Geometry Based Connectivity Compression of Triangular Meshes

S.Nachiappan
Illinois Institute of Technology
USA
sundnac@iit.edu

Sanjiv Kapoor
Dept. CSE, IIT-Delhi and Illinois Institute of Technology, USA
kapoor@iit.edu

Prem Kalra
Dept. CSE, IIT-Delhi
pkalra@cse.iitd.ac.in

Abstract

Many interactive 3D graphics applications like manufacturing, design, networked-gaming, need to access 3D data remotely. Transfer of large data sets consumes lot of bandwidth and space. In this paper we describe a lossless scheme for compressing 3D-surface data, represented by triangular meshes. We present a new algorithm which exploits geometry (vertex) information for encoding connectivity. The experimental results obtained are better than Edgebreaker, the existing best method.

1. Introduction

Triangular meshes are widely used in exchanging 3D data. A triangular mesh represents a 3D surface, approximated using triangles. The mesh has *vertex data*, i.e. the co-ordinates of the mesh vertices and *connectivity data*, i.e. how the vertices are connected to form triangles. For most meshes in practice the number of triangles is roughly twice the number of vertices. If vertex data is in floating-point co-ordinates and if triangles represented by their three vertex-ids (integers), the volume of connectivity data exceeds the volume of vertex-data. Vertex-data is also known as *Geometry-data*. Geometry compression methods in practice, compress vertex data down to one tenth of connectivity data. Pointers and references to geometry compression can be found in Rossignac [7]. Here, we focus on connectivity compression.

In this paper we present a novel approach for lossless encoding scheme for compressing connectivity data of triangular meshes: geometry based connectivity compression (Gbcc). First we overview basic terminology and prior work on connectivity compression, following which we describe the basic idea of our approach. Subsequently we present our algorithm. Finally we discuss the results.

2. Basic Terminology

Let V be the vertex set and T be the triangle set of the mesh. If an edge e is one of the edges of a triangle t, then e is said to be *incident* on t and vice versa. If a vertex v is incident on an edge e, then v is said be *incident* on e and vice versa.

Two triangles are *adjacent* if they share an edge. A dual graph G_D of the mesh is defined as follows. For each triangle in the mesh, there is a vertex in G_D. Two vertices in G_D have an edge between them, if and only if the corresponding triangles in the mesh are adjacent, i.e., they share an edge.

A depth-first traversal of G_D generates a dfs-tree, called as the *triangle spanning tree* or simply *spanning tree*, since it corresponds to a spanning of the triangles in the input mesh. This tree does not specify connectivity tree fully, since such a tree might correspond to more than one mesh. The dfs-tree induces an *ancestor-descendent* relation between triangles. Note the distinction between *neighbours of a triangle* and *children of triangle*. Two triangles are neighbours iff they share an edge, irrespective of how the mesh is traversed, whereas parent-child relation is imposed by the dfs-traversal.

The ratio of the size of connectivity encoding to the number of triangles, is known as the *connectivity cost*. It is usually measured in *bits-per-triangle*, abbreviated as *bpt*.

3. Prior Art

A raw representation of connectivity, costs $3\ log_2|V|$ bits per triangle, if a triangle is specified by its three vertex-ids. For practical meshes, the cost can be as large as 40-50 bits per triangle

Turan [10] observed that since planar graphs can be decomposed into two spanning trees, they could be encoded in constant number of bits per vertex. He gives an encoding that uses 12 bits per vertex. Taubin and Rossignac [9] describe the Topological surgery method, which encodes a vertex-spanning-tree and a triangle-spanning-tree and achieve a connectivity cost of around 2 bpt. Isenburg and Snoeyink's [4] mesh-collapse compression encodes mesh connectivity by a series of invertible edge collapse operations. Entropy coding of this sequence results in bit rates in the range of 0.6 to 1.8 bpt. Isenburg [3] proposes an edge-based algorithm, achieving 0.8 to 1.4 bpt. Gumhold and Strasser [2] describe a fast compression and decompression scheme, with 1.7 to 2.5 bpt.

Rossignac [7] proposes EdgeBreaker scheme, which requires atmost 2 bpt. Rossignac and King [5] improve this bound to atmost 1.84 bpt, the best known bound till now and which lies within 13% of the theoretical lower bound due to Tutte [11].

Bajaj [1] decomposes into layers of triangles and compresses data locally within each layer using geometric primitives, resulting in a connectivity cost of 1 to 3.5 bpt.

A preliminary investigation on using geometry for 2D meshes was done in [6]. The algorithm we present here is inspired by that work.

4. Basic Idea of Our Approach

Most compression algorithms treat connectivity and vertex-data separately, though there is often a strong corrleation between them, implying that more compression is possible if the correlation is exploited. Consider figure 1. Let us reason inductively. Suppose that triangle T has been encoded. Triangle A has just been encoded and we have to encode triangle B. By induction, suppose that the decoder knows the vertex-data and connectivity of triangles T and A. In order that the decoder be able to decode B correctly, we need tell the co-ordinates of vertex S and also if the triangle B will become the right child or the left child of triangle A. In this case, triangle B is the left child of A. Consider the point C on the median of triangle B. The medial bisection plane or in short *mbp* of triangle A is that plane which is perpendicular to the plane of A and containing its median (which is drawn from the midpoint of the entering edge to the opposite vertex). In the figure shown, L is the midpoint, QR is the entering edge and P is the opposite vertex. C lies on the left of the mbp of A and triangle B is the left-child of A.

Note that we can always find a point C on the median of a child triangle, with the property that if the child-triangle is the left-child or right child of its parent, then the point C lies to the left or right of the mbp of its parent, respectively. Let us call this the *placement property* and point C as the *control-point*. Vertex S is also known as the *third vertex* of triangle B, as two of its vertices P and Q are already known. Every triangle will have its own control-point and third-vertex.

Define ϵ to be the ratio $KC : CS$ in figure 1. Every triangle might have its own ϵ depending on its relative size and orientation. Since the point C lies on one of the sides of the mbp, the length of CK is limited and this places an upper limit on the value of ϵ. The value of ϵ for each triangle is dependent on how the mesh is traversed, since ϵ is characteristic of the relative orientation and size of the parent and child triangles.

Any point C' on the median line KS, such that $C'K < CK$ also satisfies the placement property. In other words

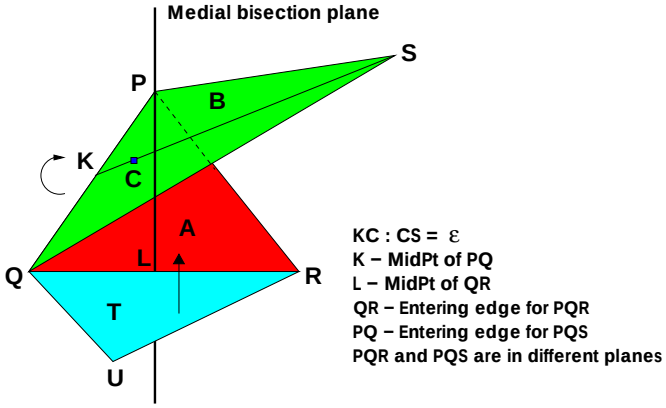


Figure 1: Triangle Encoding

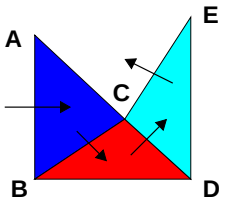


Figure 2: Closing case

any ϵ', smaller than ϵ is also acceptable, in the sense that the point C' it defines satisfies the placement property. For every triangle, we compute that largest possible ϵ. Then we pick the smallest among them to be a global ϵ value, which will work for all triangles. Thus the space required to store ϵ is constant. We make a trial run over the mesh and compute the global ϵ value and pass it to the decoder as header data. Then we traverse the mesh once more, to compute control-points for all triangles. We tell the decoder, the control-point for the triangles in the mesh.

Referring to figure 1, knowing ϵ and the control-point, the decoder can reconstruct the co-ordinates of S, since by induction it knows the co-ordinates of the vertices triangles A and T. Also by the placement property, it can also determine if triangle B is the left or right child of triangle A and hence the connectivity for triangle B. Hence by specifying the point C, we can reconstruct both vertex-co-ordinates and connectivity.

5. Closing Triangles

While making the dfs-traversal of the mesh, whenever we mark all the vertices of a triangle as soon as it gets visited. For each triangle we need to encode its third-vertex and its placement w.r.t to its parent. While encoding a triangle if its third-vertex is not yet seen, we compute a control-point for the triangle, so that its placement and its third-vertex can be reconstructed.

Consider figure 2. We enter triangles ABC, BCD, CDE in the order shown by the arrows. When entering triangle ACE, we find that its third-vertex A is already marked, since triangle ABC was already visited. We can compute a control-point for triangle ACE, but that requires specifying floating-point numbers. It turns out that we can encode the triangle ACE in 2 bits. Observe that the adjacency list of vertex C has both the vertices A and C in it. By maintaining adjacency lists for all vertices in such way that every adjacency list defines a single fan, we can guarantee that vertex A and C will be at the extremities of the adjacency list of C, as they are the extreme vertices in the fan of vertex C.

Vertex C is also known as the *common vertex* of triangle ACE, since it is common to $ACE's$ previously visited neighbours ABC and CDE. To decode triangle ACE, the decoder scans the adjacency list of the common-vertex C, starting from the where vertex E occurs. At the end of the scan, vertex A must occur. All the decoder needs to know is what is the common vertex. Note that the common vertex is one of the three vertices of the CDE, the parent of triangle ACE. Thus we need 2 bits to specify the common vertex.

Triangle ACE is also known as a *closing triangle*, since visiting it, is equivalent tracing a cycle in the dual graph. For a closing triangle, its third-vertex is seen as marked when the traversal enters it. If the third-vertex is not marked, then it is called a *proper triangle*. Control points are computed only for proper triangles.

6. Encoder and Decoder

The Encoder, makes trial run to compute a global ϵ. In the next traversal, it classifies triangles as *closing* and *proper* triangles. It computes control-points for proper triangles and common-vertex for closing triangles. For each triangle it assigns a label (explained later), based on how many children a triangle has and what is the common-vertex if it is a closing triangle. The labels are output by some well-defined traversal of the triangle-spanning tree.

The decoder receives the labels and control points, reconstructs the same spanning tree which the encoder had computed.

7. Error Control

In the decoder, the reconstruction of a vertex from control-point and ϵ involves a division by ϵ. With highly skewed triangle orientations and sizes, ϵ can be as low as 10^{-5}. Hence a small numerical error is magnified. Besides, the decoder reconstructs new vertices based on previously reconstructed vertices. Hence error of reconstruction in previous computations propagates to subsequent computations. A series of ten computations can result in an unacceptable accumulation of error.

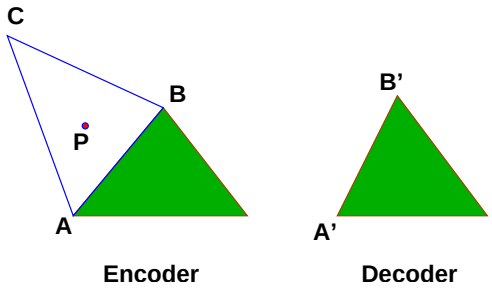


Figure 3: Error Control

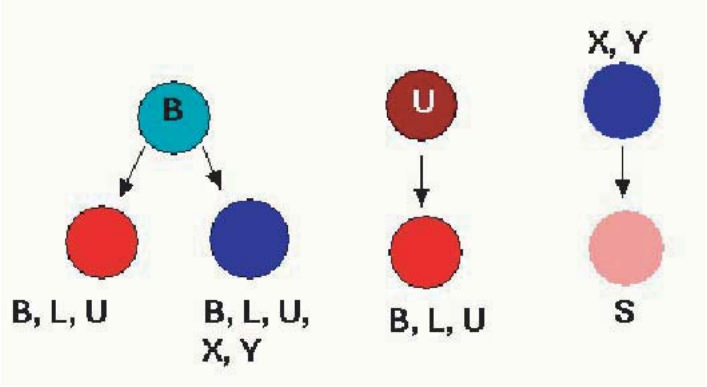


Figure 4: Triangle Labelling

Consider figure 3. The encoder computes the control-point P from using vertices A,B and C. But the decoder reconstructs C from A' and B', which are the decoder's version of vertices A and B. This is precisely the reason for *error propagation*. Hence let the encoder compute P using A' and B' instead of A and B. Using simple error analysis, we can show that error will not *propagate* if encoder does so. The error will be there and can be bounded in terms of the arithmetic-precision used, but it will not propagate. To know A' and B', the encoder runs a mini-decoder inside it.

8. Triangle Labelling

A proper triangle is a *branch* or a B triangle if it has two children in the spanning tree; a *Unichild* or a U triangle if it has one child; a *leaf* or a L triangle if it has no children.

A closing triangle is labelled an X triangle, if its common vertex is the third-vertex of its parent; a Y triangle if its common vertex is either the first or the second vertex of its parent. Though a Y label encodes two cases, it can be uniquely decoded based on context.

Thus, the label, a triangle can receive is one of B, L, U, X, Y. If it is a closing-triangle with no child then it receives an additional label - S, see figure 4.

9. DFS Constraints

We observe some constraints/properties for DFS traversal, which restrict the labels a triangle receives, depending on what the label of the parent was.

For example, suppose a U triangle t has an unvisited neighbour n, such that if n was entered from t, n would become a closing triangle. We claim that t can ignore it, considering that there exists another triangle t' which will take up n as its child. The argument of the claim is as follows. Since n would become a closing triangle if entered from t, it must have had another of its neighbours visited. Let this be triangle t'. When t' was entered t was unvisited. Since t' did not visit n it must have marked it for later exploration, thus t could ignore n. This also implies that i) a U triangle cannot have a closing triangle as its child and ii) a closing triangle is either the second child of a branch triangle, or child of another closing triangle.

These constraints reduce the possible set of labels the children of a triangle t might receive, given that t receives the label l_1. Thus the uncertainty about the label of children is reduced. This would imply reduction in the uncertainty of label-assignment - hence the entropy of the connectivity string goes down resulting into better compressibility.

10. Bad Closings

Earlier we saw that a closing triangle occurs, when its third vertex is already marked by the time the dfs-traversal visits it. In order that the decoder can reconstruct we need to tell the common-vertex of the closing. The precondition for encoding a closing triangle is that, two of its neighbours must have been previously visited, so that we can define a common-vertex for it. It is possible that when the traversal enters a closing triangle, only one of its neighbours is visited. In that case we ignore that triangle, claiming that it will be visited again along some other path. The ignored triangles are called *bad closings*.

For a mesh with atmost one boundary, bad closings cannot exist at the end of the traversal. A bad closing triangle has exactly one of its neighbours visited, by definition. Any other neighbour is unvisited. Hence every other neighbour of bad closing is another bad closing or an unvisited triangle. Thus bad closings and unvisited triangles must form chains, if they exist. These chains are closed or they terminate at a boundary. In other words, the bad closings fragment a mesh into several pieces, as shown in figure 5

A bad closing has all its vertices marked. Note that atmost two vertices a bad closing can border a fragment. The third vertex is either inside the chain or borders another fragment. To enter one fragment from another fragment the traversal must cross the bad closings. But since bad closings are not visited, we cannot move from any fragment to any other fragment. This implies that not all three vertices

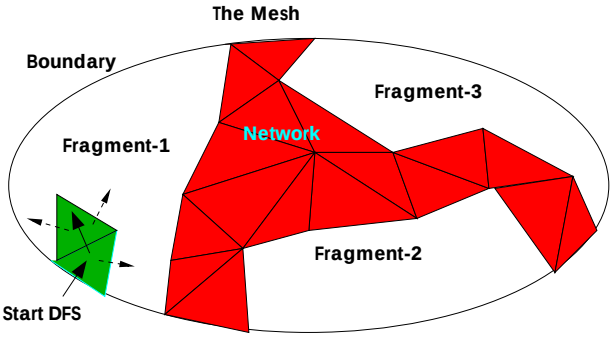


Figure 5: Fragmentation

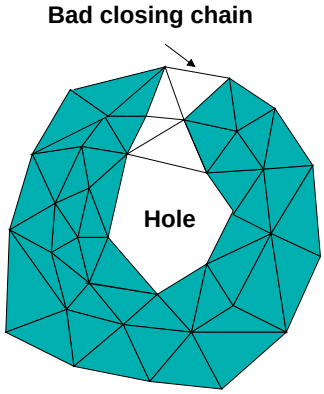


Figure 6: Mesh with a Hole

of a bad closing could have been marked, yielding a contradiction. Hence bad closings cannot occur in a mesh with atmost one boundary.

11. Holes

The earlier argument that bad closings do not exist at the end of the traversal of a mesh with atmost one boundary, was based on the fact that, the bad closing and unvisited triangles form a network which fragments the mesh into different components. If there is more than one connected component, then we could not have reached one component from another at all. Since a bad-closing can have atmost two of its vertices bordering a component, it is impossible that its remaining vertex was visited (the remaining vertex belongs to a different component or is embedded in the network).

If there are holes, then bad-closings need not necessarily fragment the mesh in to different components, as shown in figure 6. Note that there is a hole and also the chain of bad-closing triangles (shown unshaded). The mesh is still in one piece.

To deal with holes, whenever we encounter a bad closing triangle, we record it. Also when some triangle ignores

some other triangle in the traversal, we have to see of the ignored triangle can become a potential bad closing and if so we have to record that ignored triangle also. Once the traversal is complete, we have a record of bad closing triangles. Some of these might have been marked as bad-closings, but became good-closings because they were reached by other path. Hence there is no need to consider such triangles. We need to consider only those triangles in the record, which are still unvisited.

We select some arbitrary triangle from the record. It will belong to some chain. We explicitly record its vertex ids and pass it to the decoder. Thus some triangle is forcibly visited.

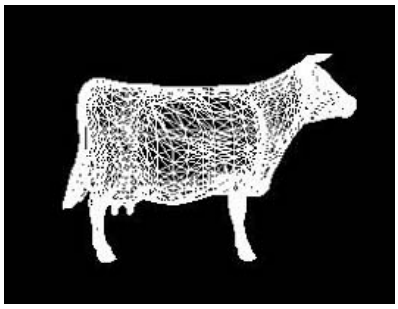
It can be proved that for every chain of bad closing triangles, if one triangle in the chain gets visited, then every other triangle in the chain will subsequently become a good closing triangle. Hence all other triangles in the chain can be visited. The additional cost per chain is found to be $c * log_2|V|$, where c is rather small.

12. Results

In our case, the source has memory i.e., the next-label emitted depends on the previous label. We have fitted an *order-1* model on our source. We have to transmit the probability distribution for this model as header-data to the decoder. In general it can be shown that in an *order-k+1* model, the uncertainty as to what is the next label, is atmost that of an *order-k* model, as indicated by Shannon [8]. But the penalty is that the space required for transmitting *order-k* model statistics, for a source-alphabet size n, is $O(n^{k+1})$. It might turn out that reduction in uncertainty of the source obtained by an *order-k* model is offset by the overhead incurred in transmitting the statistics. Besides the analysis of traversal strategy reveals a strong presence of *order-1* dependency and no more. Hence we use *ordei-1* statistics only.

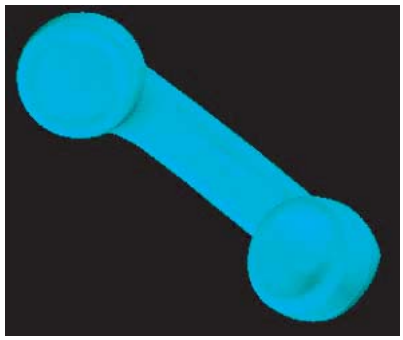
As the mesh is also part of the source, if the mesh has a regular structure, then the next-symbol might depend on many previous symbols. But the *order-1* model we use, may not capture this dependency. Higher-order models can do so, but these are costly. One good solution is to learn the model dynamically - this is what the **LZ-family** of algorithms do. These algorithms will compress their input well, only if there are repitions of long substrings. They will perform poorly if the higher-order dependencies are weak. In that case our *order-1* model will suffice.

So, we run *gzip*, an unix-implementation of the LZ-family and also we run our *order-1* encoder and pick the best. We can tell the decoder what we picked, as part of the header-data. Some results are shown in figure 7 and in table 1. The second and third columns in the table indicate the number of vertices and triangles in the mesh, respectively.



Vertices 2912
Triangles 5801

Edgebreaker (bpt): 1.88
Gbcc (bpt): 1.29



Vertices 83047
Triangles 166089

Edgebreaker (bpt): 1.90
Gbcc (bpt): 0.661

Figure 7: Results

Input-Mesh	|V|	|T|	Gbcc(E1)	Edgebreaker
Golfstick	**263**	**521**	**1.47**	**1.7**
Mannequin	**690**	**1375**	**1.47**	**1.7**
HumanFace	**989**	**1,97**	**1.48**	**1.7**
Terrain	**40,001**	**79,997**	**0.069**	**1.5**

Table 1: Results

Column 5 gives the connectivity cost in **bits-per-triangle** for edgebreaker. Column 4 gives the order-1 entropy **E1** of the connectivity string output by our algorithm (Gbcc).

The version of edgebreaker code available, could deal only with meshes without any hole. So, meshes with holes had to be converted into hole-less meshes by putting in one additional vertex per hole and triangulating the hole. The experiments then were performed on meshes for which the holes had been removed. We can use arithmetic coding to reach the entropy-limit and hence get a bits-per-triangle rate which is almost close to the entropy.

The **Terrain** mesh mentioned in table 1 has a recursive substructure. Hence the entropy is remarkably low. Other meshes do not have a significant regularity and hence the entropy is higher.

13. Conclusion

A new algorithm which encodes connectivity based upon geometry is presented. The approach is novel. The empirical results are better than the existing best algorithm – Edgebreaker. More observations have been made about the algorithm which the authors believe can provide a theoretical bound atmost 1.84 bits per triangle. Further possibilites are being researched. More efficient techniques for dealing with holes are being analyzed.

The theoretical worst case of connectivity cost for holeless meshes is 2.66 bits-per-triangle as of now. But observed entropies are much lower compared to 2.66. There are lot of constraints that have not been taken into account in computing the 2.66 figure. For instance, the number of branch triangle is exactly $N_S + N_L - 1$, where N_S and N_L are the number of S labels and L triangles respectively. Also the number of proper triangles is $|V| - 2$ and the number of closing triangles is $|T| - |V| + t$. Besides Euler's relation implies that $|T| < 2|V|$. All these constraints when applied might bring down the worst case connectivity cost.

14. Acknowledgments

The authors would like to extend thanks to Dinesh Shikhare of NCST, India and Prof. Jarek Rossignac and his group of Georgia Institute of Technology for providing the source code of Edgebreaker algorithm and related data converters.

References

[1] C. L. Bajaj, V. Pasccuci and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *Proceedings of Data Compression Conference*, pages 247–256, 1999.

[2] S. GumHold and W. Strasser. Real time compression of triangle mesh connectivity. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 133–140, 1998.

[3] M. Isenburg. Triangle fixer : Edge-based connectivity compression. Technical Report TR-99-38, University of North Carolina at Chapel Hill, 1999.

[4] M. Isenburg and J.Snoeyink. Mesh collapse compression. In *Proceedings of SIBGRAPI'99 - 12th Brazilian Symposium on Computer Graphics and Image Processing*, pages 27–28, 1999.

[5] D. King and J. Rossignac. Guaranteed 3.67 bit encoding of planar triangular graphs. In *11th Canadian conference on computational geometry*, pages 146–149, August 1999.

[6] Sanjiv Karoor, Prem Kalra, Gaurav Rastogi and Vivek Jawa. Connectivity compression of triangular meshes. Technical report, Department of Computer Science, Indian Institute of Technology, New Delhi, India, June 2000.

[7] J. Rossignac. Connectivity compression of triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), January-March 1999.

[8] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

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

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

[11] W. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.