Compressing Large Polygonal Models

Jeffrey Ho Kuang-Chih Lee David Kriegman

Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, IL 61801

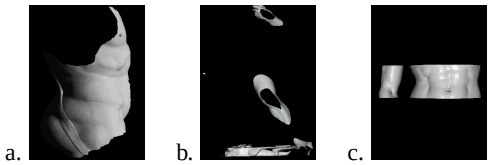

Figure 1: a. This partition is induced from a simplified mesh; b, c. Partitions using z and y axes, respectively.

With the recent and rapid advances in digital acquisition technology, meshes with millions if not billions of vertices are becoming increasingly common. Existing mesh compression/decompression algorithms are only effective if a representation of the mesh's entire topological and geometric structures (and other attributes) is small enough to fit in memory. Yet for a mesh with a few million vertices, one faces the possibility that there is insufficient memory on a regular desktop computer for the entire model. Our approach toward compressing these large models is to automatically partition the mesh into submeshes of smaller size, depending on the available local memory, and then to compress them separately.

The main purpose of the mesh partitioning is to divide the input mesh into submeshes of roughly equal sizes, i.e., the partition should be balanced. However, from the compression standpoint, it is also desirable that 1) each region of the partition is "localized" somewhere in the model, and 2) the boundary of each region is as simple as possible. Straightforward mesh partitions using x, y, z coordinate axes or the level sets of some other linear functions generally do not satisfy these requirements. See Figure 1. Instead, we propose a simple partitioning scheme based on a simplified mesh. Using vertex clustering [1], we obtained a simplified mesh that is typically 100 to 200 times smaller than the original. Each vertex of the simplified mesh has a weight that is the number of corresponding vertices in the original mesh that cluster to the given vertex, and each edge carries a weight giving the number of collapsed triangles that form the edge. The simplified mesh can be partitioned as a weighted graph to obtain a balanced partition that minimizes the edge cuts. The partition of the simplified mesh then induces a balanced partition of the original mesh that usually satisfies the two requirements above. Therefore, we have an in-core representation of a simplified mesh which is used as a kind of blueprint for partitioning and hence compressing the original mesh.

Armed with this partitioning scheme, we have experimented with two similar methods for compressing large polygonal meshes, one compresses the connectivity losslessly while the other ignores the boundary identifications. For the first method, we simply partition the mesh and compress each submesh separately without regard to how different cut boundaries should be identified. By cut boundary, we mean the non-empty intersection between two neighboring regions of the partition. The advantage of this approach is that it is easy to implement and can be built immediately on top of existing mesh compression software, e.g. [2, 3]. The vertices belonging to the cut boundaries are encoded twice; hence, depending on the way the mesh is partitioned and the number of regions in the partition, the number of duplicated vertices can range from as few as 1% of all the vertices to as high as 20%; in the worst case, it is possible that more than half of the vertices will be encoded twice. This clearly illustrates the peril of using an arbitrary partition. Using a simplified mesh as a guide, our partitioning scheme will almost never produce the worst-case partitions. It has to be noted, however, that minimizing the number of vertices on the cut boundary does not necessarily guarantee smaller size for the compressed code. Nevertheless, we have observed through our experiments that the compressed output produced by our partition scheme ranges from 2% to 8% with an average of 4% less than the compressed output from a "bad" partition.

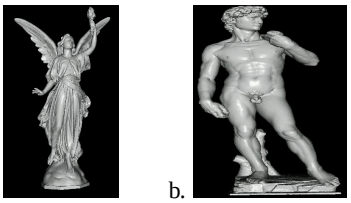

Figure 2: Two of our test models : a. Lucy from Stanford 3D Scanning Repository. 28055742 triangles and 14027872 vertices. b. David from the Digital Michelangelo Project. 8254152 triangles and 4129614 vertices.

The second method encodes the connectivity losslessly. For this, we have developed an efficient method to encode (and decode) the cut boundaries based on identifying their connected components. The main idea is to use run-length encodings for the regular vertices of the cut boundaries while for the singular vertices, we simply encode them directly. For meshes with mostly smooth cut boundaries, this part of the compressed code is generally negligible, even for very fragmented and complex cut boundaries.

Currently, we have only used the connectivity of the simplified mesh. Future research will be focused on how to utilize the geometry of the simplified mesh to define non-linear prediction rules for more efficient geometry compression.

Table 1 shows that a lossless compression ratio greater than 15 to 1 can be achieved on the meshes shown in Fig. 2.

References

[1] J. Rossignac and P. Borrel, "Multi-resolution 3d approximations for rendering complex scenes," *Modeling in Computer Graphics*, pp. 455–465, 1993.

[2] J. Rossignac, "Edgebreaker:connectivity compression for triangle meshes," *IEEE Transaction on Visualization and Computer Graphics, 5(1)*, 1999.

[3] C. Touma and C. Gotsman, "Triangle mesh compression," *Proc. of Graphics Interface'98*, pp. 26–34, 1998.

Model	Original File Size	Compressed File Size	Ratio	Bits/ Vertex
David	173 MB	10.1 MB	17	19.4
Lucy	533 MB	35.6MB	15.2	20.3

Table 1: Compression Results: 16 bit coordinate quantization was used for David and Lucy. All compressions were done on a Sun Sparc workstation with 58MB of RAM. Each submesh is compressed using Edgebreaker of [2] and Parallelogram Prediction of [3].