A Survey of Mesh Compression Techniques

Eric Lorimer*
University of Illinois, Urbana-Champaign

ABSTRACT

As the field of computer graphics advances, computers and graphics hardware become more powerful. However, the size and complexity of the meshes we wish to represent are also increasing to the point where we must find some way to compress this information in order to make storage and transmission feasible (or more feasible). This paper will survey the research in mesh compression, provide some background, and look at future directions in the field.

1. INTRODUCTION

In many areas of computer graphics, there arises a need to work with very large meshes. The Digital Michelangelo project, for example, has scanned the David statue at a resolution of .29mm which requires 32GB of data to store. Aside from the problems of manipulating such a large data set on machines with a limited amount of main memory (usually much lower than 32GB), techniques for compressing meshes such as these are becoming increasingly important in order to make storage and transmission more feasible. In addition, there remains the elusive possibility of streaming 3D content over the Internet in real-time. Many games (e.g. role-playing games) experience bottlenecks in bandwidth due in large part to transmission of game data including mesh data.

Thus, we see the need for compression at two ends of the spectrum. One side seeks to compress relatively small meshes further to reduce transmission time over slow Internet links. The other side seeks to compress very large meshes in order to make storage and transmission more feasible.

A mesh can be simply defined as the set of vertices, edges, faces together with their incidence relationships. We can consider the incidence relationships (i.e. what faces are incident on a vertex, what edges are incident on a face, etc...) the mesh connectivity and the vertex positions the mesh geometry. Most mesh compression techniques have treated the mesh geometry and the mesh connectivity separately.

*lorimer2@uiuc.edu

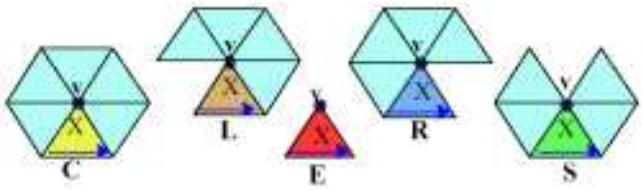

Figure 1: EdgeBreaker CLERS symbols

2. COMPRESSING MESH CONNECTIVITY

One of the earliest attempts to encode mesh connectivity was made by Deering [1]. In this paper he builds on the idea of a generalized triangle strip in which the direction to extend the chain when appending a new vertex is encoded as well. Deering introduces a fixed-length (16) queue called the mesh buffer [1]. Vertices must be explicitly pushed and later referenced from this queue. This avoids the problem with generalized triangle strips requiring frequent restarts or duplicating already decoded vertices. Deering defines four operations for his mesh buffer - replace oldest, replace middle, restart clockwise and restart counterclockwise. In addition, Deering preserves the advantage of triangle strips in that they can be decoded by a single linear scan of the data. Deering's method, after cutting the mesh into strips, has no way to stitch it back together.

EdgeBreaker [10] is a more sophisticated connectivity compression algorithm. EdgeBreaker is a face-based compression scheme which traverses the faces (triangles) of the mesh generating a spanning tree. The algorithm encodes one of five symbols at each face to keep the history so that the process can be reversed during decoding. The five symbols form the *clers* string and are defined as follows. A **C** is encoded when the vertex has not been visited. A **L** is encoded when the left triangle has been visited, a **R** is encoded when the right triangle has been visited, an **E** is encoded when both left and right triangles have been visited and **S** is encoded when neither left or right triangles have been visited. (See figure 1) In the **S** case, EdgeBreaker recurses on the right subtree and then the left. EdgeBreaker can compress the connectivity of the mesh to near optimal rates, normally around 2 bits/vertex.

A more elegant formulation of the algorithm is described by Rossignac [7]. This implementation makes use of a corner

table data structure which makes use of two tables to store the vertices and their opposite corners. This allows creation of functions such as "next corner around triangle" and "previous corner around triangle" easily. The entire algorithm can be succinctly expressed in only a page of pseudocode.

Although EdgeBreaker is a simple, efficient method for compressing the connectivity of a mesh, it has some limitations. First, EdgeBreaker as originally expressed is limited to triangle meshes. In practice, as many meshes are triangulated, this may not seem a large problem, but nevertheless it limits the areas in which it can be applied. Second, EdgeBreaker requires random access to the vertices. This is inconvenient for gigantic meshes for out-of-core processing.

Isenburg *et al.* [6] deal with the limitation of triangulated meshes and extend EdgeBreaker to deal with arbitrary polygon meshes. Gumhold and Strasser [3] use a similar idea as EdgeBreaker in their Cut-Border-Machine, but have the advantage of single-pass encoding and decoding making it more useful for out-of-core processing applications.

3. COMPRESSING MESH GEOMETRY

3.1 Lossless Methods

Mesh geometry refers to the set of vertex positions of the graph. The earliest and still most popular method involves a two-stage process of quantization and predictive encoding. Quantization reduces the range of the data. In the context of geometry compression, quantization takes the three components, (x, y, z), of each vertex and stores them in a fixed number of bits (typically 10-14 is sufficient). The quantized mesh at 10-14 bits is visually indistinguishable from the original (usually 32 bits). Therefore, this quantization can be considered "lossless."

Further quantization, however, introduces very noticeable high-frequency noise [12]. Sorkine *et al* [12] show how this high-frequency distortion can be converted to low-frequency, large-scale geometry distortion which is less noticeable to the human visual system (though still quite significant) by applying the Laplacian to the vertices before quantizing to generate "δ coordinates" which are then quantized and the process is reversed on the decoding side. This produces much more aesthetically pleasing results, but it is debatable whether distorting the low-frequency components is more or less acceptable than high-frequency artifacts.

The second part of these lossless compression algorithms involves some form of spatial prediction. Based on the fact that the decoding process usually loosely orders the vertices by position and that within a local region vertex positions are highly correlated, spatial prediction attempts to "guess" the location of the next vertex given the already decoded vertices. Linear predictive coding is a simple scheme which uses a linear combination of a small number of previous vertices to predict the next vertex. The most simple linear rule is to predict the vertex to be the same as immediately preceding vertex. This leads to a simple delta-encoding.

The most widely used prediction rule, the "parallelogram predictor," is based on the observation that adjacent triangles tend to form parallelograms, therefore it predicts the next vertex to form a parallelogram with the previous

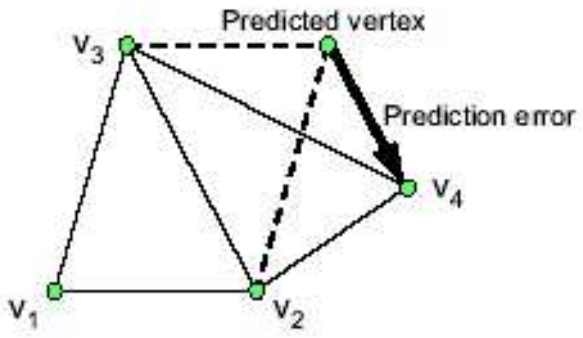

Figure 2: Touma-Gotsman parallelogram predictor

three [13]. This works well, but fails to account for the curvature in the mesh and cannot predict the crease angle between adjacent triangles. [Fig. 1]

The predicted vertex is then compared to the actual vertex position and this difference encoded into the output stream. When the predicted vertex position and actual vertex position are close, the difference can be stored in fewer bits than would be required to store the actual (quantized) position. There are various ways to encode this difference.

3.2 Lossy Methods

Because the spatial prediction rule is fixed by the algorithm, the only way to gain further compression improvements is in the quantization stage in the lossless algorithm. However, as noted, agressive quantization leads to distinct artifacts and can no longer be considered lossy.

Karni and Gotsman [8] propose to use spectral analysis to take advantage of the information present in the connectivity to aid in the geometry compression. Building on the signal processing framework introduced by Taubin [11] for surface fairing, they decompose the Laplacian matrix into its orthogonal eigenvectors and project the geometry signal (each x, y, z component separately) onto this basis.

The Laplacian matrix is a sparse n x n matrix (n is the number of vertices) defined as follows:

$$L_{ij} = \begin{cases} -1 & \text{if } i = j, \\ 1/d_i & \text{if } i \text{ and } j \text{ are neighbors,} \\ 0 & \text{otherwise.} \end{cases}$$

where d_i is the degree (or valence) of vertex i.

They succeed in demonstrating that by discarding the "high-frequency" (eigenvectors with large corresponding eigenvalues) components they can compress the mesh with less distortion than the lossless algorithm using the parallelogram predictor.

However, computing the eigenvectors of the sparse Laplacian is prohibitively expensive for anything larger than trivial meshes (approximately 600 vertices). In addition, numerical stability becomes and issue when computing eigenvectors of very large matrices. Their algorithm relies on mesh parti-

tioning to make the problem feasible [8], but this is far from ideal.

More recently, Karni and Gotsman [9] suggest using fixed spectral basis vectors computed from a 6-regular triangle mesh. This is convenient as it allows the encoder and decoder to use the FFT to compute the basis vectors. The problem then reduces to mapping vertices in the "candidate" mesh (the mesh they wish to compress) into vertices in the "host" mesh. Where the number of vertices, and in particular, the number of boundary vertices is not the same, they augment the candidate mesh by placing new vertices in such a way as to normalize the degree of the vertices in the mesh. This augmentation is just a special case of remeshing. They show very satisfactory results - a small loss in quality for the potential to encode and decode the mesh very efficiently.

In principle, spectral analysis compression methods can be used for progressive transmission, but in practice little work has been done in this area likely due to the fact that the encoding and decoding time tend to dominate the algorithm and transmission time is not usually a bottleneck.

Spectral methods work best on smooth meshes where most of the energy is concentrated in the low frequencies. Meshes generated from CAD models, for instance, with sharp edges which must be preserved are not well suited to spectral compression methods.

4. MULTIRESOLUTION AND PROGRESSIVE COMPRESSION METHODS

4.1 Progressive Meshes

Another significantly different approach to compressing meshes than the standard face-based connectivity coding and geometry compression recognizes that simplification can be considered a form of compression, albeit a very lossy one.

There are three components of progressive simplification compression methods. First, the choice of the simplification operator. Second, the choice of a metric to determine which mesh element to remove. Third, an efficient coding of the information to reconstruct the mesh.

By defining an invertible simplification operator and recording the steps during simplification, simplification can be used for lossless compression as well as lossy compression. Edge collapse simplification schemes such as Hoppe's Progressive Meshes [4] are well suited for this. The inverse of an edge collapse is a vertex split which inserts a vertex into the mesh next to an existing vertex and morphs it into place creating a new edge. Hoppe chooses the edge to collapse based on a complex energy minimization problem which yields high quality results but simpler edge collapse simplification methods exist (e.g. quadric error metric [2]) which might be more suitable for real time progressive compression.

Hoppe [5] shows that Progressive Meshes can be used to compress meshes to approximately 6 bits/vertex.

4.2 Wavelets

Wavelet compression techniques have proven very successful in the area of 2D image compression. The recent JPEG2000 standard and the MPEG4 still image coder both use wavelets to compress images. So it is not surprising that recent research in mesh compression has also tried to exploit wavelets for compression. The fundamental difference between images and meshes, however, is that images are sampled on a regular 2D grid whereas meshes, in general, have very irregular connectivity and sampling. Thus, much of the focus in wavelet mesh compression involves remeshing techniques typically based on subdivision schemes to produce a semi-regular meshing to exploit the wavelet transform.

During the remeshing, a sequence of approximations at different resolutions is generated. The wavelet transform converts this sequence into a base mesh and a sequence of coefficients which can be efficiently encoded.

An outstanding problem in all lossy geometry compression schemes is the choice of an appropriate visual metric to guide both the simplification as well as for measuring rate-distortion in order to evaluate and compare methods. Karni and Gotsman [8] propose a visual metric which is the average of the geometric distance between the models and the distance between the Laplacian (normals). Sorkine *et al.*, trying to show that the normal distortion is more important than geometric distortion, naturally argue that the ratio should significantly favor preserving normal distortion. In any case, finding a suitable visual metric for lossy compression methods is still very much an open problem.

5. CONCLUSION

Mesh compression has come a long way driven by the desire to represent more and more detailed objects (like the David statue). This trend is likely to continue and as long as the complexity of the models grows faster than storage and transmission improvements, mesh compression will remain a topic of research. It is also clear that lossless mesh connectivity compression has nearly reached the optimal limit and not much more work can be done to improve connectivity compression. However, compressing mesh geometry is still a difficult problem with no clear-cut solution. Novel methods like spectral analysis and wavelet transforms could provide new insights and directions for future research. When the restriction of losslessness is lifted, even more techniques become possible. An important issue to be resolved before lossy compression methods can be fully evaluated is the definition of a suitable visual error metric.

6. REFERENCES

[1] M. Deering. Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 13–20. ACM Press, 1995.

[2] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

[3] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. *Computer Graphics*, 32(Annual Conference Series):133–140, 1998.

[4] H. Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[5] H. Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics*, 22(1):27–36, 1998.

[6] M. Isenburg and J. Snoeylink. Face fixer: Compressing polygon meshes with properties. In K. Akeley, editor, *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 263–270. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[7] A. S. Jarek Rossignac and A. Szymczak. 3d compression made simple: Edgebreaker on a corner table. In *Proceedings of Shape Modeling International Conference, Genoa, Italy*, 2001.

[8] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In K. Akeley, editor, *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 279–286. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[9] Z. Karni and C. Gotsman. 3d mesh compression using fixed spectral bases. June 2001.

[10] J. Rossignac. EdgeBreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, /1999.

[11] G. Taubin. A signal processing approach to fair surface design. *Computer Graphics*, 29(Annual Conference Series):351–358, 1995.

[12] C. Touma and C. Gotsman. High-pass quantization for mesh encoding. *Graphics Interface '98 Conference Proceedings*, pages 26–34, 1998.

[13] C. Touma and C. Gotsman. Triangle mesh compression. *Graphics Interface '98 Conference Proceedings*, pages 26–34, 1998.