submitted (5/11/1998) to the Tenth Canadian Conference on Computational Geometry

A Framework for Memory-Efficient Levels of Detail

André Guéziec [1] Gabriel Taubin [1]

IBM T. J. Watson Research Center

Abstract

We introduce a representation, called a Multi-Level Progressive Mesh, for managing Levels of Detail (LODs) of a triangular mesh, allowing to store in memory an arbitrary number of LODs, with only about a 10 % increase in memory usage with respect to the highest resolution mesh. We assume that LODs were generated by a previous process, and concentrate on representing them efficiently, allowing progressive display and transmission. Our format is particularly adapted for changing LODs interactively (including smooth transitions, or "geomorphs"). We show snapshots of our implementation on a Pentium (TM) 133 Mhz PC, using VRML and Java.

One aspect of this approach is to encode the LOD hierarchy using an array of *representatives*, one for each vertex except a few. The same array provides information to access *all the LODs*. Another aspect is to partition the vertices and triangles of the mesh in several levels of details (by assigning an integer level to each vertex and triangle), and to re-enumerate the vertices and triangles according to the levels.

Key-words : Triangular Mesh, Levels of Detail (LODs), Progressive Transmission.

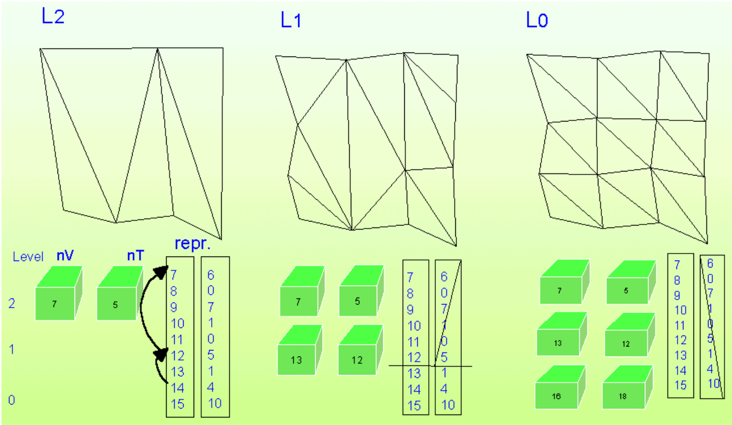

Figure 1: Accessing different levels of detail of the multi-level representation.

1 Introduction

Much of the recent research on Levels of Detail (LODs) of a triangular mesh has concentrated on developing LOD generation algorithms, and (generally) has not addressed the issue of representing and using LODs efficiently. By default, it is assumed that each LOD is represented and transmitted as a mesh of its own, and a display program for instance will choose either one for display at run time. Hoppe's Progressive Meshes (PM) [1] is a notable exception. In PM, the mesh is represented as a base mesh and a series of "vertex split" operations, specifying how to insert a vertex and two triangles at a specific location. To switch from one level to the next, all the intermediate vertex splits involved (or their inverse edge collapses) must be applied in sequence. The framework presented here is more general, allowing to switch directly from any LOD to any other LOD. The LODs can be computed using vertex clusterings, which are more flexible than edge collapses (which we relied on in [2]). De Floriani *et al.* [3] use a Directed Acyclic Graph representation. They don't explain how it applies to progressive transmission. The method in [4] used meshes with subdivision connectivity; obtaining them was quite involved. Other relevant work was discussed in [2].

Our starting point is the output of any triangle mesh simplification (polygon reduction) tool, producing a sequence of meshes by clustering vertices. (The relevance of the LOD hierarchy will depend upon the quality of the tool, but here which tool is used is unimportant.) We store the vertices and triangles of the full resolution mesh, together with a mapping between vertices of the LODs. This mapping is stored using an array of vertex representatives, no larger than the number of vertices in the highest resolution (i.e., original) mesh. After partition of the original mesh vertices and triangles into levels, and re-enumeration of such elements, the triangles belonging to a particular LOD are trivially identified, and the particular vertex mapping relative to this LOD can be quickly and efficiently extracted from the vertex representatives array.

2 Multi-Level Progressive Mesh

A *triangular mesh* is composed of a set of vertices and a set of triangles, each triangle being a triple of vertex references. A Multi-Level Progressive Mesh is as follows:

- Vertices and triangles are assigned a level starting from zero (most detailed level) to $L-1$ (coarsest level). Both are enumerated in order of decreasing level, and the maximum index n_l for a vertex of a given level l is stored.
- For each vertex v with a level $< L{-}1$ a *representative* is supplied. A representative is a reference to another vertex, with a higher level (and lower vertex number) which is substituted for the vertex v. Representatives define a graph that is a forest. A vertex has either exactly 0 (Level $L-1$) or 1 representative. This forest can be conveniently stored using an array with one entry per vertex.
- From this information, L LODs can be efficiently computed: each LOD l, $0 \leq l \leq L-1$ uses vertices and triangles of levels $l, l+1, \ldots, L-1$. For each such triangle, if a vertex reference is larger than n_l, then we follow the forest of representatives as shown in Fig. 1 until we fall below n_l. The forest of representatives can be path-compressed for speed-up (see Fig 2). The cost of pointing directly to the roots from each node is slightly super-linear in terms of the number of nodes (see [2] for a reference). By substituting vertex references in triangles with their corresponding forest root, we can switch directly from any level to any other level without having to build intermediate levels (contrary to PM[1]). There is no floating point operation involved.
- Vertices of the LODs do not have to be a proper subset of the original vertices (although this is more convenient): when evolv-

[1] IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, {gueziec,taubin}@watson.ibm.com

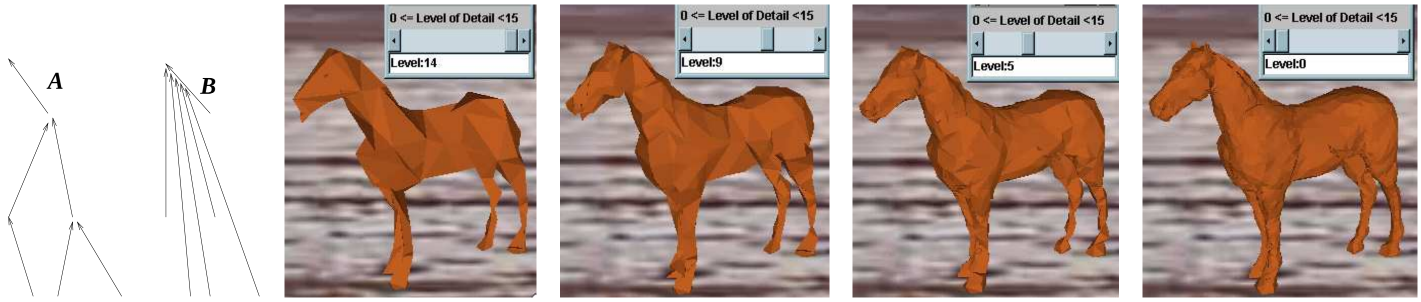

Figure 2: Left: Tree of representatives before (A) and after (B) path compression. Right: horse model (courtesy of Rythm & Hues). LODs with 247, 665, 1519 and 4350 triangles are accessed interactively on a Pentium (TM) 133 Mhz PC.

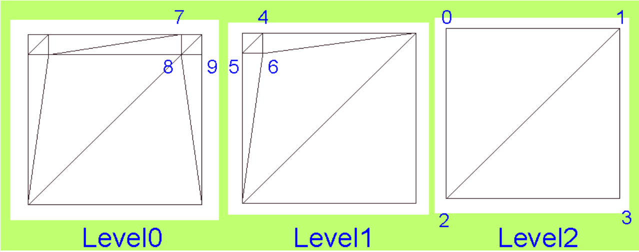

Figure 3: Building a Multi-Level mesh from vertex clustering information: vertex enumeration.

```
#3-level progressive mesh
{#level 2 vertices  (4)
-3.0,  3.0,   3.0,  3.0,    -3.0, -3.0,
 3.0, -3.0}
{#level 2 triangles (2), followed with
 #                    representatives
6, 2, 8,              0, 1,
3, 8, 2}
{#level 1 vertices  (3)
-2.3, 3.0,    -3.0, 2.3,      2.3, 2.3}
{#level 1 triangles (4)
4, 6, 7,              1,
5, 2, 6,
6, 4, 5,
4, 0, 5}
{#level 0 vertices  (3)
2.3, 3.0,      2.3, 2.3,     3.0, 2.3}
{#level 0 triangles (no representatives
            necessary)
8, 3, 9,
9, 1, 8,
7, 8, 1,
8, 7, 6 }
```

Figure 4: Exemplary ascii file for representing the mesh of Fig.3 with three progressive LODs (our software accepts this format). # signs precede comments.

ing from Level l of the triangular mesh to Level $l-1$ (increasing the resolution), positions and properties (color, texture coordinates, surface normal) of representatives of Level $l-1$ vertices may be changed by reading them from a secondary array (or list), the primary array being used for storing the original values.

Fig. 4 shows a simple file format for a Multi-Level Progressive Mesh (for the mesh of Fig. 3). We supply batches of vertices and triangles (like typical triangular meshes), with the difference that some triangles use vertex indices potentially referring to vertices in missing batches. In Fig. 4, the line `6,2,8, 0,1,` can be interpreted: when the triangle (6,2,8) is read from the storage or network, only vertices 0 to 3 can be referenced (this single vertex batch was read so far). Vertex 6 requires a representative (this is 0) as well as 8 (1). The next time 8 is read, its representative should not be specified again.

3 Method to Build our Representation

A Multi-Level Progressive Mesh can be easily built using vertex clustering information provided by any polygon reduction tool (edge collapses and some vertex removals are examples of vertex clusterings; in practice we have used [2]). We require the vertices and triangles of the most detailed mesh and for each clustering, a new set of vertices (of the mesh after clustering) and a mapping between the vertices of the previous mesh and the new vertices.

Vertices and triangles are assigned levels and are re-enumerated: For each remaining vertex after clustering, we identify its ancestors in the previous mesh using the mapping provided. Among its ancestors, one vertex is selected as a "preferred" ancestor based on geometric proximity (other criteria are possible). All vertices that are not preferred ancestors are assigned Level 0, and the largest indices (e.g., 7, 8 and 9 in Fig. 3). We also identify the triangles that become degenerate during the clustering and assign them Level 0 as well. The remaining vertices (0 through 6 in Fig. 3) and triangles are re-enumerated, the mapping adjusted to take the re-enumeration into account, and the operation repeated, whereby this time Level 1 is assigned. We stop when all clusterings are processed.

4 Applications

"Streaming": A file as in Fig. 4 can be transmitted progressively: a new level can be displayed (or processed) as soon as the corresponding batch of vertices and triangles was read.

Memory Efficient LODs: Assuming no compression, we perform a simple byte count for specifying a generic mesh, ignoring vertex and triangle properties, and supposing that there are n vertices and approximately $2n$ triangles (this depends upon the surface genus and number of boundaries; it is exact for a torus) and that 4 bytes are used to store each vertex coordinate (typically a 4 byte `float`) and vertex index (4 byte `int`). The generic mesh would be stored using $36n$ bytes. Our representation would use *less* than $40n$ bytes, since vertex representatives, the sole addition, shall not be supplied for vertices of level 0: the additional cost factor is at most $40/36 \simeq 1.1$. There are various possibilities for a compressed encoding of this representation, but they are not covered in this abstract.

Smooth Transitions between LODs: When detail is added to the triangular mesh by lowering the level from l to $l-1$ the vertices of Level $l-1$ are introduced in the mesh. All added vertices have a representative in Level l. The new triangles are determined as explained above, but for the new vertices, the coordinates of their representative are first used, resulting in a mesh that is geometrically the same as the Level l mesh. Then, gradually, the coordinates are interpolated linearly from that position to the new coordinates using a parameter λ that varies between 0 and 1.

REFERENCES

[1] H. Hoppe. Progressive meshes. In *Siggraph*, pages 99–108, New Orleans, August 1996. ACM.

[2] A. Gueziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *VRML 98*, pages 25–31. ACM, February 1998.

[3] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Visualization 97*, pages 103–110. IEEE, 1997.

[4] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberry, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Siggraph*, pages 173–182, Los Angeles, August 1995. ACM.