered a regular mesh polygon and all its free edges are pushed in clockwise order onto the stack. There is no need to check whether the boundary touches any other cut-border edges.

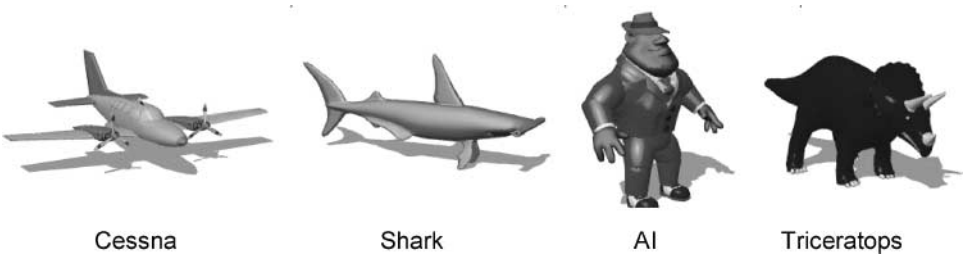

FIG. 6. Some of the 3D models used in our experiments. Models and images copyright Viewpoint Inc.

TABLE 3
Experimental Comparison of Coding Algorithms

Model	Number of vertices	Number of faces	KG (bpv)	IS (bpv)
Triceratops	2,832	2,834	2.48	2.12
Cessna	3,745	3,927	2.71	2.84
Beethoven	2,655	2,812	2.95	2.89
Sandal	2,636	2,953	3.42	2.60
Shark	2,560	2,562	2.35	1.67
Al	3,618	4,175	3.19	2.93
Tommygun	4,171	3,980	2.98	2.61
Cow	2,904	5,804	2.15	2.21

Note. The cow model contains only triangles and the remainder contain a mix of triangles, quads, and higher degree faces.

During decoding, when the decoding step whose index was stored in the BoundaryStream is reached, the active cut-border is expanded as usual. The newly created boundary is then considered a regular mesh polygon and the algorithm proceeds as usual.

The number of holes in a typical mesh is small and the polygon degrees bounded; hence the code on the BoundaryStream is expected to be very short compared to the rest of the connectivity code.

7.2. *Handles*

A mesh of nonzero genus contains *handles*, which cannot be coded by our algorithm as described above. However, the case of handles may be reduced to the case of holes by cutting the mesh along each handle, resulting in the end in two holes per handle.

In order to perform this reduction, the encoder should be capable of detecting handles and the edge cycles which sever them from the rest of the mesh. This is possible when the current polygon touches a boundary which does not contain the active gate.

8. EXPERIMENTAL RESULTS

We have implemented the algorithms described in this paper and run them on some real-world models, mostly from Viewpoint Inc. See Fig. 6 for some examples. Table 3 summarizes some of our results (denoted by KG) and compares them to those of Isenburg and Snoeyink [4] (denoted by IS). The results include Huffman entropy coding of the basic code sequences.

While, on the average, our codes seem to be slightly longer than those of Isenburg and Snoeyink, our algorithm, in our opinion, is potentially simpler to describe and implement.

9. CONCLUSION

This paper has described a direct mesh connectivity coding algorithm for general non-triangular meshes and provided explicit codes for the cases of a pure quad mesh, or a quad

mesh with a minority of triangles. The case of a triangle mesh with a minority of quads may be treated similarly. As with Edgebreaker, the code lengths are bound from above, resulting in efficient codes which are probably quite close to the theoretical lower bound. In the case of meshes containing faces with a variety of degrees, much of the code will be dedicated to specifying the face degrees. It seems that it may be possible to reduce the size of this portion of the code. This is a topic for future work.

ACKNOWLEDGMENTS

Thanks to Martin Isenburg and the reviewers for helpful comments related to this work.

REFERENCES

1. C. Bajaj, V. Pascucci, and G. Zhuang, Single resolution compression of arbitrary triangular meshes with properties, *Comput. Geom.* **14**, 1999, 167–186.
2. S. Gumhold and W. Strasser, Real time compression of triangle mesh connectivity, *in Proceedings of SIGGRAPH '98, 1998*, pp. 133–140.
3. M. Isenburg, Triangle strip compression, in *Proceedings of Graphics Interface, 2000*, pp. 197–204.
4. M. Isenburg and J. Snoeyink, Face fixer: Compressing polygon meshes with properties, in *Proceedings of SIGGRAPH 2000*, pp. 263–270.
5. M. Isenburg and J. Snoeyink, Spirale Reversi: Reverse decoding of the Edgebreaker encoding, in *Proceedings of the 12th Canadian Conference on Computational Geometry, 2000*, pp. 247–256.
6. D. King and J. Rossignac, Guaranteed 3.67v bit encoding of planar triangle graphs, in *Proceedings of 11th Canadian Conference on Computation Geometry, 1999*, pp. 146–149.
7. D. King, J. Rossignac, and A. Szymczak, *Connectivity Compression for Irregular Quadrilateral Meshes*, Technical Report GIT-GVU-99-36, GVU, Georgia Inst. of Tech., 1999.
8. J. Li and C.-C. Kuo, A dual graph approach to 3D triangular mesh compression, in *Proceedings of the IEEE International Conference on Image Processing, Chicago, 1998*.
9. J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, *IEEE Trans. Visual. Comput. Graphics* **5**(1), 1999, 47–61.
10. J. Rossignac and A. Szymczak, Wrap and zip: Linear decoding of planar triangle graphs, *Comput. Geom.* **14**, 1999, 119–135.
11. G. Taubin and J. Rossignac, Geometric compression through topological surgery, *ACM Trans. Graphics* **17**, 1998, 84–115.
12. C. Touma and C. Gotsman, Triangle mesh compression, in *Proceedings of Graphics Interface '98, 1998*, pp. 26–34.