Streaming Compression of Triangle Meshes (abstract)

Martin Isenburg*
University of North Carolina
at Chapel Hill

Peter Lindstrom
Lawrence Livermore
National Laboratory

Jack Snoeyink
University of North Carolina
at Chapel Hill

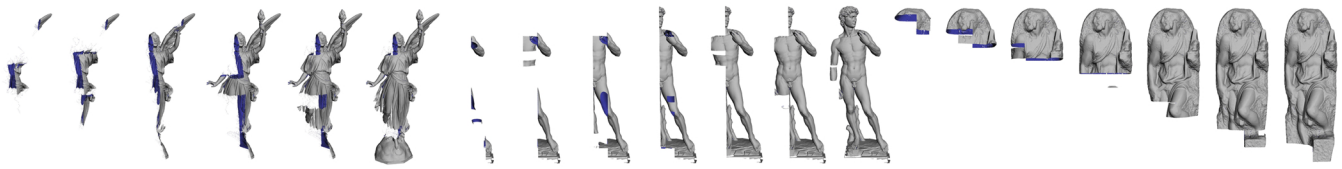

Figure 1: The original triangle orderings of lucy (28 million triangles), david (56 million triangles), and st. matthew (372 million triangles).

1 Introduction and Previous Work

Current mesh compression schemes completely disregard the original triangle and vertex order of the input mesh. They encode the triangles in an order that is derived from systematically traversing the connectivity graph and the 3D positions associated with the vertices in the order they are first encountered. The main compression gain comes from coding the mesh connectivity with only 1 to 2 bits per triangle. Standard indexed formats, in contrast, store three indices per triangle that each require at least $\log_2(v)$ bits where v is the number of vertices. Such formats not only specify the connectivity but also a particular permutation of the mesh elements as vertices and triangles can be listed in any order. By bringing the mesh elements into a more "canonical" order these compression schemes eliminate the costs for specifying the permutation.

Although such compression strategies achieve the best known bit-rates for connectivity coding, they struggle with large meshes. Before compression can start they need to construct data structures that support the topological adjacency queries that are needed for traversing the connectivity graph. For meshes that contain hundreds of millions of triangles, such as the 3D scans of Michelangelo's statues [Levoy et al. 2000], the construction of these temporary data structures with the limited main memory available on common PCs is difficult. Either the meshes must be cut into smaller pieces [Ho et al. 2001] or external memory data structures [Isenburg and Gumhold 2003] must be used. But both these approaches are IO-inefficient and require large amounts of temporary disk space.

Another drawback of current compression schemes is that they always expect the entire mesh as input. This means that a newly generated mesh needs to be stored at least once in uncompressed form (i.e. in order to hand it to the compressor) and that compression of a mesh cannot start until its generation is completed.

An executable of compressor and decompressor as well as a MSVC 6.0 project library of our compression API is available on the demo webpage.

```
void set_bbox(float* min, float* max);
void set_num_verts(int nverts);
void set_num_faces(int nfaces);

bool open(FILE* file, int bits, int delay=0);
bool write_vertex(float* v_pos);
bool write_triangle(int* t_idx, bool* t_final);
bool close();
```

```
                                  bool compress( FILE*  file,
                                                 int    bits,
                                                 int    nverts,
                                                 int    nfaces,
                                                 float* v_positions,
                                                 int*   t_indices);
```

Figure 2: An API for streaming (left) versus standard (right) compression.

2 Streaming Compression

We radically depart from the traditional approach to mesh compression and propose a scheme that incrementally encodes a mesh in the order it is given to the compressor while using only minimal memory resources. This makes the compression process basically transparent to the user and practically independent of the mesh size. This is especially beneficial for compressing large meshes where previous approaches spend significant amounts of memory, disk space, CPU time, and file I/O on pre-processing, whereas our scheme can start compressing after having been given the first few triangles.

*isenburg@cs.unc.edu http://www.cs.unc.edu/ ~ isenburg/smc

mesh	ordering	delay = 0			= 250		= 10,000		ooc-compressor [IG2003]				
		rate	time	RAM	rate	time	rate	time	rate	prepr.	compr.	RAM	disk
lucy	original	13.6	1	37	12.1	2	8.5	2	1.9	19	5	128	0.9
	breadth	3.5	1	1.6	2.6	1	3.4	2	bpv	min	min	MB	GB
$david_{1mm}$	original	15.9	2	4.8	4.9	3	3.8	4	1.8	36	14	192	1.7
	spectral	14.3	2	1.8	6.6	3	5.9	4	bpv	min	min	MB	GB
st.	original	15.6	14	5.2	4.8	20	3.9	28	1.8	7	4	384	11
matthew	spatial	13.6	15	4.0	6.6	23	5.3	29	bpv	hrs	hrs	MB	GB

Table 1: Comparing connectivity rates [bpv], timings [min], and memory use [MB] of our streaming compressor with [Isenburg and Gumhold, 2003].

We have implemented a *writer* (see Figure 2) and a corresponding *reader* through which compressed meshes can be written and read in increments of single vertices and triangles. The only requirement is that vertices are written before being referenced by a triangle and that vertices are *finalized* with the triangle that references them for the last time (e.g. simply by a boolean flag).

In Table 1 we compare our streaming compressor to that of Isenburg and Gumhold [2003]. For the "St. Matthew" they spend 7 hours creating an 11 GB data structure on disk before actual compression begins, which takes another 4 hours and uses 384 MB of RAM on a 2.8 GHz Pentium IV. In contrast, running on a 1.1 GHz mobile Pentium III we compress this model in 15 minutes using only 6 MB of RAM and no temporary disk space. Vertices are written before their first triangle and triangles are written in the original order (see Figure 1) and one alternative order. While geometry compression rates (not reported) are similar, their 11 hour/11 GB effort pays off with superior, state-of-the-art connectivity compression rates. But so far we have not reordered a single triangle.

When our compressor follows the exact triangle order in which the mesh is written, it generally needs to store at least $\log_2(width)$ bits per triangle, where *width* is the number of previously referenced, yet unfinalized vertices. We can significantly improve compression by employing a small *delay buffer* within which the compressor locally reorders triangles. It greedily brings them into a vertex-connected order that often allows avoiding those $\log_2(width)$ bits. A delay buffer of 10,000 triangles, for example, gives average connectivity rates of 4 to 5 bits per vertex [bpv] while slowing compression by a factor of two (not optimized yet) and increasing the memory footprint by only 5 MB.

References

HO, J., LEE, K., AND KRIEGMAN, D. 2001. Compressing large polygonal models. In *Visualization'01 Proceedings*, 357–362.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH 2003 Proceedings*, 935–942.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The Digital Michelangelo Project. In *SIGGRAPH 2000*, 131–144.