DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering
Category: Scientific Research

Michael Shafae and Renato Pajarola

Computer Graphics Lab
Department of Information & Computer Science
University of California, Irvine
mshafae@ics.uci.edu, pajarola@acm.org

1. Motivation

Multiresolution modelling techniques are important to cope with the increasingly complex polygonal models available today such as high-resolution isosurfaces, large terrains, and complex digitized shapes. Large triangle meshes are difficult to render at interactive frame rates due to the large number of vertices to be processed by the graphics hardware. Level-of-detail (LOD) based visualization techniques allow rendering the same object using triangle meshes of variable complexity. Thus, the number of processed vertices is adjusted according to the object’s relative position and importance in the rendered scene. Many mesh simplification and multiresolution triangulation methods have been developed to create different LODs, sequence of LOD-meshes, and hierarchical triangulations for LOD based rendering. Although reducing the amount of geometry sent to the graphics pipeline elicits a performance gain, a further optimization can be achieved by the use of optimized rendering primitives, such as triangle strips.

Triangle strips have been used extensively for static mesh representations since their widespread availability through tools such as the classic tomesh.c program, Stripe and the more recent NVidia NVTriStrip tools. However, using such triangle strip representations and generation techniques is not practical for a multiresolution triangle mesh. The problem of representing the stripped mesh and maintaining the coherency of the triangle strips is compounded when used with LOD-meshes. In view-dependent meshing methods the underlying mesh is in a constant state of flux between view positions. This poses a significant hurdle to surmount for current triangle strip generation techniques for two core reasons. First, triangle strip generation techniques tend to require too much CPU time and memory space to be practical for interactive view-dependent triangle mesh visualization. Secondly, most triangle strip generation techniques focus on producing optimized strips, but not managing the strips in light of continuous changes to the mesh. That is, for each new view position a new stripification must be computed. Our approach, on the other hand, manages triangle strips in such a way that reconstructing the entire stripification never has to be done. Instead, it either grows triangle strips, shrinks triangle strips, or recomputes triangle strips only for small patches when necessary.

In this short paper and video, DStrips is presented. DStrips is a simple yet effective triangle strip generation algorithm and data structure for real-time triangle strip generation, representation, and rendering of LOD-meshes. The im-
plementation presented in this paper is built on a LOD-mesh using progressive edge collapse and vertex split operations. It is based on a half-edge representation of the triangle mesh connectivity. However, DStrips is not tightly coupled to any one particular LOD-mesh. DStrips is easily adapted to any LOD-mesh so long as the mesh provides a mapping from an edge to its associated faces and vice versa. Also, the edges of a face must maintain a consistent ordering and orientation in the LOD-mesh. Figure 1 presents an example screenshot of pseudo-colored triangle strips of a view-dependently simplified LOD-mesh that were generated by DStrips.

2. Production Process

The video was produced by capturing the running sample implementation of DStrips on an Apple Macintosh. The PC was configured with a Pentium 4 2.8 GHz processor, 512 MB of main memory and an ATI Radeon 9700 graphics board. While the application was running on the PC, the video output was redirected to video capture hardware on an Apple Macintosh. The DStrips application was scripted and a series of short video captures were collected.

3. Innovative Aspects

The main contributions of our approach are:

- A simple triangle strip data structure based on maintaining the orientation of a face in relationship to the previous face in the strip.
- Augmenting the underlying LOD-mesh with triangle strip savvy edge collapse and vertex split operations.
- An efficient partial triangle strip destruction and re-stripping algorithm.

Unlike other LOD-mesh with some sort of triangle stripping support, DStrips does not merely shorten the initially computed triangle strips. Rather, DStrips dynamically shrinks, grows, merges and partially recomputes strips. Table 1 briefly compares other approaches which couple triangle stripification techniques which a clear distinction can be drawn between the techniques which

<table>
<thead>
<tr>
<th>Name</th>
<th>Algorithm</th>
<th>Stripification</th>
<th>Strip Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>DStrips</td>
<td>Online</td>
<td>Dynamic</td>
<td>Shorten, Grow, Merge, Partial Re-Strip</td>
</tr>
<tr>
<td>Tunneling Strips</td>
<td>Online</td>
<td>Dynamic</td>
<td>Repair &amp; Merge (Tunneling Operation)</td>
</tr>
<tr>
<td>Stripes</td>
<td>Offline</td>
<td>Static</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Skip Strips (Strips)</td>
<td>Pre-Process</td>
<td>Static</td>
<td>Resize Pre-Computed Strips</td>
</tr>
<tr>
<td>MultiResolution</td>
<td>Pre-Process</td>
<td>Static</td>
<td>Resize Pre-Computed Strips</td>
</tr>
</tbody>
</table>

Table 1: A comparison of triangle stripification techniques. Note that a clear distinction can be drawn between the techniques which dynamically manage the triangle strips and those which shorten pre-computed triangle strips.

To illustrate the novelty of our approach, experiments were performed on a Sun Microsystems Ultra 60 workstation with dual 450MHz UltraSparc II CPUs and an Expert3D graphics card. Table 2 shows the sizes of the different models we used for testing DStrips.

![Table 2](image)

Table 2: Size of the models used in the experiment.

Table 3 shows the average number of faces, LOD-updates and triangle strips encountered each frame. The time to perform the edge collapse and vertex split updates each frame is also recorded here since it is independent of the rendering mode. The average number of triangle strips per frame is given for the three stripping configurations: adjacency stripping, greedy stripping allowing swap operations and greedy stripping without swap operations (strictly left-right). One can see from Table 3 that adjacency stripping generates fewer strips than greedy stripping, in particular if strict left-right alternation is enforced.

![Table 3](image)

Table 3: Per frame average numbers of rendered triangles, LOD-mesh updates, and time to perform mesh updates. The average number of triangle strips is divided into adjacency stripping (ADJ) as well as greedy stripping with swap (GS) and without swap operations (GNS).

Table 4 presents rendering performance tests of DStrips. Note that DStrips’ overall display time is the sum of stripping and rendering (immediate or vertex array mode). It is clear from Table 4 that DStrips can maintain a good stripification without any overhead introduced compared to standard rendering. In fact, in the basic immediate mode rendering DStrips is able to dynamically maintain triangle strips and render them in less time than a standard indexed triangle mesh rendering requires for the same LOD-mesh. The improvements are in the range of 5% to 20%. If vertex arrays are used to represent the triangle strips, further significant rendering improvements can be seen. Note that DStrips efficiently allows bookkeeping of vertex arrays which cannot easily be done in a standard LOD-mesh framework.

![Table 4](image)

Table 4: Rendering and stripping times given in milliseconds and averaged per frame, with and without DStrips. Compared to standard rendering (Indexed), DStrips overall display time is the sum of stripping and rendering. For DStrips, plain immediate mode rendering time is reported as well as using vertex arrays.

4. End User Applications

DStrips, a simple and efficient method to dynamically generate triangle strips for real-time level-of-detail (LOD) meshing and rendering. Built on top of a widely used LOD-mesh framework using a half-edge data structure based hierarchical multiresolution triangulation framework, DStrips has shown efficient data structures and algorithms to compute a mesh stripification and to manage it dynamically through strip grow and shrink operations, strip savvy mesh updates, and partial re-striping of the LOD-mesh. The approach taken in DStrips can be readily adapted for use with computer graphics applications that deal with large polygon models that need to be visualized in real-time. DStrips provides a way to trade some memory space for organizing what often is left as a soup of triangles into orderly triangle strips. Example applications where DStrips’ techniques can be applied are GIS applications and CAD/CAM applications. Compared to other methods, DStrips uses a simpler and more compact data structure, is easily extended to incorporate application specific details, and can be adapted to other LOD-mesh frameworks.

References

