

HLODs for Faster Display of Large Static and Dynamic Environments

Carl Erikson

Dinesh Manocha

William V. Baxter III

Department of Computer Science

University of North Carolina at Chapel Hill

{eriksonc,dm,baxter}@cs.unc.edu

<http://www.cs.unc.edu/~walk/hlod>

ABSTRACT

We present an algorithm and a system for accelerated display of massive static and dynamic environments using hierarchical simplification. Given a geometric dataset, we represent it using a scene graph and compute levels of detail (LODs) for each node in the graph. We augment the LODs with automatically-generated *hierarchical levels of detail (HLODs)* that serve as higher fidelity drastic simplifications of entire branches of the scene graph. We extend the algorithm to handle a class of dynamic environments by incrementally recomputing a subset of the HLODs on the fly when objects move. We leverage the properties of the HLOD scene graph in our system, using them to render the environment in a specified image quality or target frame rate mode. The resulting algorithms have been implemented as part of a system named SHAPE. We demonstrate its performance on complex CAD environments composed of tens of millions of polygons. Overall, SHAPE is able to achieve considerable speedups in frame rate with little loss in image quality.

Keywords: interactive display, graphics systems, spatial data structures, level-of-detail algorithms, CAD

1 INTRODUCTION

Computer-aided design and scientific visualization applications regularly generate complex models that exceed the interactive visualization capabilities of current graphics systems. Today, large geometric databases contain tens or hundreds of millions of primitives, while typical high-end hardware can currently display only a few million at interactive frame rates. Several acceleration techniques that reduce the number of rendered polygons have been proposed. One is to precompute different levels of detail (LODs) of a given object or portions of an environment. At runtime, before rendering each frame, the appropriate LODs to display are selected so that coarser approximations are used for objects that are further away or contribute less to the scene.

Besides static environments, we are also interested in handling large dynamic scenes, which are common in design evaluation of large assemblies where a designer moves, adds, or deletes parts. Other examples of dynamic environments include animated scenes with articulated figures, simulation-based design, driving simulators, battlefield visualization, urban environment visualization, and entertainment software. All of these applications require that objects move, either by programmed behavior or interactive manipulation. It is therefore desirable that a rendering system be able to display dynamic environments, as well as static, at interactive frame rates.

The problem of computing LODs has been an active area of research over the last few years. Most of the earlier algorithms have focused on computing separate LODs of objects in the scene.

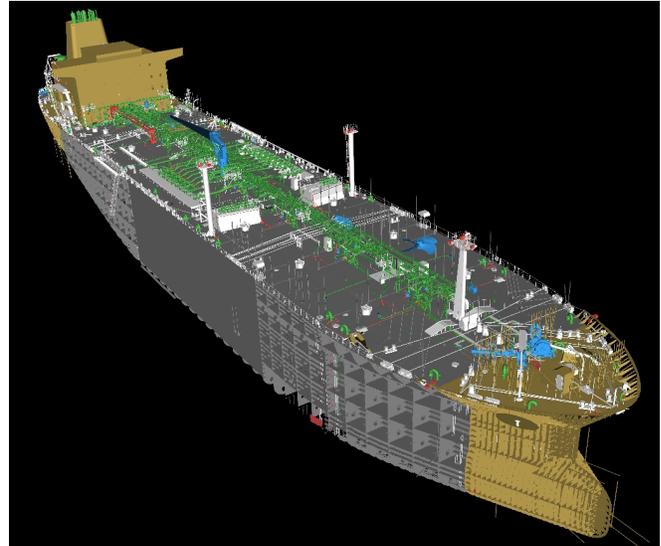


Figure 1: A view of the Double Eagle Tanker consisting of 126,630 objects and 82,361,612 triangles. Using LODs and HLODs, SHAPE renders this scene on a typical viewing path at between 1 and 8 frames per second on an SGI Infinite Reality. It achieves more than two orders of magnitude improvement in the frame rate with little loss in image quality.

Many researchers have also proposed techniques to accelerate the rendering of large environments using potentially visible sets (PVS), image-based representations (e.g. texture mapped primitives, point-based sampling), or view-dependent simplifications. Few if any of these techniques address dynamic environments.

1.1 Main Results

The main goal of this research is to devise an efficient and practical way to use LODs in a hierarchical manner to enable the interactive visualization of large environments. In this paper, we present a new approach based on *hierarchical levels of detail (HLODs)*, and the resulting system, called SHAPE, used for rendering massive datasets. HLODs are a generalization of the level-of-detail concept to hierarchical aggregations of objects. In contrast to conventional LODs of objects, HLODs are generated by simplifying separate portions of a scene together to create higher fidelity and drastic approximations.

For dynamic environments, we incrementally recompute HLODs as objects in the scene move. We describe a method for performing this recomputation asynchronously in parallel on shared memory multi-processor graphics systems. In practice, our method can efficiently handle large environments with a limited amount of object motion, i.e. scenes where relatively few objects are moved, inserted, or deleted from the scene graph.

The concept of using a hierarchy of levels of detail is not new. Clark introduced the abstract notion of *objects* and a hierarchy of levels of detail [Clar76]. Maciel and Shirley proposed the use of *impostors* and *meta-objects*, and rendered large static environments using image-based hierarchical representations [Maci95]. Erikson and Manocha [Erik98] used hierarchical levels of detail for

simplification culling. Hoppe demonstrated a hierarchical view-dependent LOD method applicable to terrain rendering [Hopp98]. Our contributions include a combination of techniques to automatically create and render geometric LODs and HLODs for large static and dynamic scenes. Some of the key features include:

- **Fidelity:** By grouping objects to create HLODs, we merge polygons from different objects during simplification. This merging increases the visual quality of drastic, or low-polygon-count, approximations.
- **Automatic Generation:** Given a large environment, our algorithm can automatically compute the HLODs of the scene graph without user intervention.
- **Generality:** Our approach is applicable to all polygonal environments and makes no assumptions about topological information or representation.
- **Efficiency:** We render LODs and HLODs using display lists, to make the best possible use of the performance of current high-end graphics systems. The HLOD recomputation algorithm can also use multiple processors on high-end graphics machines.
- **Flexibility:** Our HLOD scene graph structure allows our system to render in constant frame-rate mode or image-fidelity mode.

The memory required by this method is typically only twice that of the original model for static environments, but can be as high as a factor of six for dynamic scenes.

1.2 Main Advantages of HLODs

Traditional LOD generation methods work only on a single object at a time, so they can only minimize errors local to a particular object. The aggregate of these local approximations will typically have greater error than an approximation generated by considering all of the objects at once. HLODs are generated by simplifying separate portions of a scene together and thus are in general higher fidelity approximations for a group of objects than a set of LODs composed of the same number of primitives. The number of primitives that must be rendered is related directly to frame rate, so this quality advantage of HLODs can be exploited in one of two ways: by rendering fewer polygons than a comparable LOD-only system with essentially the same image quality, or by increasing image quality without decreasing the frame rate. SHAPE defers this choice to the user in the form of the two display modes mentioned above:

- **Image Quality Mode:** Render as many polygons as it takes to achieve a specified image quality, no matter how long it takes.
- **Target Frame Rate Mode:** For a given target frame rate, render as many polygons as possible.

The combination of these modes makes it easy for a user to quickly navigate to a point of interest and view that region in high detail.

HLODs have been implemented as part of SHAPE and used to render several massive environments such as a power plant consisting of 13 million triangles and a Double Eagle Tanker consisting of 82 million triangles (See Figure 1).

1.3 Organization

The remainder of the paper is organized as follows. We survey related work on object simplification and interactive display of large models in Section 2. Section 3 presents an overview of our approach. We present the details of our HLOD generation algorithm in Section 4, and the extension to dynamic environments in Section 5. Section 6 presents the display algorithm, and Section 7 discusses our implementation and its performance on various models. We summarize our results and highlight areas for future research in Section 8.

2 RELATED WORK

In this section, we briefly survey previous work related to object simplification and interactive display of large static environments.

2.1 Object Simplification

The problem of generating LODs of a single object has received a great deal of attention in the last few years. Different algorithms can be classified based on a number of properties: whether or not they preserve topology, whether they handle appearance attributes, whether they assume that the input model is a valid mesh, the kind of error metric used for generating the approximation, etc. The underlying decimation operations used in computing LODs include vertex removal, edge collapse, face collapse, vertex clustering and vertex merging [Cohe96, Garl97, Hopp96, Ross93]. Due to space restrictions we do not review all of these algorithms here. Instead, we refer the reader to other existing surveys on the topic [e.g., Heck97, Lueb97, Erik00].

2.2 Interactive Display of Large Static Environments

Many techniques have also been proposed for interactive display of large static environments. Object simplification algorithms can be combined with a suitable scene graph hierarchy [Clar76]. As part of pre-computation, *static LODs* are generated for each object. Then during visualization, the viewpoint is used to cull portions of the hierarchy outside the view frustum and suitable LODs for each visible object are rendered. [Schn94] used the vertex-clustering algorithm of [Ross93] in BRUSH to simplify a number of large CAD models. [Cohe96] used LODs generated by simplification envelopes in a Performer scene graph to render large CAD models. Aliaga et al. [Ali99a] used static LODs generated by GAPS [Erik99] in the MMR system.

Many researchers [e.g., Hopp97, LE97, Xia97] have proposed using *view-dependent simplification* for rendering large individual objects or the whole scene. These algorithms adaptively simplify across the surfaces of objects. They store simplification information in a hierarchical tree of vertices produced by collapse operations and traverse this tree when rendering. Different kinds of selective refinement criteria based on surface orientation and screen-space projection error are used at runtime [Hopp96, Hopp97].

The idea of using hierarchical levels of detail for interactive display of large static and dynamic environments was proposed in [Erik98]. Other similar strategies include that of [Hopp98], which describes a method for building and using hierarchical progressive meshes to display large terrain datasets, and that used by the TerraVista terrain rendering system from TERREX, which is capable of merging buildings with multiple LODs into a terrain tile for coarser representations [TERR].

Other techniques for rendering large environments include *occlusion culling* [Tell91, Gree93, Zhan97, Dura00] that accelerates the rendering of high depth complexity environments. Many researchers have proposed the use of image-based representations [Maci95, Shad96, Scha96, Alia99b] to replace distant geometry with texture-mapped primitives or point samples. [Alia99a] combined occlusion culling, polygonal simplification techniques, and image-based representations for fast display of large static datasets.

2.3 Frame Rate Regulation

Several techniques have been developed to target a frame rate while rendering. [Funk93] described an adaptive display algorithm for interactive frame rates by posing it as a constrained optimization problem. [Rohl94] used a feedback loop to maintain a target frame rate. [Maci95] extended the predictive framework of [Funk93] to a hierarchical version and used texture-mapped primitives as impostors for clusters of objects.

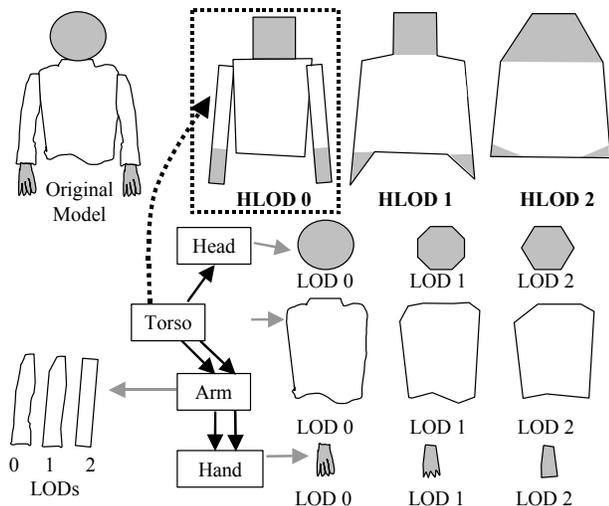


Figure 2: Rendering a model using LODs and HLODs in SHAPE. Gray arrows indicate the LODs of the geometry within each node of the scene graph, while the dotted black arrow indicates the HLODs that represent the entire scene graph. Note that the HLODs of the Arm nodes have been omitted for clarity. SHAPE traverses the scene graph from the root, i.e. the Torso node. If the viewer is far enough away, then it will decide that HLOD 0 is an acceptable approximation of that node and its children. In this case, since this HLOD represents the entire scene graph, SHAPE is able to stop the traversal. HLOD 1 demonstrates the merging of the two arms into the torso, something not possible in a traditional object-based LOD algorithm.

2.4 Dynamic Environments

Not much research has been done on simplification and display of massive dynamic environments. IRIS Performer [Rohl94] uses traditional LOD techniques and is capable of changing the structure of the scene graph to reflect motion. Many researchers have proposed techniques to update bounding volume hierarchies, spatial partitions, and scene graphs when objects undergo motion [Torr90, Chry92, Suda96]. [Dret97] presented an algorithm that provides interactive update rates of global illumination for scenes with moving objects. [Zhan97] presented an occlusion-culling algorithm that is applicable to dynamic scenes. [Jeps95] described an environment for real-time urban simulation that allows dynamic objects to be included in the scene.

3 OVERVIEW

In this section, we give an overview of the HLOD approach and discuss issues related to the design and implementation of SHAPE. We discuss the tradeoffs between view-dependent and static LOD approaches, scene representation, and the use of HLODs for static and dynamic environments.

3.1 Static LODs Versus View-Dependent Simplification

Geometric levels of detail have been used in two forms for fast display of large environments: static LODs and view-dependent simplification. Both of these approaches can be useful in different situations. Although view-dependent algorithms are elegant and provide useful capabilities, they impose significant memory and processor overhead during visualization. Instead of choosing an LOD per visible object, view-dependent algorithms may query every active vertex of every visible object [Hopp97, LE97, Xia97]. Furthermore, object instantiation is expensive since each instance

must contain its own list of active vertices. Another issue is that current high-end graphics machines are able to render *display lists* faster than *immediate mode* primitives [OGL98]. Existing view-dependent algorithms are inherently immediate mode and, therefore, cannot take advantage of display lists.

Given our emphasis on performance, we use static LODs and HLODs, and render them using display lists. We accept their limitations in terms of potential “popping” artifacts as we switch between different simplification levels. In Section 4 we describe how we can combine HLODs with partitioning to effectively arrive at a discrete approximation of view-dependent simplification.

3.2 Scene Representation

We represent the polygonal environment with a traditional *scene graph* [Clar76, Rohl94, Cohe96] extended to include inter-object simplification with HLODs.

Figure 2 shows a simple example of a 2D scene graph. Boxes enclosing text represent nodes while black arrows represent transformations. Gray arrows indicate the polygonal representation that is contained in each node. Note that geometry in the scene graph need not all reside in the leaf nodes. Even though the Torso is the root node and not a leaf, it contains a polygonal representation. The Arm node demonstrates instancing: the single representation of an Arm is instantiated twice in the model, as indicated by the two incoming arcs. The arcs use different transformations to instance the Arms at distinct positions on the body. The same applies to the Hand model, which is also instantiated twice.

3.3 Hierarchical Levels of Detail

Traditional LODs represent the geometry of single nodes in the environment’s scene graph. HLODs represent branches of the scene graph, or the polygons of multiple nodes. A traditional LOD rendering system renders an appropriate level of detail for every object or node in the scene. Since an HLOD of a node in the scene graph is an approximation for the node as well as its descendants, if we render a node’s HLOD in traversing the scene graph, we do not need to visit its descendants (see Figure 2).

3.4 Dynamic Environments

We assume that the polygonal model within each node of the scene graph is static and not deformable. Our approach deals with *rigid body* environments where objects in the scene move due to modifications of the scene graph. Dynamic environments are represented in terms of scene graph operations such as adding nodes and arcs, deleting nodes and arcs, and changing transformations at arcs. This last operation is the most common for scenes composed of moving objects.

Besides model complexity, a dynamic environment is also characterized by the number of dynamic changes in the scene. We highlight three broad categories of dynamic environments:

- **Global Continuous Motion:** In these environments, almost every object is in motion. Some computer games or simulations of an earthquake are examples.
- **Local Continuous Motion:** Some environments exhibit continuous motion, but only in localized regions of the scene. An example is a swinging pendulum within a stationary environment.
- **Infrequent Motion:** These environments are normally static, punctuated by brief periods of dynamic activity. A design and review scenario is an example of this type of scene. A user interacts with objects, and then takes some time to inspect the results before continuing.

Since our environments are composed of rigid bodies, we expect that LODs for a node are precomputed. The algorithm also precomputes all HLODs for intermediate nodes in the scene graph.

In a dynamic environment, the structure of the scene graph changes. As a result, some of the HLODs may no longer be valid and we need to recompute them efficiently. Our algorithm incrementally recomputes these HLODs in a bottom-up manner.

Our approach is currently most effective on scenarios with “infrequent motion”, as the time required to recompute the HLODs is typically greater than that for rendering the scene. As a result, the algorithm performs these computations asynchronously. We also parallelize the recomputation step on shared memory multi-processor graphics systems.

4 HLOD GENERATION

In this section, we present the details of our method for generating HLODs. HLODs are generated by hierarchically grouping the nodes in a scene graph and simplifying them together. The algorithm also partitions spatially large objects in order to gain limited view-dependent rendering capabilities for these objects. It computes and stores LODs and HLODs at each node in the scene graph.

4.1 Basic Algorithm

We will refer to an LOD or HLOD with more polygons as *finer*, and one with fewer polygons as *coarser*. For example, in Figure 2 the shape labeled *HLOD 0* is the finest HLOD, and the one labeled *HLOD 2* is the coarsest.

The HLOD generation algorithm uses a combination of an LOD computation algorithm and hierarchical clustering. The underlying LOD algorithm must be able to perform topological simplification and combine a collection of non-overlapping or disjoint objects. Many known topology simplification algorithms have these capabilities [Ross93, Schr97, Garl97, Popo97]. In SHAPE, we used the GAPS simplification algorithm [Erik99], as it provides a good balance between generality, fidelity and running time.

After LODs are computed for each individual node in the scene graph, our algorithm computes HLODs in a hierarchical, bottom-up manner. The HLODs of a scene graph are computed as follows:

- The HLODs of a leaf node are equivalent to its LODs.
- The finest HLOD of an internal node is computed by simplifying the coarsest LOD of the node itself with the coarsest HLODs of its children.
- The coarser HLODs of an internal node are generated as successive simplifications of its finest HLOD.

As an example, in Figure 2 the HLODs in Torso are formed from *LOD 2* of Torso and *HLOD 2* of Head, Arm (left), and Arm (right). Since Head does not have child nodes, its HLODs are equivalent to its LODs, but the Arm HLODs (not shown) are a merged combination of Arms and Hands.

Other choices for how to compute HLODs are possible. For instance, the initial HLOD could be generated by directly simplifying all the original geometry of the sub-tree that the HLOD represents. In some cases this would lead to higher quality HLODs; however, starting with existing LODs greatly reduces the cost of computation—which is especially critical for dynamic environments—and still yields high quality drastic simplifications. In the end, each HLOD is a complete, simplified representation of the node and all of its descendants.

4.2 Grouping Nodes

Our method requires a hierarchical scene graph representation for each environment. If not provided then SHAPE creates one using *partitioning* (see Section 4.3). Conversely, it is sometimes advantageous to ignore a provided scene graph, since not all scene graphs are optimized for rendering performance. For instance, CAD models commonly group objects by functionality rather than

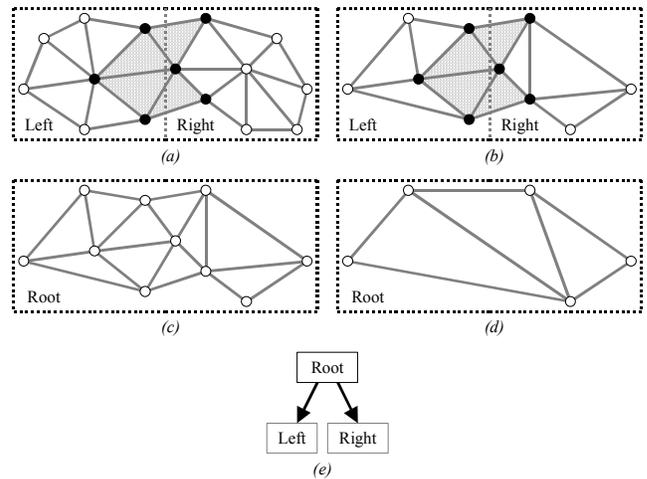


Figure 3: An example of partitioning. (a) The object has been split into two partitions. Gray triangles are restricted since they lie on the border of a partition. The black vertices cannot move during decimation operations. (b) We simplify the partitions independently, noting the restricted triangles. (c) The partitions are grouped hierarchically. There are no more restricted triangles. (d) We can simplify the final partition drastically. (e) The resulting scene graph. The Left and Right nodes contain LODs of the Left and Right partitions in (a) and (b). The Root node contains HLODs of the final partition.

proximity. Flat scene graphs present a similar problem. Both tend to be inefficient for view-frustum culling and HLOD creation. To solve these problems, we use *grouping* to create a more spatially-coherent scene graph. If a node has more than two children, we use an octree spatial partitioning to find nearest neighbors and use these pairings to create a hierarchy with better spatial coherence. This not only aids view-frustum culling, but also results in higher quality HLODs.

4.3 Partitioning Spatially Large Objects

Since we use static LODs, spatially large objects can pose a problem. When the viewer is close to any region of a spatially large object the entire object must be rendered in high detail, even though portions of it may be very far from the viewer. To alleviate this problem, we *partition* the model to gain limited view-dependent rendering capabilities. We simplify each partition while guaranteeing that we do not produce cracks between partitions by imposing restrictions on simplification. Finally, we group the unrestricted polygons of these partitions hierarchically and generate HLOD approximations of them. This partitioning procedure for large geometric objects was initially described in [Erik98]. [Hopp98] describes a similar approach for terrain models. [Avil97] also partitions large models for simplification, by using an OBB bounding volume hierarchy.

Partitioned simplification begins by laying a uniform three-dimensional grid over the large object and determining polygons that are completely within each partition. Polygons that lie on the boundaries of partitions are labeled as *restricted* (Figure 3a). Next, SHAPE simplifies the unrestricted polygons within each partition independently (Figure 3b). We do not allow the simplification algorithm to move any vertices that are incident to a restricted polygon during a decimation operation (e.g. an edge collapse). This restriction guarantees that the algorithm will not generate cracks between any two partitions. We simplify the unrestricted geometry until no more decimation operations can be performed or the algorithm exceeds a deviation distance error threshold associated with LODs and HLODs.

When all partitions have been simplified independently, SHAPE groups them hierarchically. When partitions are grouped, some polygons that were once labeled restricted become unrestricted (Figure 3c). This freeing of polygons enables the simplification algorithm to perform more decimation operations in order to create HLODs for this new hierarchical grouping of partitions (Figure 3d). This process of grouping and simplifying is repeatedly applied until there is only one partition that contains all of the remaining polygons. Since all of the remaining polygons are contained in this partition, there are no more restrictions, and the simplification algorithm can drastically simplify these polygons to any target number.

One can view the HLODs generated through the partitioning process as representing a *discrete approximation* of view-dependent simplification. In this way they occupy a practical niche between completely continuous view-dependent simplification algorithms on the one hand, and a single set of view-independent static LODs on the other. Figures 4 and 5 show the results of partitioning.

Using HLODs allows our algorithm to choose from many discrete samples in order to achieve a balance between rendering speed and image quality. Since each partition is simplified independently of the others, partitions far from the viewer can be rendered in lower detail while those near the viewer are rendered in higher detail (Figures 4, 5). When several partitions in close proximity are very far away from the viewer, they are rendered together using HLODs. An added benefit of partitioning is that it allows us to perform view-frustum culling on parts of the object that lie outside the view frustum. This capability is shown in Figure 5.

5 DYNAMIC ENVIRONMENTS

In this section, we present our approach for dynamic environments. Our algorithm updates HLODs in response to dynamic changes within the environment. It first updates error bounds associated with HLODs affected by object motion. Then, it regroups nodes in the scene graph according to their spatial proximity. Finally, it incrementally updates the scene graph's bounding volume hierarchy. Once the scene graph has been modified, we insert nodes whose HLODs need to be recomputed into a queue. We use one or more simplification processes that run asynchronously from the rendering process to compute the HLODs. If the motion of the objects is relatively small, then it may be acceptable for the rendering process to use previously created HLODs while waiting

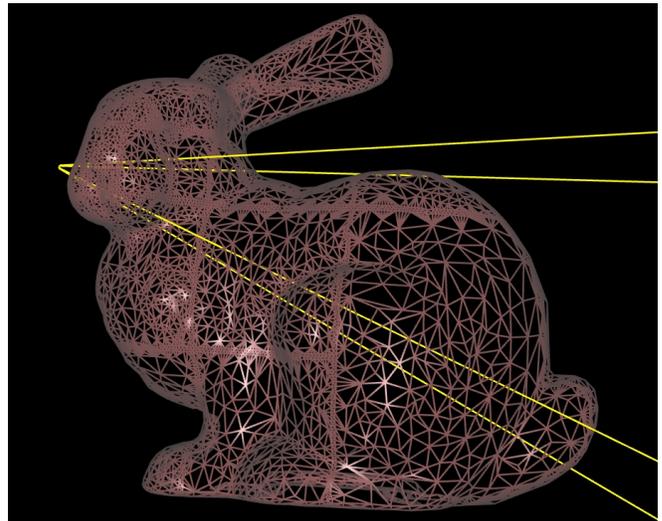


Figure 4: A demonstration of how HLODs generated with partitioning approximate view-dependent simplification. Even with a relatively small number of discrete LODs and HLODs available, the algorithm is able to roughly adapt the tessellation of the model according to the viewpoint. Here the viewpoint is near the nose of the bunny.

for the simplification processes to finish the recomputation.

5.1 Updating the Scene Graph

Our algorithm seeks to minimize the amount of recomputation necessary after any node insertion, deletion, or object motion. Fortunately, while insertions and deletions require more recomputation than motions, they are typically less frequent. We first present the method SHAPE uses to update HLODs after object motion, and then describe how it handles insertion and deletion.

5.1.1 Motion

Our algorithm determines the effect of node motion on the accuracy of HLODs, regroups nodes based on the new positions of the objects, and updates the bounding volume hierarchy of the scene graph.

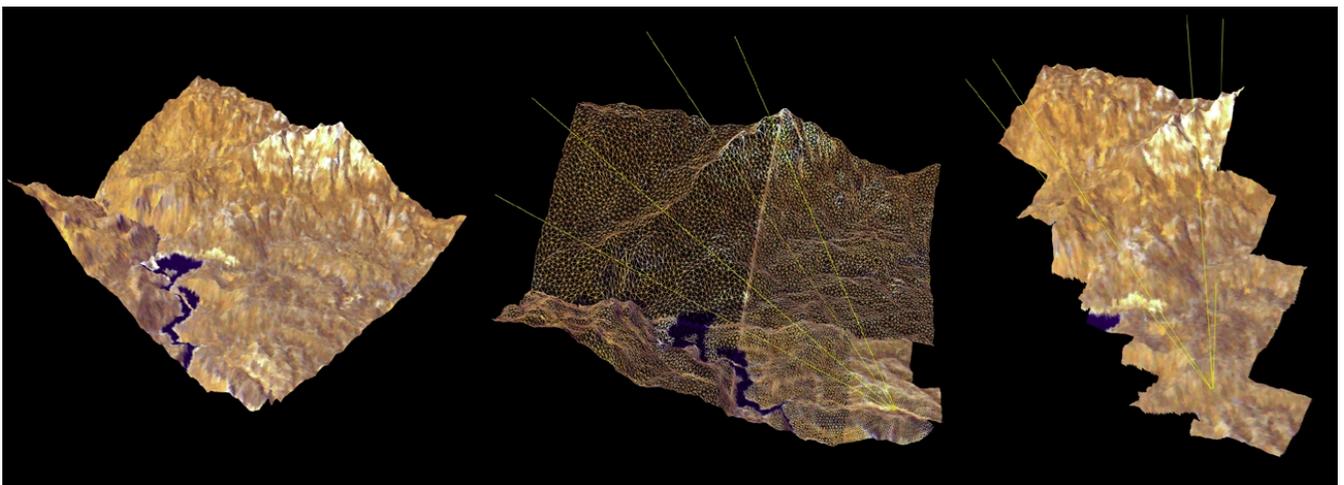


Figure 5: On the left is a terrain model consisting of 162,690 triangles. In the middle, we use partitioning to adaptively simplify the model. This image is shown in wire-frame to illustrate this view-dependent rendering more clearly. Partitions near the origin of the yellow view frustum are in higher detail than partitions further away from the viewer. On the right, we demonstrate that partitions lying outside the view frustum can be culled.

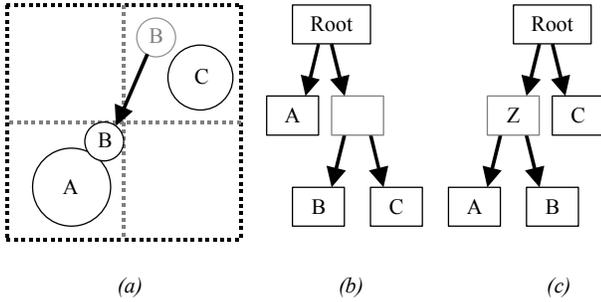


Figure 6: Dynamic movement in a simple scene graph. (a) Node B moves to a position nearby node A. (b) The initial grouping of nodes in the scene graph. (c) The nodes have been regrouped for more efficient view-frustum culling and HLOD creation. Now node Z will contain HLODs that represent nodes A and B.

5.1.1.1 Updating Error Bounds of HLODs

SHAPE measures the distance between the old and new positions of the object and adds this distance to the error bounds of HLODs of the node. Since this movement also affects the error bounds of HLODs of ancestor nodes, we propagate the distance error up the scene graph. Increasing the error bounds of HLODs does not automatically make them useless, since they may still be of acceptable quality if viewed from further away.

5.1.1.2 Regrouping Nodes

As described in Section 4.2, we group nodes hierarchically when a parent node has more than two children. By grouping these children according to their spatial proximity, we make HLOD creation and view-frustum culling more efficient. As objects move in the scene, the relative locations of these nodes change. Thus, a previously efficient grouping of nodes may no longer be efficient. SHAPE updates the scene graph by regrouping these nodes. The degree of movement affects the amount of work we have to perform on the scene graph. If a node moves a small distance, then it is sometimes possible to update the bounding volume hierarchy without changing the structure of the scene graph. If a node moves a larger distance, then the scene graph structure changes. An example of this regrouping process is shown in Figure 6.

Movement of objects causes the approximation quality of HLODs to decrease. However, when nodes are regrouped, some HLODs become *invalid* altogether. Since the descendants of the node in the regrouped scene graph may have changed, its HLODs are no longer valid approximations. We add the nodes containing invalidated HLODs into the simplification queue. They remain invalid until a simplification process can recompute the HLODs. During this time, the display algorithm renders non-hierarchical LODs for those nodes and their children.

5.1.2 Insertion and Deletion

Insertion and deletion of nodes in the scene graph cannot be handled as efficiently as object motion. If a node is inserted or deleted, then we invalidate all the HLODs of its ancestors. Furthermore, for insertions we must also regroup nodes based on positions of objects in the new scene graph. Both insertion and deletion can cause the bounding volume hierarchy to change as well. Nodes with invalid HLODs are inserted into the simplification queue so that their HLODs will be recomputed.

5.2 Asynchronous Simplification

Movement of nodes in the scene causes many HLODs to become invalid, which must then be recomputed. The required polygonal simplification can take much longer than it takes to render the

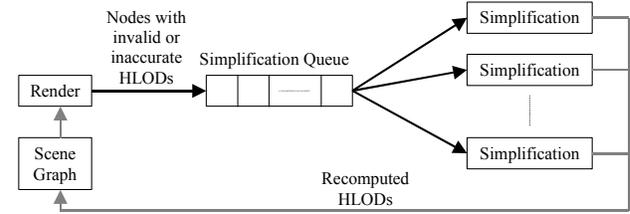


Figure 7: Diagram showing how the different processes in our dynamic algorithm interact.

scene. To prevent rendering delays, SHAPE can run the simplification process asynchronously in a background thread. SHAPE can use multiple asynchronous simplification processes on a multiprocessor machine to increase the rate of simplification.

The job of a simplification process is to dequeue a node and recompute a set of HLODs for that node. We use the topology simplification algorithm at run-time to update these HLODs as we used in the preprocessing. Once it finishes creating a set of HLODs for the node, it copies them into the scene graph. The interaction between the rendering process and simplification processes is shown in Figure 7.

We prevent the multiple processes from corrupting the scene graph data by using a combination of three semaphores. One semaphore protects the scene graph; another protects HLODs at nodes, and the final semaphore controls access to the simplification queue. The simplification processes are designed to lock the scene graph and HLOD semaphores for very brief periods of time, thereby giving priority to the rendering process.

Our algorithm works well for “infrequent motion” environments, such as design and review scenarios. Objects seldom move in such scenes and therefore the simplification processes are usually able to update the HLODs in a few seconds. Often, a user will only manipulate objects that are near their current viewing position. Since the rendering algorithm only uses HLODs for coarse approximations, the user must move some distance away from these objects before the rendering algorithm will require the new HLODs. In most situations the simplification process will be able to recompute the HLODs within this time duration.

5.3 Analysis

In this section, we derive a rough measure of how much dynamism our algorithm can handle at interactive rates. The performance of the algorithm is determined by the complexity of the scene, the height of the scene graph, the choice of simplification algorithm, and its performance. We make a few assumptions and present a model for our analysis:

- **Scene Graph:** Each parent node in the scene graph has c children. The height of the scene graph is h and all the objects are at leaf nodes. Therefore, for every object that moves, we need to recalculate $h - 1$ sets of HLODs in the scene graph. Finally, the polygonal representation of a node consists of v vertices.
- **Operation Cost:** The cost of the recomputation step is mostly dominated by decimation operations performed by the simplification algorithm. As a result, we ignore the cost of updating error bounds of HLODs, regrouping nodes, and updating the bounding-volume hierarchy.
- **Simplification Algorithm:** We use vertex merges as the decimation operation and are able to perform m such operations per second. The simplification algorithm simplifies the polygons of a node until v/r vertices remain, for LOD as well as HLOD computations.
- **Interaction Mode:** The algorithm will not need to render the newly created HLODs before s seconds have passed.

Let us assume that $c \neq r$ and we are using a single simplification process. For each moving object, the algorithm needs to perform

$$M = \left(cv/r \right) \left[\frac{\left(c/r \right)^{h-2} - 1}{\left(c/r \right) - 1} \right] \left(1 - 1/r \right) + c^{h-1} \left(v/r^{h-1} \right)$$

vertex merge operations. This equation is derived using a geometric series involving the number of vertices merged to create each HLOD and the number of vertices propagated up the scene graph. If $M \leq ms$ then our algorithm is capable of recomputing the affected HLODs before they are needed for rendering.

For a more concrete example, suppose we are attempting to recalculate HLODs for a Torpedo Room model (shown in Figure 10) consisting of 356 objects, 883,537 polygons, and 545,949 vertices. The scene graph has an average of 3.1 children per node, its HLOD reduction ratio is approximately 8.3, and its height is 7. Assume HLODs must be recalculated in 10 seconds and that our simplification algorithm performs, on average, 650 vertex merge operations per second on our graphics workstation. Then the constants for the Torpedo Room model are: $m \approx 650$, $s = 10$, $c \approx 3.1$, $h = 7$, $v = 545949 / 356 \approx 1534$, and $r \approx 8.3$. Substituting these constants into the above expressions yields $M \approx 803$. Since $803 = M \leq ms = 6500$, our algorithm should be able to handle this scenario within the given time constraint.

Suppose we move n objects in the scene. As long as $nM \leq ms$ our algorithm should still be able to recalculate the HLODs within the time constraint. Thus, our analysis suggests that our current implementation should be able to update the Torpedo Room HLODs within 10 seconds after 8 objects have moved ($8 \times 803 = 6424 \leq 6500$). In practice, our algorithm took approximately 13 seconds to update the HLODs of the scene using one simplification process after moving 8 parts of the Torpedo Room model. Note, that our estimate relied upon approximate values for m , c , v , and r and ignores the costs of updating the scene graph and accessing the semaphores. More details on the analysis are given in [Erik00].

6 RENDERING MODES

In this section we discuss our algorithm for rendering a scene graph containing LODs and HLODs. We begin with a description of how HLODs can be used to cull out entire portions of the scene graph, and follow with a detailed explanation of our image-quality and frame-rate targeting modes.

6.1 Simplification Culling

We assume the polygonal simplification algorithm we use to generate LODs and HLODs is capable of producing a distance error bound for these approximations. This error measures the quality of an approximation based on deviation from the original object, and is projected onto the view plane to determine the screen-space error of the LOD or HLOD.

With a traditional scene graph containing only LODs, rendering a node involves determining which LOD to use for the geometry contained in that node. If the screen-space error is less than a user specified error tolerance, that LOD can adequately represent the geometry at the node. The lowest LOD that passes the screen-space error criteria is rendered. The traversal of the scene graph continues recursively for each of the node's children.

Adding HLODs to each node changes the traversal of the scene graph. When the traversal reaches a node, it first determines whether an HLOD can be rendered. Like LODs, each HLOD has an associated maximum distance error. If an HLOD's projected screen-space error is less than the current error tolerance, then it can be rendered. The algorithm renders the lowest HLOD that meets the screen-space error criteria and does not traverse any of the node's children. This is, in effect, *simplification culling* since the scene graph rooted at the node is culled away by substituting a

simpler representation for it. If no HLOD meets the error tolerance, we select an LOD to represent the node and then recursively traverse each of its children.

6.2 Image Quality Mode

In this mode, the user is allowed to specify a desired image quality in terms of maximum screen space deviation. While rendering, the projected screen-space error associated with each LOD and HLOD is used to determine an acceptable representation given the image quality constraint. This much is typical of most LOD rendering algorithms. The main difference is that if an HLOD has acceptable image quality then the entire branch of the scene graph it represents need not be traversed.

6.3 Target Frame Rate Mode

Target frame-rate systems have the goal of rendering the best image possible within a user-specified frame-rate constraint. [Funk93, Maci95] show that targeting a frame rate using prediction techniques is a variant of the Knapsack problem. Both of these algorithms rely on pre-computed estimates of system performance that may not be accurate during run-time. [Roh194, Muel95] describe reactive *feedback loops* to target a frame rate, meaning that the time it took to draw previous frames is used to calculate the image quality for the next. Feedback loops suffer from the potential problems of *oscillation* and *hysteresis*, against which countermeasures must be taken. HLODs enable SHAPE to achieve a target frame rate with a combination of predictive and reactive techniques.

Like a predictive scheme, SHAPE uses a target number of faces. This number is a best guess prediction of how many polygons the system can render given the user-specified frame-rate constraint. However, this number is updated reactively: if we cannot render the number of faces within the frame-rate constraint, the target number of faces is decreased for the next frame. The main difference between this method and previous feedback methods is that previous methods used frame time to adjust LOD quality settings whereas we use frame time to adjust a polygon budget. Having an HLOD based scene graph gives us an easy way to meet any such polygon budget using a greedy scheme without having to resort to omitting portions of the scene. Using only quality settings can lead to overcompensation and therefore oscillation in frame rate.

We search the scene graph to determine the faces that will be rendered. Our greedy method refines the node with the greatest projected pixel-error at each step. We repeat this procedure until any refinement would cause the total number of faces to exceed the target number. Specifically, the algorithm starts with the coarsest HLOD of the root node of the entire scene graph. It attempts to refine the node with the most screen-space error by replacing it with its children. If replacing a node would cause our algorithm to render more polygons than the target number of faces, then this action is not allowed. We refine the nodes until no more nodes can be replaced.

In the case of dynamic objects, some HLODs can become inaccurate. However, these approximations can still be used to target a frame rate. Large movement causes some HLODs to become invalid. We ignore invalid HLODs during our traversal of the scene graph. In such cases, SHAPE can either render LODs of nodes or stop the traversal and render an incomplete approximation of the scene.

7 IMPLEMENTATION AND RESULTS

We have tested our approach on different models including two large CAD environments. The first is a 13 million-polygon power plant, and the second is an 82 million-polygon Double Eagle Tanker ship. We also show a model of a Ford Bronco to demonstrate the effectiveness of HLODs. The Bunny and Sierra

Terrain models show the benefits of partitioning (see Section 4.3). Neither the Bunny nor the terrain model came with a scene graph hierarchy.

SHAPE uses the GAPS simplification algorithm [Erik99] to compute LODs as well as HLODs. It is based on the quadric error metric proposed by [Garl97] and works well on large models. Other candidates are algorithms based on vertex clustering [Ross93] and progressive simplicial complexes [Popo97]. In our experience, GAPS provides a good balance between quality of approximation and execution speed. On average, GAPS performs 650 vertex merge operations per second on an SGI Reality Monster with a 300 MHz R12000 processor and 16GB of main memory. We implemented our system using C++, GLUT, GLUI, and OpenGL. The code is currently portable across PC and SGI platforms.

7.1 Preprocessing Time

Table 1 shows the amount of time it took to preprocess selected test models. The number of objects, or leaf nodes, in the scene graph is shown in the table as well as the number of triangles that make up the model. Note that there is a certain amount of overhead involved in grouping objects and simplifying them to create HLODs. The amount of overhead depends on the complexity of the scene graph and the number of polygons combined to compute each HLOD. Note also that HLOD creation takes less time on the Sierra Terrain model than LOD creation. The reason for this difference in performance is that by partitioning the terrain model, the simplification algorithm is initially able to work on local portions of the terrain model independently. Simplifying subsets of polygons is faster than simplifying all the polygons at once due to the performance behavior of the algorithm we use.

7.2 Targeting Frame Rate

Figure 8 shows the effectiveness of the target frame-rate mode on two test environments with the range of acceptable frame-rates set to between 17 and 23 frames per second. Note that the algorithm knew nothing about the specifications of the machine on which it was running. For a majority of the time, our method rendered each model within the acceptable range. However, for each viewing path of each model there are sharp transitions in polygonal geometry, causing low or high spikes in these regions. Our algorithm is able to quickly react to these changes and bring the frame rate within acceptable bounds. Sometimes we render much faster than the target frame rate. In these cases, we are rendering the original polygons of the model, but there are not enough of them in the view frustum to keep a constant frame rate. This behavior is not troubling since the frame rate could easily be clamped if so desired.

7.3 Memory Usage

For static scenes, we pre-compute a series of LODs or HLODs that consist of half the number of polygons of the previous LOD or HLOD, respectively. By doing so, we limit the memory requirements of our algorithm to at most two times that of the original environment. All the LODs and HLODs are represented using display lists.

For dynamic environments, we re-compute the HLODs at runtime by pooling polygonal geometry from different nodes. It is not possible to access this data from an OpenGL display list, and therefore, we need to store a separate copy of the polygonal geometric representation. Along with this extra polygonal geometry, we also store all the data structures used by the simplification algorithm. This includes the error quadrics and mesh

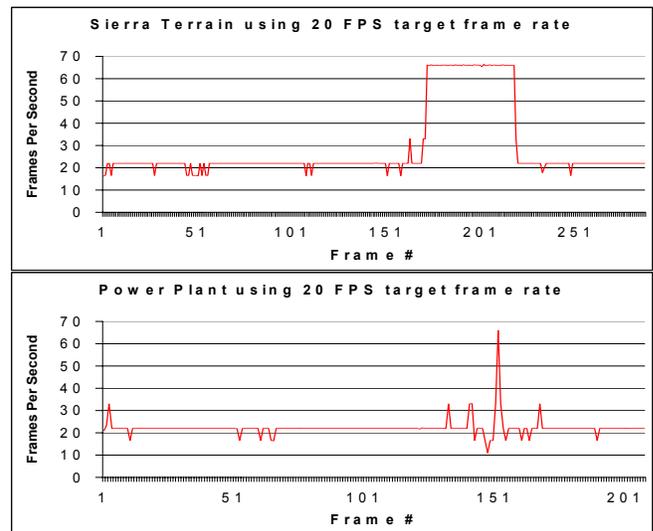


Figure 8: Frame rates obtained using the target frame-rate mode to view the power plant model and the Sierra Terrain dataset. The target frame rate was 20 frames/sec.

connectivity used by the GAPS algorithm [Erik99]. This data is commonly 3–4 times the size of the original model. Altogether this means that dynamic scenes may require as much as six times the memory of the original polygonal geometry. The choice of a different simplification algorithm, and the resulting data structures, will affect the memory requirements of our system.

7.4 Visual Comparison

The main benefit of using HLODs is that they provide higher quality drastic approximations for groups of objects. Using only LODs, groups of objects tend to break apart or disintegrate at coarse approximations. However, by using a combination of LODs and HLODs, we can produce more solid-looking drastic approximations. Because HLODs promote the merging of objects in close proximity, they are most effective on scenes where objects are closely spaced. Most CAD environments fall in this category.

We show the visual difference between LODs and HLODs for drastic approximations of the Bronco (Plate 1), power plant (Plate 3), Double Eagle (Plate 4), and Torpedo Room (Figure 10) environments. Note how the solid shape of these scenes tends to suffer when using only LODs. By pooling the geometry of several objects into HLODs, we are able to better preserve the general shape and surface area of these environments further into the simplification process.

7.5 Asynchronous Simplification

We tested the performance of the *parallel* algorithm to determine the amount of motion it can handle efficiently. To perform the tests, we created a simple scene consisting of a root node that has multiple instances of a cube object as its children (as shown in the video). We tested multiple scenes using different numbers of simplification processors. The cubes were initially arranged in a 3D grid and then every cube was moved a random distance (less than the side of the whole grid) from its original position. After the movement was completed, the simplification processors recomputed HLODs for the scene. We ran these tests on an SGI Reality Monster with 31, 300 MHz R12000 processors and 16GB of main memory.

A graph of these timing tests is shown in Figure 9. This data demonstrates that as the scenes consist of more moving objects, the benefit of using more simplification processors increases. Ideally, we would expect a linear speedup as we use more processors. However, test cases show that SHAPE achieves a sub-linear speedup. We conjecture that overhead costs such as contention for data in the scene graph, plus the relatively small size of the scenes being tested, are to blame. As the scene grows larger, we expect that the speedup from using multiple simplification processors will approach linearity.

These results show that using multiple processors for simplification is most beneficial when a large number of objects are moving in the scene. For many cases, however, the recomputation of HLODs of the cube grid scenes does not occur in real-time. For a scene of moderate dynamic complexity, such as the environment with 512 moving cubes, it takes more than half a minute to update the affected HLODs even using 31 processors. This performance suggests that our dynamic system is best used on environments with “infrequent motion” such as design and review scenarios.

To simulate a design and review scenario on real-world environments, we allow the user to select and move any objects in view. During visualization sessions of the various models, we changed the locations of a few of the objects (see Plate 2 and the video). For each test run, we used 4 simplification processes. The execution speed of HLOD recomputation for these examples is shown in Table 2.

7.6 Display Lists

We found the ability to use display lists was an advantage of SHAPE in many cases. However, when the number of nodes in the scene graph becomes very large, the amount of memory required for display lists may exceed the amount of display list cache available. At that point the display lists may no longer improve the rendering speed. In these cases, the rendering program may benefit from managing the display cache directly.

8 CONCLUSION AND FUTURE WORK

We have presented a simple and effective system for rendering very large static and dynamic environments using only geometric LODs and HLODs. The HLODs serve as higher fidelity drastic approximations for entire branches of the scene graph. The process of generating HLODs is completely automatic, which is important when working with massive models. For dynamic environments, we are able to incrementally recompute HLODs on the fly, an approach which we have shown works well on scenes exhibiting “infrequent motion.” Our system, SHAPE, gives empirical evidence that the HLOD approach is viable for interactively displaying massive geometric environments.

Our approach for dynamic scenes is a first step towards handling arbitrary object motion interactively in massive model walkthroughs. It makes no assumptions about object motion, and

Scene	Objects	Triangles	Preprocessing times	
			LOD only	LOD & HLOD
Bronco	466	74,308	31 secs.	43 secs.
Sierra Terrain	1	162,690	2 mins. 21 secs.	1 min. 57 secs.
Power Plant	1,179	12,731,154	4 hrs.	4 hrs. 12 mins.
Double Eagle	126,630	82,361,612	12 hrs. 22 mins.	12 hrs. 57 mins.

Table 1: Preprocessing times for the creation of LODs versus the creation of LODs and HLODs for several polygonal environments. We performed these tests on an SGI Reality Monster with a 300 MHz R12000 processor and 16GB of main memory.

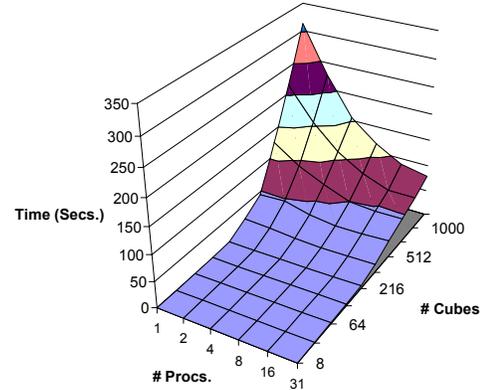


Figure 9: This graph shows the time it takes to recalculate HLODs of a scene consisting of a specific number of cubes using a specific number of processors. The number of processors and number of cubes axes are log plots.

is effective for a class of dynamic scenes. Furthermore, it can automatically make use of multiple processors on a graphics system.

Broadly speaking, a massive model display system needs to address three issues:

1. Using approximations for polygons that are too small or cover too few pixels on the screen,
2. Not rendering polygons that are occluded
3. Ensuring that the resulting polygons are in main memory.

SHAPE handles only the first of these issues, and appears to work well on large environments. In future work we hope to address the remaining two. We are very interested in combining the results of SHAPE with occlusion culling methods to address the second. One possibility is to use HLODs as hierarchical occluders. Finally, we would like to combine these with database and memory management techniques. This will also improve the load time for very large models. In terms of model simplification, we would like to explore other methods of parallelization of HLOD recomputation and explore avenues for reduced memory and out-of-core runtime simplification for dynamic scenes.

9 ACKNOWLEDGEMENTS

The Power Plant environment is courtesy of James Close and Combustion Engineering, Inc. The Sierra terrain model is courtesy of Herman Towles and Sun Microsystems. Viewpoint and Division, Inc. allowed us to use the Ford Bronco dataset. The Double Eagle model is courtesy of Rob Lisle, Bryan Marz, and Jack Kanakaris at NNS. We are also grateful to the members of UNC Walkthrough Group for many useful discussions and support.

Our work was supported in part by an ARO Contract DAAD19-99-1-0162, NSF award 9876914, DOE ASCI grant, NIH National Center for Research Resources Award 2P41RR02170-13 on Interactive Graphics for Molecular Studies and Microscopy, ONR Young Investigator Award, and NCAA Graduate, NSF Graduate, and Intel Fellowships.

Scene	Objects	Triangles	Recalculation Time
Bronco	466	74,308	3 secs.
Cassini	127	349,281	6 secs.
Torp. Room	356	883,537	9 secs.
Power Plant	1,179	12,731,154	43 secs.

Table 2: HLOD recalculation speed for simulated design and review scenarios.

10 REFERENCES

- [Alia99a] Aliaga, D., Cohen, J., Wilson, A., Baker, E., Zhang, H., Erikson, C., Hoff, K., Hudson, T., Stuerzlinger, W., Bastos, R., Whittom, M., Brooks, F., and Manocha, D. "MMR: An Interactive Massive Model Rendering System Using Geometric and Image-Based Acceleration," *Symposium on Interactive 3D Graphics '99 Proceedings*, 199-206, 237, 1999.
- [Alia99b] Aliaga, D. and Lastra, A., "Automatic Image Placement to Provide a Guaranteed Frame Rate," *Computer Graphics (SIGGRAPH 99 Proceedings)*, 307-316, 1999.
- [Avil97] Avila, L.S., and Schroeder, W., "Interactive Visualization of Aircraft and Power Generation Engines," *IEEE Visualization '97 Proceedings*, 483-486, 1997.
- [Chry92] Chrysanthou, Y., and Slater, M., "Computing Dynamic Changes to BSP Trees," *Computer Graphics Forum (Eurographics '92 Proceedings)*, 321-332, 1992.
- [Clar76] Clark, J., "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, 547-554, 1976.
- [Cohe96] Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W., "Simplification Envelopes," *Computer Graphics (SIGGRAPH 96 Proceedings)*, 119-128, 1996.
- [Cohe98] Cohen, J., Olano, M. and Manocha, D., "Appearance Preserving Simplification", *Computer Graphics (SIGGRAPH 98 Proceedings)*, pp. 115-122, 1998.
- [Dret97] Drettakis, G. and Sillion, F., "Interactive Update of Global Illumination Using a Line-Space Hierarchy", *Computer Graphics (SIGGRAPH 97 Proceedings)*, pp. 57-64, 1997.
- [Dura00] Durand, F., Drettakis, G., Thollot, J., Puech, C., "Conservative Visibility Preprocessing using Extended Projections," *Computer Graphics (SIGGRAPH 00 Proceedings)*, 239-248, 2000.
- [ElSa97] El-Sana, J., and Varshney, A., "Controlled Simplification of Genus for Polygonal Models," *IEEE Visualization '97 Proceedings*, 403-410, 1997.
- [Erik98] Erikson, C. and Manocha, D., "Simplification Culling of Large Static and Dynamic Environments", Technical Report TR98-009, Univeristy of North Carolina at Chapel Hill, 1998.
- [Erik99] Erikson, C., and Manocha, D. "GAPS: General and Automatic Polygonal Simplification," *Symposium on Interactive 3D Graphics '99 Proceedings*, 79-88, 225, 1999.
- [Erik00] Erikson, C., "Hierarchical Levels of Detail to Accelerate the Rendering of Large Static and Dynamic Polygonal Environments," PhD Dissertation, Department of Computer Science, *UNC Chapel Hill*, 2000.
- [Funk93] Funkhouser, T., and Séquin, C., "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," *Computer Graphics (SIGGRAPH 93 Proceedings)*, 247-254, 1993.
- [Garl97] Garland, M., and Heckbert, P., "Surface Simplification Using Quadric Error Metrics," *Computer Graphics (SIGGRAPH 97 Proceedings)*, 209-216, 1997.
- [Gree93] Greene, N., Kass, M., and Miller, G., "Hierarchical Z-Buffer Visibility," *Computer Graphics (SIGGRAPH 93 Proceedings)*, 231-238, 1993.
- [Heck97] Heckbert, P., and Garland, M., "Survey of Polygonal Surface Simplification Algorithms," *Technical Report, Department of Computer Science, Carnegie Mellon University* 1997.
- [Hopp96] Hoppe, H., "Progressive Meshes," *Computer Graphics (SIGGRAPH 96 Proceedings)*, 99-108, 1996.
- [Hopp97] Hoppe, H., "View-Dependent Refinement of Progressive Meshes," *Computer Graphics (SIGGRAPH 97 Proceedings)*, 189-198, 1997.
- [Hopp98] Hoppe, H., "Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering," *Proceedings IEEE Visualization '98*, 35-42, 1998.
- [Jeps95] Jepson, W., Liggett, R and Friedman, S., "An Environment for Real-time Urban Simulation", ACM Symposium on Interactive 3D Graphics, pp. 165-166, 1995.
- [Lind98] Lindstrom, P., and Turk, G., "Fast and Memory Efficient Polygonal Simplification," *IEEE Visualization '98 Proceedings*, 279-286, 1998.
- [Lind00] Lindstrom, P., "Out-of-core Simplification of Large Polygonal Models," *Computer Graphics (SIGGRAPH 00 Proceedings)*, 259-262, July 2000.
- [Low97] Low, K., and Tan, T., "Model Simplification Using Vertex-Clustering," *Symposium on Interactive 3D Graphics '97 Proceedings*, 75-82, 1997.
- [Lueb97] Luebke, D., "A Survey of Polygonal Simplification Algorithms," *UNC Chapel Hill Computer Science Technical Report TR97-045*, 1997.
- [LE97] Luebke, D., and Erikson, C., "View-Dependent Simplification of Arbitrary Polygonal Environments," *Computer Graphics (SIGGRAPH 97 Proceedings)*, 199-208, 1997.
- [Maci95] Maciel, P., and Shirley, P., "Visual Navigation of Large Environments Using Textured Clusters," *Symposium on Interactive 3D Graphics '95 Proceedings*, 95-102, 1995.
- [Muel95] Mueller, C., "Architectures of Image Generators for Flight Simulators," *UNC Chapel Hill Computer Science Technical Report TR95-015*, 1995.
- [OGL98] *Advanced Graphics Programming Techniques Using OpenGL*, ACM SIGGRAPH Course Notes, Course #17, 1998.
- [Popo97] Popovic, J., and Hoppe, H., "Progressive Simplicial Complexes," *Computer Graphics (SIGGRAPH 97 Proceedings)*, 217-224, 1997.
- [Rohf94] Rohlf, J., and Helman, J., "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics," *Computer Graphics (SIGGRAPH 94 Proceedings)*, 381-394, 1994.
- [Ronf96] Ronfard, R., and Rossignac, J., "Full-range Approximation of Triangulated Polyhedra," *Computer Graphics Forum (Eurographics '96 Proceedings)*, 67-76, 1996.
- [Ross93] Rossignac, J., and Borrel, P., "Multi-Resolution 3D Approximations for Rendering Complex Scenes," *Geometric Modeling in Computer Graphics*, 455-465, 1993.

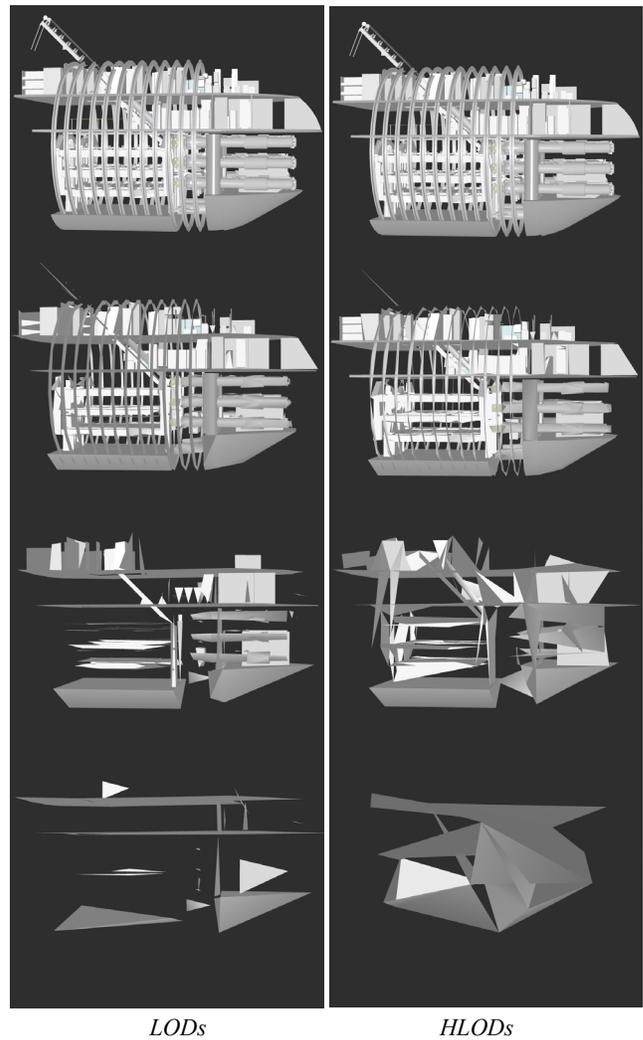
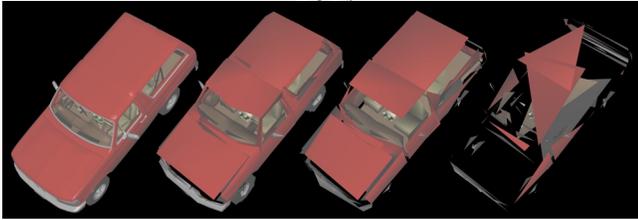


Figure 10: Comparison of LODs and HLODs for the Torpedo Room model. (left) LODs consisting of 883,537 faces, 6,160 faces, 822 faces, and 95 faces. (right) HLODs consisting of 883,537 faces, 6,160 faces, 822 faces, and 95 faces.

- [Scha96] Schaufler, G., and Stuerzlinger, W. "Three Dimensional Image Cache for Virtual Reality," *Computer Graphics Forum (Eurographics '96 Proceedings)*, 227-235, 1996.
- [Schn94] Schneider, B., Borrel, P., Menon, J., Mittleman, J., and Rossignac, J., "Brush as a Walkthrough System for Architectural Models," *Fifth Eurographics Workshop on Rendering*, 389-399, 1994.
- [Schr97] Schroeder, W., "A Topology Modifying Progressive Decimation Algorithm," *IEEE Visualization '97 Proceedings*, 205-212, 1997.
- [Shad96] Shade, J., Lischinski, D., Salesin, D., DeRose, T., and Snyder, J., "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments," *Computer Graphics (SIGGRAPH 96 Proceedings)*, 75-82, 1996.
- [Suda96] Sudarsky, O., and Gotsman, C., "Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality," *Computer Graphics Forum (Eurographics '96 Proceedings)*, 249-258, 1996.
- [Tel91] Teller, S., and Séquin, C., "Visibility Preprocessing for Interactive Walkthroughs," *Computer Graphics (SIGGRAPH 91 Proceedings)*, 61-69, 1991.
- [TERR] Terrain Experts, Inc. <http://www.terrex.com>.
- [Torr90] Torres, E., "Optimization of the Binary Space Partitioning Algorithm (BSP) for the Visualization of Dynamic Scenes," *Computer Graphics Forum (Eurographics '90 Proceedings)*, 507-518, 1990.
- [Xia97] Xia, J., El-Sana, J., Varshney, A., "Adaptive Real-Time Level-of-Detail-Based-Rendering for Polygonal Models," *IEEE Transactions on Visualization and Computer Graphics*, 171-183, 1997.
- [Zhan97] Zhang, H., Manocha, D., Hudson, T. and Hoff, K., "Visibility Culling using Hierarchical Occlusion Maps," *Computer Graphics (SIGGRAPH 97 Proceedings)*, 77-88, 1997.

LODs



HLODs

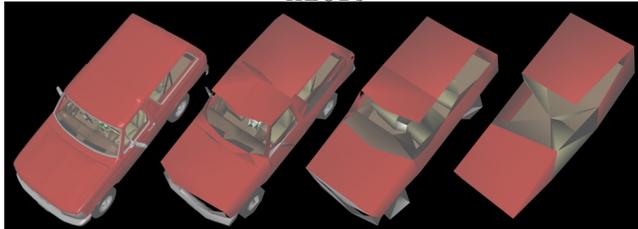


Plate 1: (top) LODs of the Ford Bronco. They consist of 74,308 faces (the original model), 1,357 faces, 341 faces, and 108 faces. (bottom) HLODs consisting of 74,308 faces, 1,357 faces, 338 faces, and 80 faces.

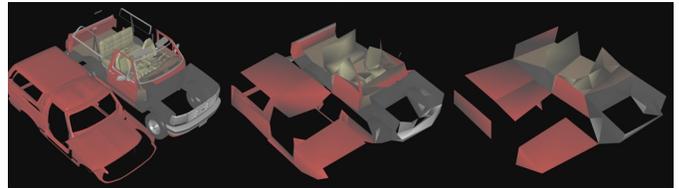


Plate 2: Dynamic modification of the Ford Bronco from Plate 1. We have moved the top of the Bronco in order to look into its interior. The two HLODs consist of 552 and 136 faces respectively and took 3 seconds to recompute using 4 simplification processes on an SGI Reality Monster with 300 MHz R12000 processors and 16GB of main memory.

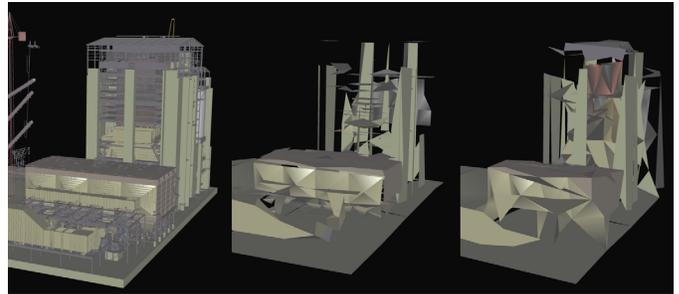
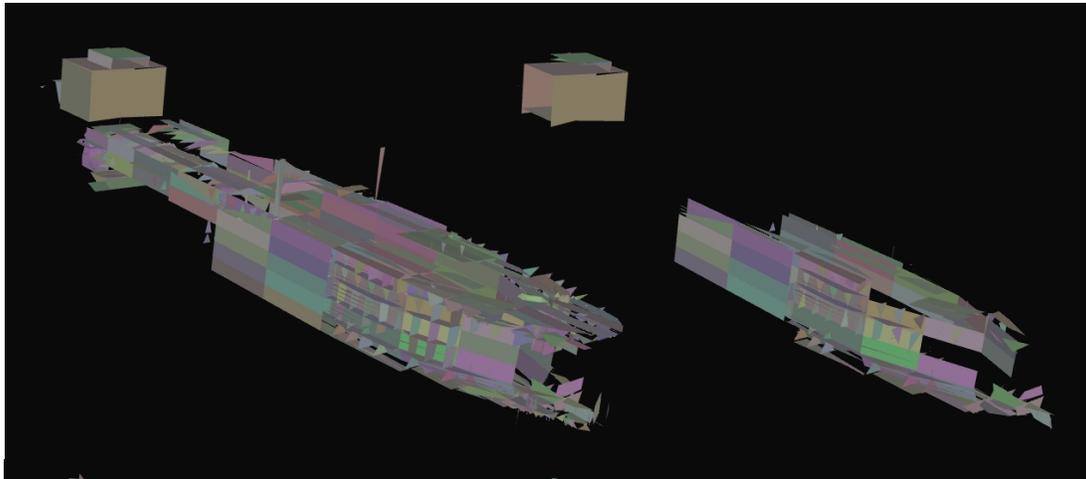


Plate 3: (left) The original power plant model consisting of 12,731,154 faces. (middle) LODs of the power plant consisting of 2,515 faces. (right) HLODs consisting of 2,379 faces.

LODs



HLODs

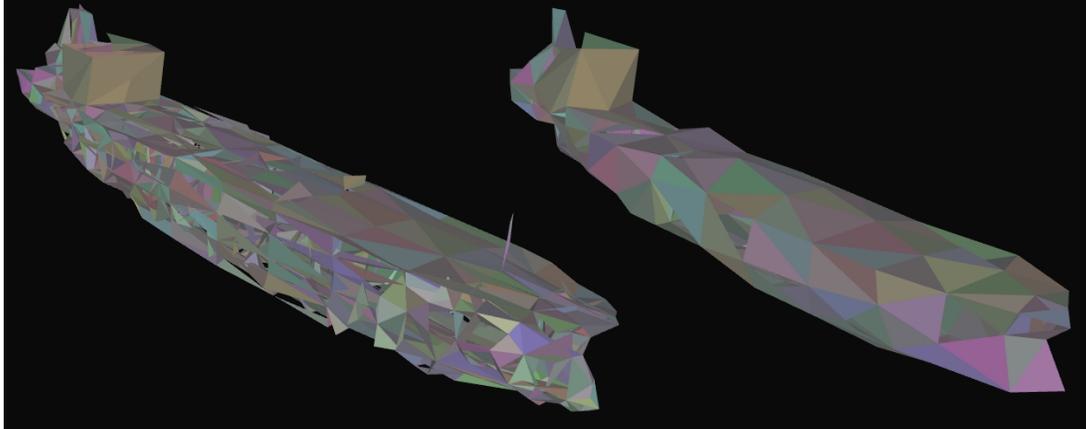


Plate 4: Comparison of LODs and HLODs of the Double Eagle Tanker. The original model is shown in Figure 1 with an alternate color scheme. (above) LODs consisting of 7,887 and 1,922 faces. (below) HLODs consisting of 7,710 and 1,914 faces respectively.