Advancing Fan-front: An Efficient Connectivity Compression Technique for Large 3D Triangle Meshes

S. P. Mudur
Department of Computer Science
Concordia University
Montreal, Quebec, Canada.
mudur@cs.concordia.ca

S. Venkata Babji Dinesh Shikhare
Graphics & CAD Division
National Centre for Software Technology
Juhu, Mumbai, India.
{babji,dinesh}@ncst.ernet.in

Abstract

In this paper we present a new algorithm for compressing large 3D triangle meshes through the successive conquest of triangle fans. In this algorithm, a triangle fan is the sequence of adjacent triangles incident on a start vertex of a boundary edge called as the "gate." As each fan is conquered, the current mesh boundary is advanced by the fan-front and the gate(s) for conquering the rest of the mesh is(are) identified. The process is continued till the entire mesh is traversed. The mesh is then compactly encoded as a sequence of fan configuration codes. In the earlier reported techniques, the conquest is usually a triangle, a vertex or an edge at a time, and the number of symbols in the resulting encoding is of the order of the total number of triangles, vertices or edges respectively. On the other hand, the number of fans is typically one fourth of the total number of triangles and the number of fan configurations is bounded by the vertex valence variation. While the number of distinct fan configurations depends on the variation in the degree of fans, only a few of these fan configurations occur with high frequency, enabling very high connectivity information compression using range encoding. A simple implementation shows considerable improvements, on the average, in bit-rate per vertex, compared to earlier reported techniques.

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. With recent advances in technologies for acquisition of accurate and highly detailed digital representations of complex 3D shapes, many large and complex 3D polygonal models are being generated. 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, compression of triangle meshes has gained much interest in the recent years. In this paper we present a new 3D triangle mesh connectivity compression algorithm that achieves higher compression ratios than earlier reported techniques, basically using a larger chunk of the mesh, namely a fan, as the unit of encoding.

A *polygon mesh* consists of a set V of vertices, a set E of edges and a set P of polygons. Each vertex is associated with a geometric point position in the 3D Euclidean space, say, $x_i, \{x_i \in \mathbf{E}^3\}$. An edge is represented as a pair (v_1, v_2) and a polygon as a sequence $(v_1, v_2, ..., v_k)$ of vertices. A *triangle mesh* is a special case with all polygons being triangles.

A mesh having f faces, e edges and v vertices satisfies the equation, $f - e + v = 2$ (Euler's relation [8]). When all the faces have at least 3 sides, we can show that $f \leq 2v - 4$ and $e \leq 3v - 6$, with equality if and only if all faces are triangles. For meshes with g handles (genus g) the relation becomes $f - e + v = 2 - 2g$ and the bounds on the number of faces and edges increase correspondingly.

A common scheme for representing and storing polygon meshes is to use a list of vertex geometry coordinates to store the geometry and a list of vertex indices for each face to store mesh connectivity. Edges are implied and not explicitly stored. For a triangle mesh having v vertices and t triangles, this requires approximately $3v$ space to store the vertex coordinates and $3t \log_2 v$ space to store connectivity among the vertices, where $t \approx 2v$ for most triangle meshes. The storage and transmission costs increase non-linearly as the number of vertices increase. Typical large models consist of several hundred thousand triangles and occupy many mega-bytes of storage space. The complexity of these 3D models far exceeds the limits of what can be quickly downloaded at popular connection speeds and what can be rendered for interactive exploration on personal desktops even with graphics acceleration hardware.

Most connectivity compression techniques consider meshes as undirected graphs and cleverly encode the traversals in the graphs so that vertex references are implied, thus avoiding repeated vertex referencing and achieving compression [6, 3, 7, 9, 11, 5]. These traversals not only minimize repeated references to vertices, but also ensure correct reconstruction of the original connectivity. The algorithms differ in the way they span the graphs using connectivity among vertices, edges and faces.

Earlier techniques can be broadly categorized into three classes [4], viz. face-based, edge-based and vertex-based traversals. Rossignac's Edgebreaker [9] and the Cut-border machine of Gumhold and Strasser [6] are face-based, the FaceFixer algorithm of Isenburg and Snoeyink [7] is edge-based and vertex based techniques have been proposed by Touma and Gotsman [11] and Alliez and Desbrun [2]. In these techniques, the number of symbols in the compressed encoding is approximately equal to number of faces, edges and vertices respectively. There could be some additional operators to take care of exceptional situations during the course of the traversal, which may require explicit vertex references. Alliez and Desbrun [2] have argued that such exceptions are sub-linear in order. They also show that a valence-based encoding scheme gets closest to the theoretical limit of 3.245 bits per vertex proved by Tutte [12] and outperforms face-based and edge-based techniques. This is explained by noting that the size of the compressed representation is proportional to the number of operators encoded and the fact that for polygon meshes representing closed manifold surfaces the relation $v < f < e$ holds.

The Advancing Fan-front (AFF) algorithm presented in this paper proposes and uses a *fan-based* traversal of triangle meshes. In almost all the triangle meshes, the number of fans is approximately half the number of vertices and one-fourth the number of triangles (as can be seen from the results). The traversal of the mesh is then represented as a sequence of operators describing, on the average, $v/2$ or $t/4$ fans. While the number of distinct fan configuration codes depends on the variation in the degree of fans, only a few of these fan configurations occur with high frequency, enabling a very compact representation using range encoding.

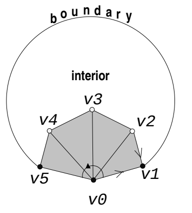
The rest of this paper is organized as follows: Section 2 describes the Advancing Fan-front (AFF) algorithm including details for handling meshes having complex boundary interactions. The connectivity compression results of our implementation are presented in Section 3. We summarize the contributions and conclude the paper in Section 4.

2. Advancing Fan-front Algorithm

2.1. Some Definitions

In a mesh representing a manifold, each edge is either shared by two polygons, called as an *interior edge*, or belongs to a single polygon, called a *boundary edge*. A closed loop formed by linking up such boundary edges forms a *boundary* of the mesh. The boundary may have an orientation, in which case, every boundary edge is a directed edge, with a start and end vertex. Note that a mesh may have multiple boundaries.

We call two polygons as *adjacent polygons* if they share an edge. There exists a *path* between polygons p_i and p_j if there is a sequence of adjacent polygons $p_i, p_1, p_2, ..., p_j$. A maximal subset O_c of the mesh model O is called a *connected component* if there exists a path between every pair of polygons in O_c. Note that a given mesh model may have multiple connected components.



fan-center: `v0`
gate: `(v0, v1)`
fan-front: `(v1,v2,v3,v4,v5)`
vertex bit pattern: `<10001>`
Edge bit pattern: `<0000>`

Figure 1: Fan description.

A *fan* is a polygon that admits a triangulation in which all triangles have a common vertex, called as *fan center*. The other vertices of the polygon are called as *front-vertices*. The front-vertices and the edges between them form a *fan-front*. The number of front-vertices is the *degree* of the fan. In AFF, fans are always constructed such that all the triangles of the fan are incident on the start vertex of a boundary edge called as the *gate*. Criteria for the selection of gates are described in the following subsection.

The following notation is used in the illustrations throughout this paper. We denote an unvisited vertex by a small hollow circle, a visited vertex by a small filled circle. A gate is denoted as an edge with an arrow. The orientation of the fan is shown with a curved arrow around its fan center. Also, we assume that the orientation for mesh boundary and the fans is always counter clockwise.

2.2. Overview of the Algorithm

For the purpose of quickly understanding the AFF algorithm, we first consider the simple input of a consistently oriented triangle mesh forming a single connected component of genus 0, that represents a manifold with at most one boundary. Multiple boundaries can be handled with a little additional effort, closed meshes may be trivially handled by removing one initial triangle to create a boundary, and meshes of higher genus can be handled as described in [1].

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

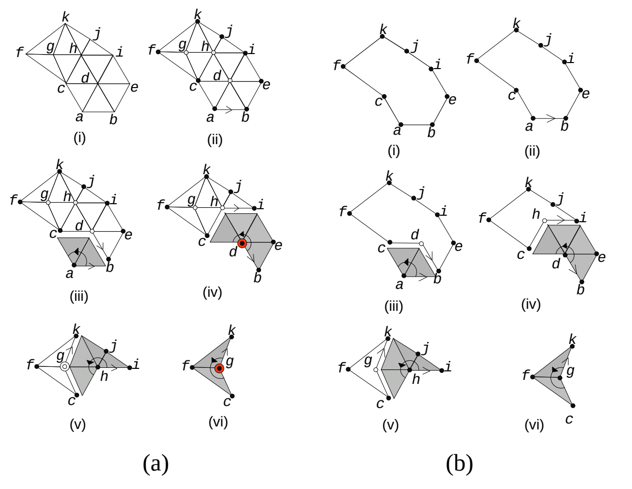


Figure 2: Advancing Fan-front algorithm – a simple example : (a) encoding and (b) reconstruction.

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 process is recursively continued until the entire mesh is conquered.

During this conquest, AFF algorithm records the following information for subsequent reconstruction of the mesh connectivity: (a) vertex coordinate list re-ordered to implicitly reflect the sequence of fan-based conquest, (b) number of vertices on the mesh boundary at start, (c) the start vertex of the starting gate, and (d) a sequence of fan descriptions as given below.

A fan description takes the following form:

<degree><VertexBitPattern><EdgeBitPattern>

where, *VertexBitPattern* is the sequence of binary flags, one for each front-vertex – the flag is 1 if the vertex is on boundary, 0 otherwise; and *EdgeBitPattern* is a sequence of binary flags, one for each edge on the front – the flag is 1 if the edge is on boundary, 0 otherwise (see Figure 1).

Reconstruction of the mesh connectivity proceeds in the same sequence as the conquest of fans in the mesh connectivity.

In the course of advancing the fan-front into the mesh boundary, the process may split the mesh into multiple connected components. In such cases, multiple gates are identified on the front to continue the recursive conquest. *A "gate" is identified as the first interior edge on the fan-front with respect to each connected component adjacent to the currently conquered fan.* When more than one gate is identified on the fan-front, the gates are pushed on a stack and processed recursively. The algorithm carries on its conquest from each of the gates until all the connected components are conquered.

2.3. A Simple Example

We have chosen the simple triangle mesh of Figure 2(a)(i) to illustrate how AFF spans the mesh with successive conquest of fans.

Encoding: Initially, AFF declares all the vertices on mesh boundary (*a, b, e, i, j, k, f, c*) as visited (see Figure 2(a)(i)). Let (*a*,*b*) be the starting gate. (see Figure 2(a)(ii)). Starting with this boundary and the gate, AFF then recursively decomposes the mesh into fans. The fan with the fan-center *a* and fan-front *b, d, c* is spanned in counter-clockwise direction (Figure 2(a)(iii)). After this step, the fan description (<3><101> <00>) is written to the output and vertex *d* is appended to the vertex stream. Next, we advance the fan-front (*b, d, c*) and the new mesh-boundary now becomes (*b, e, i, j, k, f, c, d*). With the earlier mentioned condition for gate, we identify edge (*d, b*) as the next gate. The next fan is conquered using this gate. Further steps in the conquest are illustrated in Figure 2(a).

During this process AFF writes to the output stream the reordered vertex stream as (*a, b, e, i, j, k, f, c, d, h, g*), the number of mesh boundary vertices, *8*, the initial fan-center *a* and a sequence of fan descriptions
<3><101> <00>,
<5><11101> <1100>,
<5><11101> <1100>,
<3><111> <11>.

Note that in the *VertexBitPattern*, the first and last bit will always be 1, since they correspond to extreme boundary vertices on the fan. In this discussion, we retain this redundant representation only for the clarity of expression.

Reconstruction: The reconstruction algorithm restores the initial boundary (*a, b, e, i, j, k, f, c*) of the mesh (see Figure 2(b)(i)), and initial fan-center *a* (see Figure 2(b)(ii)). Now from the first fan description <3><101> <00>, the degree of the fan is 3. So the first $3 - 2 = 1$ vertices are to be restored and assigned to the fan-front, which results in fetching of vertex *d* from the vertex stream. First and last of the fan-front vertices are the next and the previous to the fan-center on the mesh-boundary – so, the fan front is (*b,d,c*). Thus the first fan has been reconstructed. Construction of the fan results in a modified new boundary (*d, b, e, i, k, j, f, c*) (see Figure 2(b)(iii)). The fan description indicates that the next gate is (*d, b*). This successive reconstruction of fans continues till the entire mesh is reconstructed as shown in Figure 2(b).

2.4. An Example with Splits

In the simple example above, advancing the fan-front always resulted in the unconquered part of the mesh remaining as a single connected component. However in more complex situations, as mentioned earlier, advancing of the fan-front into the mesh may result in the fan-front intersect-

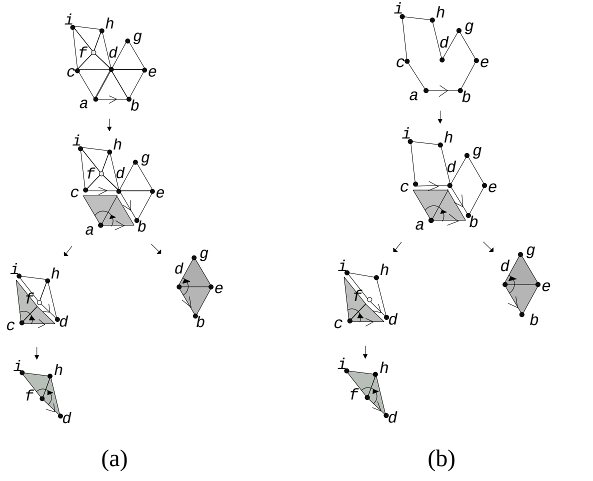


Figure 3: Advancing Fan-Front algorithm – an example with splits: (a) encoding and (b) reconstruction.

ing with the original mesh boundary in such a way as to split the mesh boundary into multiple loops, and as a consequence splitting the unconquered parts of the mesh into multiple connected components. We now examine an example to illustrate how the conquest of fans can result in splits in the boundary and the selection of multiple gates on a fan front.

Encoding: AFF's depth-first traversal of fans is shown in Figure 3(a). Note that the removal of the first fan from the mesh results in creation of two boundaries, (*b, e, g, d*) and (*d, h, i, c*), along with corresponding gates (*d, b*) and (*c, d*). During this conquest, AFF writes to the vertex stream (*a, b, e, g, d, h, i, c, f*), the initial boundary length as *8*, the initial fan-center *a*, and for every fan it gives a fan description:
<3><111> <00>
<3><101> <00>
<3><111> <11>
<3><111> <11>
and an explicit reference to vertex *d* for the first fan as it is being revisited.

Reconstruction: The reconstruction algorithm restores the initial boundary (*a, b, e, g, d, h, i, c*) of the mesh, and the initial fan-center *a* (see Figure 3(b)). The first fan description denotes that the degree of the fan is 3. So we need $3 - 2 = 1$ vertices. From the visited bit pattern we note that this vertex is already visited and hence we get it from the explicitly referenced list, (that is *d*)[1]. Having got the reference to *d*, we construct the fan front as (*b, d, c*). The addition of this fan results in splitting of the boundary giving two new boundaries: (*b, e, g, d*) and (*d, h, i, c*). From

[1]As we shall see later, we do not need explicit re-referencing of earlier visited vertices in all the cases. In most of the cases we can automatically derive the reference to an earlier visited vertex. However, in this example vertex *d* is one of the cases which require explicit referencing. We discuss these cases in greater detail in the next subsection.

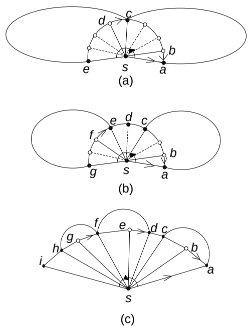


Figure 4: Different interactions between fan-fronts and mesh-boundaries.

the fan-definition, we have edge (*d, b*) as gate for the boundary (*b, e, g, d*) and edge (*c, d*) as gate for boundary (*d, h, i, c*). Recursive application of the procedure is carried out for each boundary, with corresponding gate until all the fan descriptions are added and the original mesh is completely reconstructed.

The AFF algorithm terminates in finite time and completely conquers any given triangle mesh with a single connected boundary. This can be deduced from the following observations. Let M be the mesh, f the currently conquered fan and $T(M)$ the number of triangles.
(1) In order to show that no triangle of the mesh is left unconquered, we consider the two situations of the algorithm discussed earlier. First, when conquering f leaves M_r, such that either M_r remains a single connected component or is empty. In case M_r is not empty, then the fan conquest process continues until M_r becomes empty. Clearly, in this case, no triangle is left unconquered. Second, when conquering fan f splits the remaining mesh into multiple connected components $M_i, i = 1, 2, ...$ For each M_i, AFF selects a gate on the boundary and thus AFF leaves no triangle unconquered.
(2) Since a fan is by definition non-empty, $T(M - f) < T(M)$. Hence AFF terminates in at most $T(M)$ conquest operations.

2.5. Boundary Interactions and Vertex Rereferences

For large meshes, most of the fans conquered by our algorithm have all but the first fan-front edge as interior edges. Hence the second edge on the fan-front usually becomes the gate. Thus, the most common pattern that the *VertexBitPattern* takes is: $< 110^*1 >$ and correspondingly *EdgeBitPattern*: $< 10^* >$.

When encoding such fans, the boundary vertices/edges are implicitly referred to as a connected sequence on the boundary. The other interior vertices on the fan-front are placed on the vertex buffer in an order implied by the orientation of the fans. Thus most of the vertex referencing is implicitly recorded in the fan descriptions.

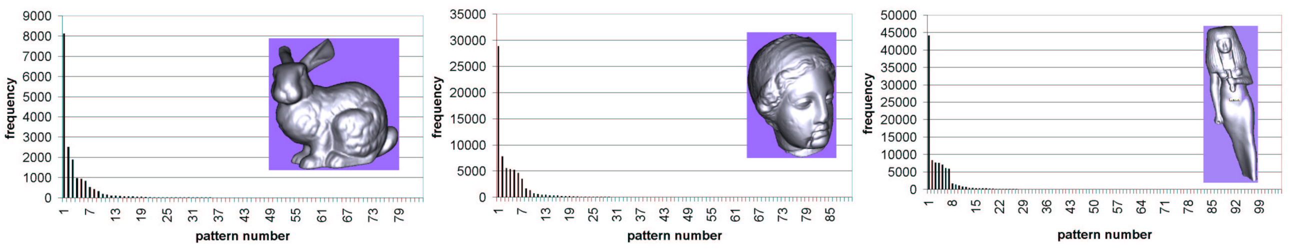


Figure 5: Distribution of fan description patterns.

However, as already mentioned earlier, in certain situations, the fan-front can interact with the mesh boundary in various ways as illustrated in Figure 4. When a vertex is common in both mesh-boundary and fan-front and both of its neighbors on fan-front are not on boundary then we say the mesh-boundary is “touching” the fan-front. For example, in Figure 4(a), mesh boundary touches the fan-front at vertex *c* and in Figure 4(c) at vertex *f*. When the fan front and mesh boundary have common edges, we say they are “overlapping” along the common adjacent edges. For example, the mesh boundary and fan-front are overlapping along (c, d, e) in Figure 4 (b). Similarly in Figure 4 (c) along (c, d) and (h, i). If the fan front is not completely interior to the mesh, the mesh boundary and fan-front can interact only by touching, overlapping or as a combination of these two.

When fan-front overlaps with or touches the mesh boundary, there exist some common edges, or vertices on fan-front and mesh boundary. For example, in Figure 4(a) fan-front $(a, b, ..., c, d, ..., e)$ touches the mesh boundary $(s, a, ..., c, ..., e)$ at vertex *c*. In Figure 4(b) fan-front $(a, b, ..., c, d, ..., e)$ overlaps with the mesh boundary $(s, a, ..., c, d, e, ..., q)$ along (c, d, e).

In the AFF algorithm, at any time the vertices on the mesh boundary are visited vertices. Hence the vertices occurring in configurations where the fan-front and the boundaries interact by either touching or overlapping are previously visited vertices and may need to be re-referenced. However, not all boundary interactions of vertices demand vertex re-referencing. For example, a sequence of vertices representing an overlap between the fan-front and the boundary needs only one re-reference to the start of the sequence and the remaining overlapping sequence can be automatically derived from the order in the list of boundary vertices. As a further illustration of this point, consider the configuration in Figure 4(b). If we know the reference to vertex *c* then we can derive the reference to the vertices *d* and *e*. However whenever a fan-front touches the mesh-boundary at a vertex, a re-reference to that vertex must be recorded. This situation is illustrated by vertex *c* in Figure 4(a). This was also the case in our second example discussed earlier in subsection 2.4 and illustrated in Figure 3.

In our implementation, the explicit references to the previously visited vertices involved in boundary interactions are recorded as integer offsets of those vertices from the current fan-center on the boundary along its orientation. The number of bits needed to represent these offsets are dynamically determined depending on the length of the boundary. These vertex re-references are similar to those in [2], and hence sub-linear in number, by the same arguments.

2.6. Complexity

Each triangle is visited exactly once during the conquest of fans in AFF’s encoding as well as reconstruction algorithms. The queries to determine adjacency information for vertices, edges and faces are satisfied in constant time by using the half-edge data-structure [8]. Thus the time complexity of both the algorithms is linear in the number of triangles in the input mesh.

Model	#vertices	#fans	#fan types	#rerefs	bpv	A&D [2]
Bunny	34839	17884	83	472	1.62	1.98
Dinosaur	56194	29680	133	1488	2.21	2.25
Horse	48485	25080	109	813	1.91	N.A.
Igea	134245	68905	90	1691	1.63	2.71
Isis	187644	95562	105	1688	1.50	N.A.
Knee	37890	18945	7	0	0.05	N.A.
Sphere	1026	515	9	2	1.14	N.A.
Vase	68098	33795	7	0	0.03	N.A.

Table 1: Connectivity compression results (*Since we could not get the exact models as used by others, we are giving the best earlier bit-rates reported in* [2] *for the corresponding models by name; others are not avaliable* (N.A.)).

3. Implementation Results

In the final compressed representation the fans spanning the mesh are encoded using a vertex stream, the number of vertices in the boundary, the starting gate, a sequence of fan descriptions and the sequence of vertex re-references. The structure of a fan description seems to suggest that even for a fixed degree of fan, a large number of fan configurations are possible. However, our experiments show that only a few of the fan description patterns dominate the distribution (see Figure 5), thus enabling very high compression using techniques such as range encoding [10] of these patterns.

Table 1 summarizes the application of AFF connectivity compression to a number of mesh models (see Figure 6) that are freely available on the net. Note that in all these models, as expected, the total number of fans constructed in the

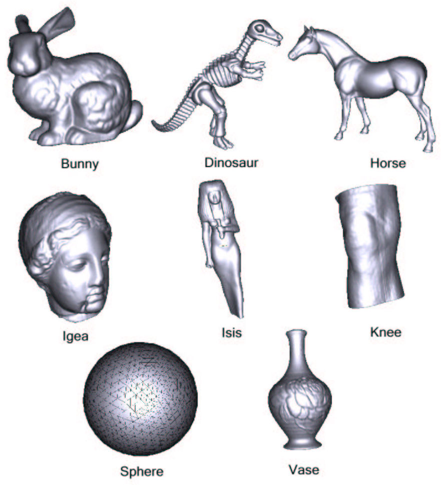


Figure 6: 3D Triangle mesh models used for measurements.

conquest is roughly half the number of vertices in the mesh models. While the total number of fan description patterns (denoted “fan types” in Table 1) is many in number, variable length coding of these fan types yields a very low cost in terms of bits per vertex (bpv), lower than the previously reported encoding costs.

In order to get an intuitive idea of why this compression scheme works so well, consider the performance of AFF for model called Dinosaur in Table 1, which has the largest number of distinct fan types in the table, i.e., 133. At most 8 bits would be required to represent each fan type without any kind of special variable length encoding. The total number of fans is half the number of vertices, hence the cost per vertex would be 4 bits and hence 2 bits per triangle. Clearly, for all the other models listed in Table 1, the cost would be less than this even without any special variable length encoding scheme.

It could be argued that since the fans can easily be decomposed further into triangles, edges or vertices, the AFF algorithm is merely forcing a specific traversal path and hence the mesh could be encoded by detecting repeating subsequences of earlier symbols encountered in the course of this traversal. While this is indeed possible, it is important to note that the special traversal of the triangles induced by AFF lowers the storage cost, on the average, for large meshes which are not highly irregular.

4. Conclusions and Future Directions

In this paper we have presented a new algorithm for very efficient compression of large triangle meshes. While earlier algorithms have used an edge, a face or a vertex as the unit of encoding, our algorithm uses a larger unit, namely a fan of triangles, thus reducing the total number of symbols in the resulting encoding. In large meshes, major parts of the mesh can be spanned by using a very small number of distinct fan types. As a result, our technique provides better compression than earlier known techniques. This is amply illustrated with the help of a simple implementation and tests carried out on a number of large 3D models.

While we have worked out the algorithm for triangle meshes, the algorithm can be extended to cover general polygon meshes, by suitably redefining a fan to include general polygons incident on a single vertex. We also believe that the spatial coherence present in adjacent triangles forming the fan can be exploited to enable very good vertex geometry data compression using simple geometry prediction techniques.

It is worth noting here that decomposing of triangle meshes in terms of fans has the added advantage that it is also well suited for improved performance using graphics acceleration hardware and standard graphics software APIs, which usually have special primitives to handle fans.

Acknowledgment: (a) Dinosaur, Igea, Isis, Knee and Vase – *http://www.cyberware.com/samples/*, (b) Bunny and Horse – *http://www.cc.gatech.edu/projects/large_models/* and (c) the tessellated Sphere was created by the authors.

References

[1] L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal of Discrete Mathematics*, 9:129–150, 1996.

[2] P. Alliez and M. Desbrun. Valence-driven connectivity encoding of 3D meshes. In *Eurographics’01*, 2001.

[3] C. Bajaj, V. Pascucci, and G. Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *In Proceedings of IEEE Visualization 1999 Conference*, pages 307–316, October 1999.

[4] C. Gotsman, S. Gumhold, and L. Kobbelt. Simplification and compression of 3D meshes. In *Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling (PRIMUS), Munich*, August 2001.

[5] A. Guiziec, F. Bossen, G. Taubin, and C. Silva. Efficient Compression of Non-manifold Polygonal Meshes. In *IEEE Visualization 1999*, pages 73–80, 1999.

[6] S. Gumhold and W. Strasser. Real-time Compression of Triangle Mesh Connectivity. In *SIGGRAPH 98*, pages 133–140, 1998.

[7] M. Isenbueg and J. Snoeyink. Face Fixer: Compressing Polygon Meshes with Properties. In *SIGGRAPH 2000*, pages 263–270, 2000.

[8] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, MD, 1988.

[9] J. Rossignac. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January-March 1998.

[10] M. Schindler. A fast renormalization for arithmetic coding. In *Proceedings of IEEE Data Compression Conference, Snowbird, UT*, 1998.

[11] C. Touma and C. Gotsman. Triangle Mesh Compression. In *Proceeding of Graphics Interface 98*, June 1998.

[12] W. T. Tutte. A census of planar triangulations. *Canad. J. Math.*, 14:21–38, 1962.