

# A study of different approaches to compression of 3D meshes

Akash M Kushal

Department of Computer Science and Engineering  
Indian Institute of Technology, Delhi  
New Delhi 110016

*email: csu99132@cse.iitd.ernet.in*

## 1. Introduction

Polygon meshes, in particular triangle meshes, are increasingly being used as the primary geometric modeling representation for interoperability in a large number of applications in diverse fields such as engineering, manufacturing, entertainment, education, cultural heritage, etc. Hardware accelerators for 3D graphics also provide extensive capabilities for high speed rendering of triangle meshes. In order to capture very fine shape details, these meshes are captured at high resolution and modeled with a large number of triangles, resulting in huge storage requirements and long transmission times. Consequently, triangle mesh compression has got a lot of interest.

## 2. Edge Breaker

The Edgebreaker [1] compression algorithm performs a series of steps. Each step removes one triangle from the current mesh. At each stage, the remaining portion of the mesh is composed of one or several regions, denoted by  $R_i$  which are simple meshes. Technically, each region is the union of triangles of  $T$ , whose interior is contained in one maximally connected component of the interior of the union of the remaining triangles. Note that two regions may share a vertex, but not an edge. The edges bounding each region form a closed polygonal curve, called loop, which has no self-intersections. One edge of each loop is called a gate. A stack contains references,  $S_0, S_1, S_2, \dots$  to all the gates. The top of the stack,  $S_0$ , references the active gate,  $g$ . Let  $R_0$  be the region incident upon  $g$  and let  $B$  denote the bounding loop of  $R_0$ . Note that  $B$  contains  $g$ .

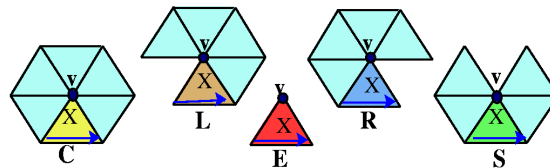


Figure 1: The OP-CODES

At each step, Edgebreaker identifies the unique triangle,  $X$ , that is part of  $R_0$  and is incident upon  $g$ . Let  $v$  be the only vertex of  $X$  that does not bound  $g$ . Edgebreaker analyzes the relation that  $v$  has with respect to  $B$  and  $g$ , distinguishing 5 cases labeled  $C$ ,  $L$ ,  $E$ ,  $R$ , and  $S$  as shown in the figure.

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

```

IF  $v \notin B$  THEN case  $C$ 
ELSE IF  $v$  follows  $g$  THEN IF  $v$  precedes  $g$  THEN case  $E$  ELSE case  $R$ 
ELSE IF  $v$  precedes  $g$  THEN case  $L$  ELSE case  $S$ 

```

Edgebreaker constructs a compression history  $H$  by appending op-codes selected from the set  $C$ ,  $L$ ,  $E$ ,  $R$ , or  $S$  to identify the successive steps that must be used to reconstruct the mesh during decompression. Edgebreaker also builds a list  $P$  of vertex references in the order in which they are reached by  $C$  operations as the third vertex,  $v$ , of the triangle incident upon the gate. This list will define the order in which the vertices will be encoded. The history  $H$  will be compressed using binary codes or any desired compression scheme.

For meshes with boundary,  $P$  is initialized to the references of the vertices of the initial loop  $B$  as they are encountered by walking around it, starting with the end-vertex of the gate.

Decompression performs two traversals of the input stream: Preprocessing computes  $|T|$ ,  $|V_E|$ ,  $|V_I|$ , and the offsets for all the  $S$  operations, which are stored in the offset table  $O$ ; Generation creates the triangles in the order in which they were deleted by the compression process and, for each triangle, stores the labels of its 3 vertices. To compute the correct labels at decompression, the algorithm simply increments an integer vertex-counter,  $c$ , each time we encounter a  $C$  operation and use  $c$  as the label of the new vertex.

## 2.1 Edge Breaker is 2bpv

If we use fixed binary op-codes with 1 bit to encode each  $C$  operation and 3 bits to encode each other operation. The total number of bits needed to encode  $H$  is  $c = |C| + 3(|S| + |L| + |R| + |E|)$ , where  $|X|$  denotes the number of  $X$ -type operations in  $H$ . Because there is a one-to-one association between the vertices of  $V_I$ (internal vertices of the mesh) and the triangles processed by a  $C$  operation, we have  $|C| = |V_I|$ .

Hence,  $|S| + |L| + |R| + |E| = |T| - |C| = |T| - |V_I|$  and  $c = |V_I| + 3(|T| - |V_I|)$ , which yields:  $c = 2|T| + (|T| - 2|V_I|)$ . Given that  $|T| - 2|V_I| = |V_E| - 2$ , we obtain  $c = 2|T| + |V_E| - 2$  where  $|V_E|$  is the number of boundary edges. Consequently, for simple meshes with a relatively simple initial boundary, we have  $|V_E| \ll |V_I|$ , leading to  $|V_E| \ll |T|$ , and to  $c = 2|T|$ . To encode a simple mesh without boundary, such as the entire surface that bounds a manifold 3D solid, it suffices to "cut open" one of its edges, declare it to be the initial loop,  $B$ , and include the encoding of its two vertices at the top of the vertex list, as discussed above. In that case,  $|V_E| = 2$  and  $c = 2|T|$ , which is exactly 2 bits per triangle.

## 3. Advancing Fan-front

The Advancing Fan-front [2] algorithm works like Edge Breaker. The only difference is that it moves into the mesh with fans instead of triangles. The algorithm starts the conquest of a mesh by arbitrarily choosing an edge on the mesh boundary as the initial gate. A fan is formed with the selected gate. Each newly constructed fan has its two extreme fan-front vertices on the boundary and includes all the triangles incident on the start vertex of the gate. The conquest of the fan removes the fan triangles from the mesh, identifies the gate(s) on this fan-front and modifies the mesh boundary by suitably inserting the interior edges on the fan-front. Thus the fan-front is advanced into the mesh. This continues until the whole mesh is conquered.

In complex situations the advancing fan-front may intersect the original mesh boundary splitting the boundary into multiple loops. When this occurs an explicit reference to the position of the intersection

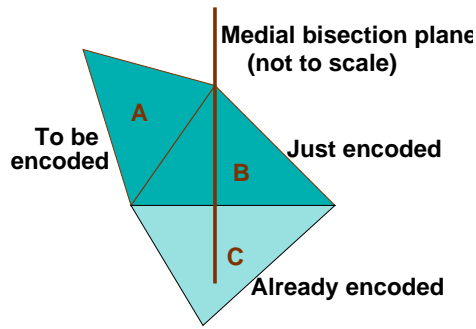


Figure 2: A fundamental observation

vertex in the boundary list must also be sent with the coded data. Then the resulting pieces are conquered one by one.

The algorithm exploits the fact that the number of different types of fans in large meshes is not very large. So, the fans can be encoded efficiently using any variable length encoding or arithmetic encoding.

## 4. Geometry Based Connectivity compression

Most algorithms, try to compress connectivity and geometry separately, though there is invariably a strong correlation between them. Hence connectivity compression, given the geometry, is expected to be better than abstract connectivity compression without any geometry information. This algorithm [4] uses geometry to drive our connectivity encoding process.

Consider figure 2. The vertex  $P$  of triangle  $A$  is on the same *side* of triangle  $B$  as the triangle  $A$  itself. In this case, vertex  $P$  and triangle  $A$  both lie on the *left* of triangle  $B$ . Let us see how this observation helps. Let us reason inductively. Suppose that we have encoded  $C$  already. We have entered triangle  $B$  and just encoded it. In order to encode  $A$  now, we have to specify  $P$  and on which side of  $B$  does  $A$  get placed. Suppose we specify only  $P$ . We can find out on which side of medial-bisection-plane of  $B$  does  $P$  lie. The medial bisection plane is that plane perpendicular to  $B$  and containing its median. This decision may be referred to as a placement query. Very often it turns out that  $P$  and the triangle  $A$  lie on the same side of the medial bisection plane. Thus the knowledge of vertex location of  $P$ , i.e. the *geometry* gives us a clue about the placement of triangles, i.e. *connectivity*. In this case, the *geometry* completely captures the *connectivity* information - specifying vertex  $P$  completely defines the triangle  $A$ .

**Catch** But what if  $P$  lied on the right side of  $B$  as in figure 3 ?. We cannot correctly reconstruct  $A$  just using  $P$ . But consider any point in the hatched area in figure 3. Any point in this area is on the same side of  $B$  as  $A$ . If this point is well-defined, then we can reconstruct  $P$  from it.

We can specify a triangle by specifying two of its vertices and some well-defined point inside it. Specifically, consider a triangle  $PQR$ . Let  $S$  be the midpoint of  $PQ$  and let  $C$  divide the median  $SR$  in the ratio

$$SC : CR = \epsilon : 1$$

The point  $C$  is called the **control-point** of triangle  $PQR$ .

The algorithm first generates a spanning tree on the triangulation. Let triangle  $t = \{x, y, u\}$  be entered from its parent triangle  $p$  via the edge  $xy$ . If  $u$ , the third-vertex of  $t$  is not visited while entering  $t$ , then we generate a control-point for  $t$ , so that  $u$  can be reconstructed (by the decoder).

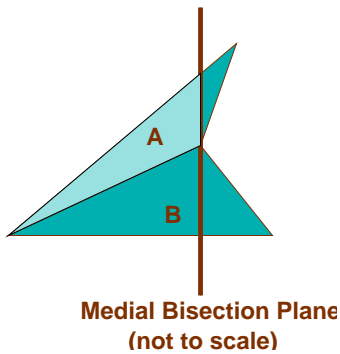


Figure 3: The Catch

When entering a triangle, if its third-vertex is already visited, then the triangle is called a *closing-triangle*. No control-point is generated for these triangles. The geometry of these triangles is known, since all its vertices will be marked visited, when we try to enter them. We will only have to encode its connectivity. This is done by having special codes for these cases.

As the spanning tree is generated, we compute for each triangle, the largest possible  $\epsilon$  value such that the control point is on the same side as the edge to which the new triangle was connected to. Then the global minimum of all these  $\epsilon$  values is taken as the epsilon for the whole mesh. Now, in the second pass, this  $\epsilon$  is used to generate control points. These control points are transmitted instead of the vertex coordinates.

#### 4.1 Error Control

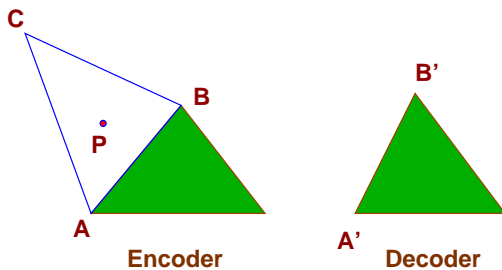


Figure 4: Error control

Consider figure 4. The encoder computes the control point  $P$  from the original vertices  $A$  and  $B$ . But the decoder reconstructs using the previously reconstructed vertices  $A'$  and  $B'$ . To eliminate the error, let the encoder compute control-points using  $A'$  and  $B'$ . The encoder runs a mini-decoder inside it and gets feedback about  $A'$  and  $B'$ .

The algorithm can assure a bits/vertex value of 2.66.

### 5. Valence Driven approach

The valence driven approach [3] is another approach which encodes the connectivity of the mesh in a sequence of valence of vertices. The valence of a vertex is the number of edges incident on the vertex.

## 5.1 The basic algorithm

The encoding process starts with a triangulated mesh, and generates a closed topology by adding and connecting one common dummy vertex to each boundary vertex. An arbitrary seed face  $f_{seed}$  is chosen, its vertices are added to an active (conquest) edge list and their three corresponding valences are output to the code sequence, which will eventually be compressed using conventional entropy encoders. The face  $f_{seed}$ , its vertices and its edges are flagged conquered. The first vertex of  $f_{seed}$  is chosen as the pivot vertex. The conquest can now begin.

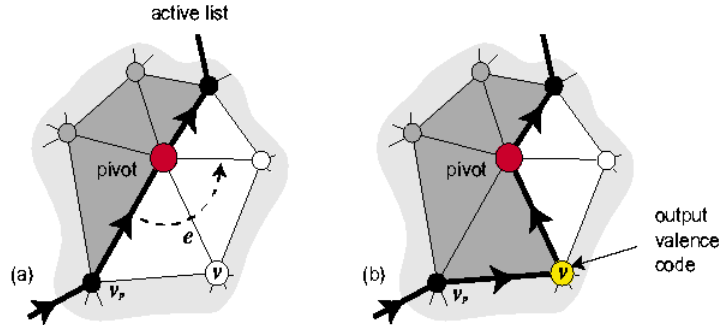


Figure 5: Conquest

## 5.2 Conquest of a next free edge

The current pivot vertex  $v_{pivot}$  tries to conquer its next free edge  $e$  in the counter-clockwise order. Let us call  $v$  the vertex belonging to  $e$ , and  $v_p$  the vertex preceding  $v_{pivot}$  in the active edge list. The vertex  $v$  is inserted before the pivot in the active list, the edges  $e$  and  $(v, v_p)$ , and the face  $(v_p, v, v_{pivot})$  are flagged conquered. According to the current state of  $v$ , four cases may occur:

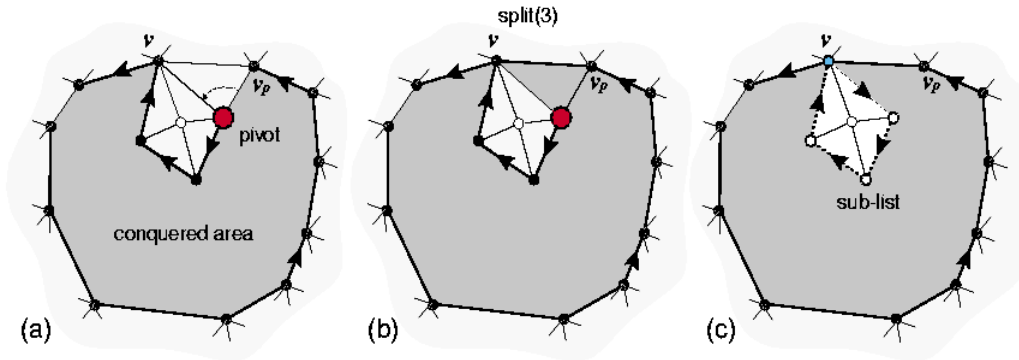


Figure 6: Split code

1.  $v$  has not been conquered yet. It is now flagged conquered, and its valence is output to the code sequence. The active edge list is thus expanded as shown in the figure.
2.  $v$  is a dummy vertex and has not been conquered yet. It is thus flagged conquered, and a code dummy and its valence are output to the code sequence. The active edge list is also expanded.

3.  $v$  has already been conquered and belongs to the active list. To lift the local ambiguity of connectivity, a split code and an associated forward offset (which indicates where  $v$  is located in the current active list) are generated. The active list is then split in two: the (internal) sub-list is pushed to the stack for future treatment, while the conquest continues using the (external) list.
4.  $v$  has already been conquered and belongs to an inactive (sub-)list present on the stack. This case requires the localization of the inactive list in the stack and of  $v$  in that list. A merge code along with its associated list index in the stack and offset in the inactive list are generated. The two lists are merged, leading to an active list containing two occurrences of  $v$ . The inactive list is discarded from the stack. The merge code can only occur when the genus is greater than 0.

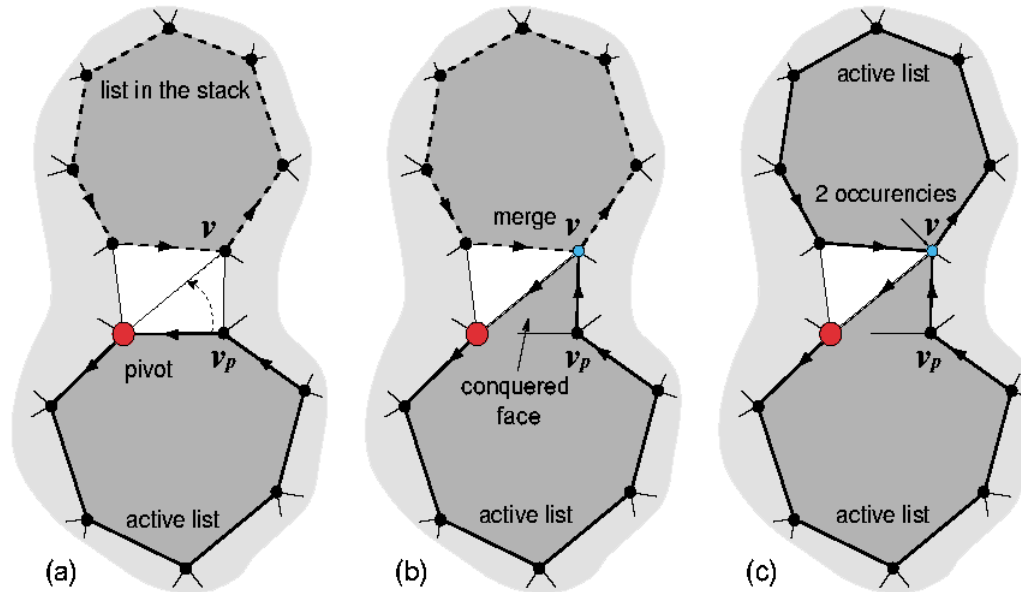


Figure 7: Merge code

It can be shown that just this valence information is enough for the decoder to decode the connectivity of the mesh. Also this technique is asymptotically optimal if we ignore splits. This means that a bound equal to the Tutte's bound can be proven for this algorithm. Tutte in 1962, counted the number of triangulations possible for a mesh with  $n$  points to get a bound of 1.62 bpv for any algorithm. Many heuristics can be used to reduce the number of splits.

## References

- [1] *J. Rossignac*- Edgebreaker: Connectivity Compression for Triangle meshes, *IEEE Transactions on Visualization and Computer Graphics*, 1998
- [2] *S.P. Mudur S Venkata Babji Dinesh Shikhare*- Advancing Fan-front: An efficient Connectivity Compression technique for Large 3D Triangle meshes, *Proc. ICVGIP 2002*
- [3] *Alliez and Desbrun*- Valence-Driven Connectivity Encoding for 3D Meshes *Eurographics 2001*
- [4] *S. Nachiappan Sanjiv Kapoor Prem Kalra*- Geometry Based Connectivity Compression of Triangular Meshes *Proc. ICVGIP 2002*