Hierarchical Structures For Dynamic Polygonal Simplification

TR 96-006

David Luebke

Department of Computer Science

University of North Carolina at Chapel Hill

Chapel Hill, North Carolina

Abstract

This paper presents a novel technique for simplifying polygonal environments. Unlike previous multi-resolution methods, the technique operates dynamically, simplifying the scene on-the-fly as the user's viewing position shifts, and adaptively, simplifying the entire database without first decomposing the environment into individual objects. Each frame the simplification process queries a spatial subdivision of the model to generate a scene containing only those polygons that are important from the current viewpoint. This spatial subdivision, an octree, classifies the polygons of a scene according to which regions of space the polygons intersect. When the volume of space associated with an octree node occupies less than a user-specified amount of the screen, all vertices within that node are collapsed together and degenerate polygons filtered out. An *active list* of visible polygons is maintained for rendering. Since frame-to-frame movements typically involve small changes in viewpoint, and therefore modify the active list by only a few polygons, the method can take advantage of temporal coherence for greater speed. The algorithm has been tried on a wide range of models. The current implementation increases rendering performance two to four times with only slight degradation of image quality, or ten to seventeen times with more drastic degradation. On larger, more complex models the algorithm should perform even better. This is a promising sign since models of every category are growing in size and complexity, continuously outpacing the ability of graphics hardware to render them.

1. Introduction

Polygonal simplification is at once a very current and a very old topic in computer graphics. As early as 1976 James Clark described the benefits of representing objects within a scene at several resolutions, and flight simulators have long used hand-crafted multi-resolution models of airplanes to guarantee a constant frame rate [Clark 76, Cosman & Schumacker 81]. Recent years have seen a flurry of research into generating such multi-resolution representations of objects automatically by simplifying the polygonal geometry of the object. This paper presents a new approach which simplifies the geometry of entire scenes dynamically, adjusting the simplification as the user moves around.

The rest of this introduction provides a brief overview of the algorithm and describes results achieved with the current implementation. The next section steps back to review related work in polygonal simplification, and explains the motivation for a new approach. The data structures and algorithm are then revisited in detail, along with some schemes to optimize and parallelize the algorithm. A few remarks and some directions for future work conclude the paper.

1.1 Hierarchical Dynamic Simplification

The new approach, referred to here as h*ierarchical dynamic simplification* or HDS, has some novel features. Rather than representing the scene as a collection of objects, each at several resolutions, the entire model comprises a single large data structure. This data structure, an octree, is queried dynamically to generate an simplified scene. The octree contains information only about the vertices and triangles of the model; valid topology is neither required nor preserved. Each node in the octree contains one or more vertices, each with an *importance* rating based on local criteria such as curvature. HDS operates by collapsing all of the vertices within an octree node together to the single most important vertex. Triangles whose corners have been collapsed together become redundant and can be eliminated, decreasing the total polygon count. Likewise, a node may be expanded by splitting its representative vertex into the representative vertices of the node's children. Triangles filtered out when the node was collapsed become visible again when the node is expanded, increasing the polygon count.

The entire system is dynamic; nodes to be collapsed or expanded are continuously chosen based on their projected size. The screenspace area of the node is monitored: As the viewpoint shifts certain nodes in the octree will fall below the size threshold. These nodes will be *folded* into their parent nodes and the now-redundant triangles removed from the display list. Other nodes will increase in apparent size to the user and will be *unfolded* into their constituent child nodes, introducing new vertices and new triangles into the display list. The user selects the screenspace size threshold and may adjust it during the course of a viewing session for interactive control over the degree of simplification. Since many nodes will be folded and unfolded each frame, efficient methods for finding, adding, and removing the affected triangles are crucial.

1.2 Results

HDS has been implemented and tested on a Silicon Graphics Onyx system with RealityEngine2 graphics hardware. The models tested span a range of categories. Bone6 and Skin are medical datasets created from the Visible Man volumetric dataset. Sierra is a terrain database originally acquired from satellite topography. Torp is a complex CAD model of the torpedo room on a nuclear submarine. Bunny is a digitized model from a laser scanner. Finally, Horse is a hand-crafted dataset manually digitized from a clay model. The table below details the size of each database and the time in milliseconds required to render the entire object as a GL display list. In addition, the table lists one or more views of the model made with different simplification thresholds. Rendering times and polygon counts for these simplification thresholds are also provided in the table.

Model Name	Category	Figure	Triangles	% Reduction	Render Time (msec)	% Reduction
Bone6 (Original)	Medical	1	1136797		2930	
Bone6 (Simplification 1)	Medical	2	323996	29%	1360	46%
Bone6 (Simplification 2)	Medical	3	67241	6%	270	9%
Skin (Original)	Medical	4	480934		1230	
Skin (Simplification 1)	Medical	5	96285	20%	390	32%
Skin (Simplification 2)	Medical	6	6344	1%	15	1%
Skin (Simplification 3)	Medical	7	136795	28%	580	47%
Sierra (Original)	Terrain	8	162690		540	
Sierra (Simplification)	Terrain	9	33443	21%	150	28%
Torp (Original)	CAD	10	420466		1370	
Torp (Simplification)	CAD	11	105141	25%	500	36%
Bunny (Original)	Scanned	12	69451		170	
Bunny (Simplification 1)	Scanned	13	11372	16%	30	18%
Bunny (Simplification 2)	Scanned	14	3069	4%	10	6%
Horse (Original)	Handmade	15	21996		40	
Horse (Simplification)	Handmade	16	9212	42%	25	63%

Overall, HDS provided a two to four times increase in rendering performance for these scenes with only slight image degradation, and a ten to seventeen times increase with more drastic degradation. Note the discrepancy between the rendering times and the actual polygon counts. This is due to the overhead associated with traversing the linked list of active triangles; the rendering hardware is optimized for GL display lists. As with other simplification techniques, the results depend greatly on the model being simplified. Unlike other simplification techniques, the results presented here also depend on the particular view being rendered, so each view described in the table is also pictured in the figures.

2. Previous Work

Polygonal models currently dominate the field of interactive three-dimensional computer graphics. This is largely because their mathematical simplicity allows rapid rendering of polygonal datasets, which in turn has led to widely available polygon rendering hardware. In addition, polygons serve as a sort of lowest common denominator for computer models, since almost any model representation (spline, implicit-surface, volumetric) may be converted with arbitrary accuracy to a polygonal mesh. For these and other reasons, polygonal models are the most common representation for visualization of medical, scientific, and CAD datasets.

In many cases the complexity of such models exceeds the ability of graphics hardware to render them interactively. The methods employed to alleviate this problem may be roughly categorized into three approaches:

- Augmenting the raw polygonal data to convey more visual detail per polygon. Features such as gouraud shading and texture mapping fall into this category.
- Using information about the model to cull away large portions of the model which are occluded from the current viewpoint. The visibility processing approach described by Seth Teller and Carlo Sequin falls into this category [Teller 91].
- Simplifying the polygonal geometry of small or distant objects to reduce the rendering cost without a significant loss in the visual content of the scene. This is the category which concerns this paper.

Polygonal simplification algorithms may be further divided into two subcategories: those which preserve the topology of the original model and those which do not. Topology-preserving techniques are more common. In the re-tiling approach described by Turk, a new set of vertices are distributed across the object's surface by mutual repulsion and connected in a manner that matches the connectivity of the initial mesh [Turk 92]. Triangle mesh decimation, presented by Schroeder et al, identifies and removes unimportant vertices from the model, then locally

re-triangulates the resulting holes in the mesh [Schroeder 92]. Varshney calculates inner and outer "offset surfaces" at a user-specified distance from the model, then greedily re-triangulates the original mesh with triangles that do not intersect the offset surfaces [Varshney 94]. Eck and colleagues partition the surface into "Voronoi tiles" which capture the shape and topology of the original model. From this they generate a "base mesh" of triangles, each of which can be recursively subdivided to approximate the initial surface as closely as necessary.

Very few topology-discarding schemes have been proposed to date. Voxel-based object simplification converts the surface into a volumetric representation and low-pass filters the volume dataset, possibly simplifying the topology by discarding small holes in the model. The resulting lower-resolution volume is then reconverted into polygons [Kaufman 95]. Rossignac & Borrel apply a uniform grid to the surface and cluster all vertices within each grid cell together, filtering out degenerate polygons. Applying multiple grids provides multiple static "snapshots" of the object at varying resolutions.

3. Motivation

In practice, choosing a polygonal simplification algorithm involves tradeoffs. None of the algorithms described above apply equally well to all classes of polygonal models. Indeed, so numerous and varied are the sources of such models that a perfect all-encompassing simplification algorithm seems unlikely. For example, Turk notes that the re-tiling approach is best suited for smoothly curved organic objects [Turk 92], while Rossignac and Borrel target industrial CAD models [Rossignac 92]. Many algorithms are designed to preserve the topology of the input mesh, which limits the amount of simplification they can accomplish. Topology-preserving algorithms also require that their input meshes must possess a well-defined topology to preserve. Other algorithms are designed to simplify to any degree necessary, and do not as a rule preserve topology. Each scheme has advantages and disadvantages, and the choice of which to use depends entirely on the application and models in question.

The algorithm presented in this paper was conceived for very complex hand-crafted CAD databases, a class of models for which existing simplification methods are often inadequate. Real-world CAD models are rarely topologically sound, and may entail a great deal of clean-up effort before a topology-preserving algorithm can be applied. Sometimes such models even come in 'polygon-soup' formats which do not differentiate the various objects in the scene, instead describing the scene as an unorganized list of polygons. Few existing algorithms, and none of those described above, deal elegantly with such models. The approach described and by DeRose et al [Chamberlain et al 95], and elements of the approach described by Shirley and Maciel [Maciel 95], show promise for polygon-soup databases: the first replaces distant geometry with colored cubes and the second with textured cubes. However, neither method actually provides a simplified polygonal description of the geometry. Such a description is often crucial: dynamic lighting of the scene, for instance, absolutely requires underlying geometry. Furthermore, neither of the mentioned schemes applies well to nearby portions of the scene, which may still comprise more polygons than the hardware can render interactively. Therefore the need still exists for a polygonal simplification algorithm which operates on completely unorganized models.

Even when the model format correctly delineates individual objects, simplifying very complex CAD datasets with current schemes can involve many man-hours. To begin with, topology-preserving schemes require that each object be broken up further into topologically correct pieces (typically the result of a few boolean CSG operations in CAD models). In addition, physically large objects must be subdivided. Consider a model of a ship, for example: the hull of the ship had better be divided into several sections, or the end furthest from the user will be tessellated as finely as the nearby hull. Finally, each simplification must be inspected for visual fidelity to the original object, and an appropriate switching threshold selected. This can be the most time-consuming step in the simplification of a complicated model with thousands of parts, but few existing techniques address automating the process. Ideally, a simplification scheme for very large CAD models should go further and eliminate the need for these tasks altogether.

These considerations led to a new approach with three primary goals. First, the algorithm should be very general, making as few assumptions as possible about the input model. This implies that the algorithm must deal robustly with topologically unsound models. Second, the algorithm should be completely automatic, able to simplify a complex unorganized model without human intervention. This implies that the algorithm must simplify the entire scene adaptively rather than simplifying objects within the scene. Third, the algorithm should be dynamically adjustable, supplying the system with a fine-grained interactive 'dial' for trading off performance and fidelity *at run time*. Such a dial is necessary for frame rate driven LOD management and is traditionally implemented by scaling the switching threshold for each object's levels of detail, but this method fails when the scene contains no explicit objects.

4. Data Structures

The key HDS data structures are the aforementioned octree and the *active list* of triangles to be rendered. The octree provides a spatial decomposition of the model; each node in the octree refers to a volume of space containing one or more vertices. Associated with each node are several fields:

- **Id**: a 64-bit tag which labels the path from the root of the octree to the node. 3 bits describe which branch to take at each level of the tree.
- **Depth**: the depth of the node in the octree. The depth and id together uniquely identify the node.
- **Tris**: an array of triangles with exactly one corner in the node. If the node is *active* (not currently collapsed), these triangles must be displayed.
- **Subtris**: an array of triangles which will be filtered out if the node is collapsed, or re-introduced if the node is expanded. The subtris of a node are those triangles which have two or three corners within the node, but no more than one corner within any child of the node [figure].
- **RepVert**: the coordinates of the most important vertex in the node. The importance rating assigned to vertices follows that described in [Rossignac 92].
- **Status**: The active status of the node. A node may be labeled either Active, Inactive, or Leaf.

Active nodes must be descendants of other active nodes; children of inactive nodes must themselves be inactive. This gives rise to the notion of the *active tree*, defined as that contiguous portion of the tree consisting only of active nodes and rooted at the root node of the entire tree. Active nodes whose children are all inactive, or which have no children, are labeled Leaf. Since the subtris of each node are calculated in a preprocess, only these leaf nodes need to be maintained as the viewpoint shifts around. This is the key observation which makes dynamic simplification of the database possible.

The active list is a display list of those triangles which are currently visible, namely those triangles registered in the tris field of the active nodes. Expanding a leaf node appends its subtris to the active list; collapsing the node removes them. The active list is maintained in the current implementation as a doubly-linked list of triangle structures, each of which includes the following fields:

- **Corners**: pointers to the three octree nodes whose representative vertices comprise the original three corners of the triangle.
- **Proxies**: pointers to the *first active ancestor* of each of the three corner nodes. The representative vertices of these nodes are the coordinates to which each corner of the triangle will be collapsed.

5. The Algorithm

The chief tasks of the HDS algorithm are dynamically updating the octree to reflect the viewer's changing position, and rendering the active list. The functions adjustNode(), collapseNode(), and expandNode() handle updating the tree. AdjustNode() recursively descends the active tree, checking the projected screenspace area of each node against the user-specified size threshold. Each leaf node in the active tree whose size falls above the threshold is unfolded by expandNode(), while those leaf nodes whose parents' size fall below the threshold are folded by collapseNode(). CollapseNode(N) removes N's subtris from the active list and visits each of N's child nodes, updating the proxies of their tris to point at N. ExpandNode(N) appends N's subtris to the active list and updates the proxies of N's tris to point at the appropriate child of N. Rendering the active list is considerably simpler. The function drawActiveTris() simply traverses the linked list in order, spinning over the active list each frame.

A number of simple optimizations improved the performance of the initial implementation considerably. AdjustTree() is continuously calculating the size of a node's screenspace projection. The most expensive step in this calculation is an inverse square root used to find the distance from the viewpoint to the node; however, temporal coherence suggests that this distance changes little from frame to frame. Storing last frame's distance with each node and using it as the seed for an iterative square-root algorithm accelerates the node size operation by an order of magnitude. Temporal coherence is also exploited to speed up the firstActiveAncestor() function, used heavily by CollapseNode() and expandNode(). FirstActiveAncestor(N) searches up the octree for the nearest ancestor of node N which is tagged Active or Leaf. Starting each search from the result of the last search speeds up adjustTree() still further.

On a multi-processor platform the HDS algorithm can be parallelized for more fundamental gains. Updating the octree and rendering the active list are basically independent tasks; splitting them into separate processes allows the user to move smoothly around even if the octree maintenance falls somewhat behind. Since drawActiveTris() reads from the active list and adjustTree() writes to it , the two functions are ideal threads for a shared-memory system such as the SGI Onyx architecture. Although the current implementation uses only these two processes, parallelizing the adjustTree() operation further seems straightforward. In this scheme several adjustTree() threads would wander around the octree searching for subtrees in need of maintenance.

6. Remarks

Polygonal simplification is a process of approximation. As with any approximation, a simplification algorithm taken to the limit should recover the original object being approximated. This holds true for the HDS algorithm: as the screenspace area threshold approaches subpixel size, the visual effects of collapsing vertices become vanishingly small. Note that the polygon counts of large and complex enough scenes will be reduced even under these extreme conditions. This is important; with complex CAD models, finely tessellated laser-scanned objects, and polygon proliferating radiosity algorithms all coming into widespread use, databases in which many or most visible polygons are smaller than a pixel are becoming increasingly common.

HDS was designed with large CAD datasets such as the Torp model in mind. Since these models are often wildly degenerate and sometimes even completely undifferentiated polygon soup, robustness was a primary goal. That the HDS algorithm can be successfully applied without modification to the wide range of models presented in this paper is a tribute to the robustness of the approach. The Bunny and Horse models are perhaps the least interesting; both are individual objects which can be simplified nicely by previous approaches. Dynamic terrain simplification is not new, but the Sierra model illustrates nicely how well the HDS general polygonal simplification applies to the more constrained domain of terrain meshes. The Bone6 and Skin datasets from the Visible Man project are an exciting and unforeseen application of the algorithm to the totally different domain of medical volume datasets. Bone6 contains over a million polygons and lies well beyond the interactive rendering capabilities of today's graphics workstations. Like the polygon soup CAD models, the Bone6 dataset contains no objects; conventional simplification methods would have little luck simplifying scenes like figure 1, with parts of the skeleton quite near the viewpoint. The near-interactive rendering rate achieved with this model indicates both the promise of the technique and the opportunity for future improvements.

7. Ongoing and future work

The field of dynamic polygonal simplification is quite new. This paper takes only the first tentative steps into that field; many possibilities to explore and improve this algorithm present themselves upon reflection. One straightforward enhancement is to collapse information such as color, normal, and texture data along with the coordinates of vertices. A simple version of color-collapsing has already been implemented in the existing HDS system and was used with the Torp and Sierra models. Investigating alternate spatial hierarchies should also prove fruitful. The octree provides a simple, regular decomposition of the scene, but a bottom-up bounding box approach might prove more optimal. Nothing in the HDS algorithm precludes overlapping or irregularly shaped cells; a bottom-up approach could (for example) make use of object information when available to cluster vertices together that belong to the same object before collapsing vertices belonging to different objects. This seems like an excellent way to minimize the visual artifacts introduced when two different materials are combined.

One interesting experiment would involve simplifying offscreen geometry more aggressively. Triangles which are completely contained within a node may be identified easily; it should be possible to design data structures which facilitate culling such triangles from the active list when the node is out of the viewing frustum. The importance function used to rank vertices is currently based entirely on the local geometry of the model. Designing new importance functions using more global information and based more on perceptual properties of the scene seems like rich area for research. Perhaps the most intriguing possibilities for future work lie in the notion of angular view-dependence, in which the color, normal, texture, and position of a node's representative vertex depend on the angle from which the user views the node.

8. References

[Chamberlain 95] Bradford Chamberlain, Tony DeRose, Dani Lischinski, David Salcsin, John Snyder. "Fast Rendering of Complex Environments Using a Spatial Hierarchy". University of Washington Tech Report # UW-CSE-95-05-02.

[Clark 76] Clark, James H. "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, Vol 19, No 10, pp 547-554.

[Cosman & Schumacker 81] Cosman, M, and R. Schumacker. "System Strategies to Optimize CIG Image Content". *Proceedings Image II Conference* (Scotsdale, Arizona), 1981.

[Kaufman 95] Taosong He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. "Voxel-Based Object Simplification". *Proceedings Visualization 95*, IEEE Computer Society Press (Atlanta, GA), 1995, pp. 296-303.

[Maciel 95] Maciel, Paulo, and Shirley, Peter. "Visual Navigation of Large Environments Using Textured Clusters," *Proceedings 1995 SIGGRAPH Symposium on Interactive 3D Graphics* (Monterey, CA), 1995, pp. 95-102.

[Rossignac 92] Rossignac, Jarek, and Paul Borrel. "Multi-Resolution 3D Approximations for Rendering Complex Scenes", pp. 455-465 in *Geometric Modeling in Computer* Graphics, Springer-Verlag, Eds. B. Falcidieno and T.L. Kunii, Genova, Italy, 6/28/93-7/2/93. Also published as IBM Research Report RC17697 (77951) 2/19/92.

[Schroeder et al 92] Schroeder, William, Jonathan Zarge and William Lorenson, "Decimation of Triangle Meshes," *Computer Graphics*, Vol 26 (SIGGRAPH 92).

[Teller 91] Teller, Seth, and Carlo Sequin. "Visibility Preprocessing for Interactive Walkthroughs", *Computer Graphics*, Vol 25 (SIGGRAPH 91).

[Turk 92] Turk, Greg. "Re-tiling Polygonal Surfaces," *Computer Graphics,* Vol 26 (SIGGRAPH 92).

[Varshney 94] Varshney, Amitabh. "Hierarchical Geometry Approximations", Ph.D. Thesis, University of North Carolina Department of Computer Science Tech Report TR-050