

# VISIBILITY-BASED FOREST WALK-THROUGH USING INERTIAL LEVEL OF DETAIL MODELS

**Paulius Micikevicius**

School of Computing  
Armstrong Atlantic State University  
Savannah, Georgia  
paulius@cs.armstrong.edu

**Charles E. Hughes**

School of Electrical Engineering and Computer Science  
University of Central Florida  
Orlando, Florida  
ceh@cs.ucf.edu

Modern simulation and training applications often require visualization of large scenes with many complex objects. However, this places prohibitive requirements on graphics hardware. For example, while methods for rendering near photo-realistic vegetation scenes have been described in the literature, they require minutes of computation. In this paper we present scene and level of detail management (LOD) techniques for achieving interactive frame rates for a forest walk-through. The framework selects the LOD based on object visibility, in addition to projected size. Moreover, visibility is calculated at run-time so we can support dynamic scene modification, such as interactively altered trees due to wind, explosions or fire. We also introduce an inertial level of detail model to minimize popping artifacts – rather than instantly switching the LODs, discrete LODs are smoothly blended over a number of frames. The described techniques can be readily adapted for scenes other than forests, providing support for simulations involving both static and dynamic synthetic environments, whether natural (e.g., dense vegetation) or man-made (e.g., urban landscapes).

**Keywords:** Simulation/training walk-through, level-of-detail, synthetic natural environments, visibility, visualization, graphics processing unit (GPU).

## 1. 1. Introduction

Many defense simulation and training applications require interactive walk-through visualizations of large and complex scenes (natural and man-made). For example, driving simulators often traverse large terrain databases, rendering the driver's surroundings in high detail; search and rescue training simulators require enough detail and unique environment features to allow navigation based on landmarks. Rendering such complex terrains with the details required for walk-throughs generally exceeds the capabilities of even the fastest graphics hardware.

In order to be useful for the kinds of simulation and training applications we envision, the virtual forest must satisfy the following constraints:

- Each tree must be unique.

Instantiating affinely transformed copies of the same tree is unacceptable in applications requiring feature-based navigation.

- Each tree must be persistent.

The exact same tree model must be rendered, given the same viewpoint position, direction and visibility. Thus, tree structures cannot be generated randomly when their positions fall within the viewing frustum and then discarded when they are no longer visible.

Though several systems geared for military simulation incorporate forest visualization, none of these meet our criteria for uniqueness and persistence of each tree, and for unconstrained real-time walk-throughs of large forested areas.

Object Raku Technology's Sextant software [1] offers three levels of forest abstraction, in increasing geometric complexity: tree-line (forest silhouette), tree-line with canopy, and groups of tree models. Since fast generation of urban environments from intelligence data is the focus of Sextant software, it is not designed to visualize dismounted operations inside large forest scenes. America's Army videogame [2] uses handcrafted terrain databases, an approach not practical for geographically accurate military simulations due to the long development cycle. GENETICS (Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations), a PhD project [3] at Naval Postgraduate School, later incorporated into Delta3D software, uses only two tree levels of detail – full geometry and billboard (a single, view-aligned image).

All of these existing systems instantiate copies of just a few unique trees to create forests and consequently do not meet our criterion for uniqueness/persistence. Moreover, these systems select level-of-detail models based on distance, a very effective measure in urban settings or sparsely populated natural environments, but one that is not appropriate in dense vegetation where visibility is the primary metric of visual complexity.

In contrast to these existing systems, the research reported in this paper meets the uniqueness/persistence criteria and effectively addresses three major challenges raised by interactive rendering of complex forest scenes.

First, we introduce an inertial level-of-detail (LOD) model for rendering large numbers of unique individual trees. Each level of detail approximates the original tree and can be drawn faster than the original. The model minimizes popping artifacts by linearly interpolating over time between discrete LODs whenever a transition is necessary. This technique is relevant to objects, such as trees, that are not amenable to progressive surface simplification methods [4].

Second, we present a visibility-based walk-through framework. The framework renders objects in front-to-back order, each time using visibility, in addition to the traditional projected object size, to determine the optimal LOD. Object visibility is computed for each frame at run-time, making use of hardware-assisted occlusion queries. No pre-computation is necessary, thereby allowing dynamic scene modification (for example, burning trees due to explosions and fires, or swaying tree branches due to wind and rain).

Third, the system we develop runs on a standard PC platform with a commodity graphics system. This supports cost-effective delivery of trainers, a critical issue for wide acceptance in the simulation industry. Moreover, although our focus is on walk-throughs of forest environments, the techniques presented here have broad application to interactive rendering.

The remainder of this paper is organized as follows. The tree model is briefly reviewed in Section 2. The inertial LOD model and the underlying discrete and continuous LOD models are described in Section 3. The visibility-based framework is presented in Section 4, which also includes performance results from a variety of experiments. Conclusions and future work are reported in Section 5.

## 2. Tree Model

Computer modeling of trees has been an active research area for a number of decades. Cohen [5] and Honda [6] pioneered computer models for tree generation. Prior to that, Ulam [7] and Lindenmayer [8] studied purely mathematical models of plant growth. Lindenmayer proposed a set of rules for generating text strings as well as methods for interpreting these strings as branching structures. His method, now referred to as L-system, was subsequently extended to allow random variations, and to account for the interaction of growing plants with their environment [9,10,11]. A wide range of environmental factors, including exposure to sunlight, distribution of water in the soil, and competition for space with other plants, were incorporated into the model, leading to biologically accurate ecosystems. Ray-tracing methods were employed to produce visually stunning images of vegetation scenes [11,9] at the cost of high computation complexity. Another botanical tree model was proposed by de Reffye *et al.* [12].

The above approaches focus on biologically accurate generation of vegetation, not necessarily on the graphical rendering. A number of methods have been proposed for generating plants with the goal of visual quality without relying on botanical knowledge. Oppenheimer [13] used fractals to design self-similar plant models. Bloomenthal [14] assumed the availability of skeletal tree structures and concentrated on generating tree images using splines and highly detailed texture maps. Weber and Penn [15], and Aono and Kunii [1] proposed various procedural models. Chiba *et al.* [17] utilized particle systems to generate images of forest scenes.

Looking at these various approaches, there are several compelling reasons to choose L-systems for simulation/training applications:

- Given different random number generator seeds, the same set of rules can generate a variety of instances of the same tree species.
- An L-system provides a very compact representation of a tree: a set of simple rules (common to all the trees of the same species) and a random number generator seed are sufficient to generate a unique tree. L-system methods for advanced biological simulation exist, should a need arise for such accuracy.
- L-system rules are readily available for a variety of tree species. Thus, environments from different geographic locations can be generated quickly and automatically, requiring no additional modeling by artists.

To generate trees for the forest walk-through framework we use the stochastic L-system 5, described by Prusinkiewicz *et al.* [10]. This system has the property of simplicity, and yet it generates the branching structure of a family of elegant looking trees.

$$\begin{aligned} \omega: & FA(1) \\ p_1: & A(k) \rightarrow /(\varphi)[+(\alpha)FA(k+1)] - (\beta)FA(k+1): \\ & \qquad \qquad \qquad \min\{1, (2k+1)/k^2\} \\ p_2: & A(k) \rightarrow /(\varphi) - (\beta)FA(k+1): \\ & \qquad \qquad \qquad \max\{0, 1 - (2k+1)/k^2\} \end{aligned}$$

The initial string is specified by the axiom  $\omega$ , which consists of two modules. Module  $F$  is rendered as a branch segment. Module  $A()$  is used for “growing” the tree and has no graphical interpretation (the integer in the parentheses denotes the number of times rewriting rules have been applied). Modules  $+$ ,  $-$  denote rotation around the z-axis (the axis pointing back to front with  $+$  being counterclockwise,  $-$  being clockwise), while  $/$  denotes rotation around the y-axis (the one pointing toward the sky). The angles for the rotations are specified in the parentheses ( $\alpha = 32^\circ$ ,  $\beta = 20^\circ$ ,  $\varphi = 90^\circ$ ). There are two possibilities when rewriting module  $A(k)$ . Rewriting (production) rule  $p_1$  produces two branches with probability  $\min\{1, (2k+1)/k^2\}$ , while  $p_2$  produces a single branch segment. Thus, the tree has a high probability of bifurcating at the lower branches, but normally exhibits single offshoots at higher branches. For more details on L-systems and their interpretation refer to [10].

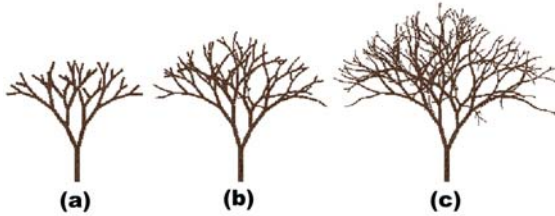
In order to produce models suited for real-time rendering, our interpretation of the L-system strings has a number of minor differences from that of Prusinkiewicz *et al.* First, the length of a branch segment in the modified model is decreased with each rewriting step. Second, leaf clusters are rendered as textured *cross-polygon* impostors. A cross-polygon is made up of two quadrangles, intersecting along their respective center lines. Leaf clusters are attached to the last three levels of the tree, whereas the original model due to Prusinkiewicz *et al.* used only the last level. Furthermore, the color of a leaf cluster depends on the branch level: interior clusters are darker to approximate light occlusion within the tree canopy.

The L-system description of a tree is stored in a singly-linked list, with a list element for each module. Times to render a single tree after a varying number of L-system productions are listed in Table 1. Times were averaged over 100 randomly generated trees; time to clear and swap the buffers is not included (the experiments did clear and swap the buffers, but the time for these operations was determined separately and subtracted). The numbers of branch segments (cylinders) and leaf clusters are also listed. Experiments were conducted on a PC equipped with a 2.4GHz Xeon processor, 512MB RAM, and an nv35 (GeForce fx5900) graphics card. The application was written in ANSI C++ using OpenGL 1.5. Textures for leaf clusters were mipmapped 512x512 RGBA images.

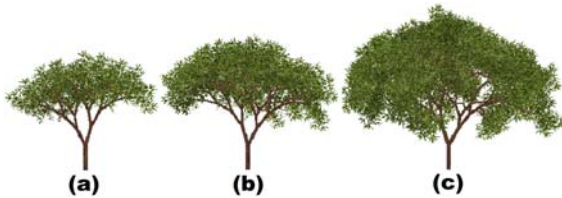
**Table 1.** Time required to render a single tree generated after different number productions

n. prod	n. cyls	n. leaves	time (ms)	FPS
4	15	14	0.44	2288.33
8	170	139	0.83	1207.73
12	706	443	2.93	341.56
16	1979	1016	6.72	148.83
20	4595	1933	13.92	71.83

The trees for the walk-through were produced by 12 productions of the modified L-system. Branch structure as well as foliated trees, produced after 6, 8, and 12 productions, are shown in Figures 1 and 2, respectively.



**Figure 1.** Branch structures: 6, 8, and 12 productions



**Figure 2.** Foliated structure: 6, 8, and 12 productions

### 2.1. Tree Levels of Detail

As can be seen in Table 1, due to the high complexity of the tree structures, scenes with only a trivial number of fully rendered mature trees can be displayed at 30 frames per second. To accommodate the rendering of large forests, tree models must be simplified. Traditional simplification methods, such as multi-resolution meshes [4], assume a large connected mesh structure. These methods are not suitable for trees because they consist of many disconnected polygons (leaves or leaf clusters). For example, Remolar *et al.* [18] showed that a general mesh simplification approach resulted in a “pruned” appearance. Thus, a number of custom methods have been proposed to speed up tree rendering. These can be roughly categorized into two approaches, geometry-based and texture-based.

Oppenheimer [13] used polygonal cylinders to render large branches, replacing the small ones with lines. Weber and Penn [15] proposed a similar simplification approach by replacing cylindrical branch segments with lines and leaf polygons with points or not rendering them at all. Branches and leaves are grouped into “masses” and a decision on the rendering method is made for each mass. When selecting the level of detail, Weber and Penn take into consideration the field of view in addition to the distance from the viewer, which “compensates for the effect of a telephoto lens” [15]. Deussen *et al.* [19] sampled each object to obtain a list of points. Based on the projected size of each object,

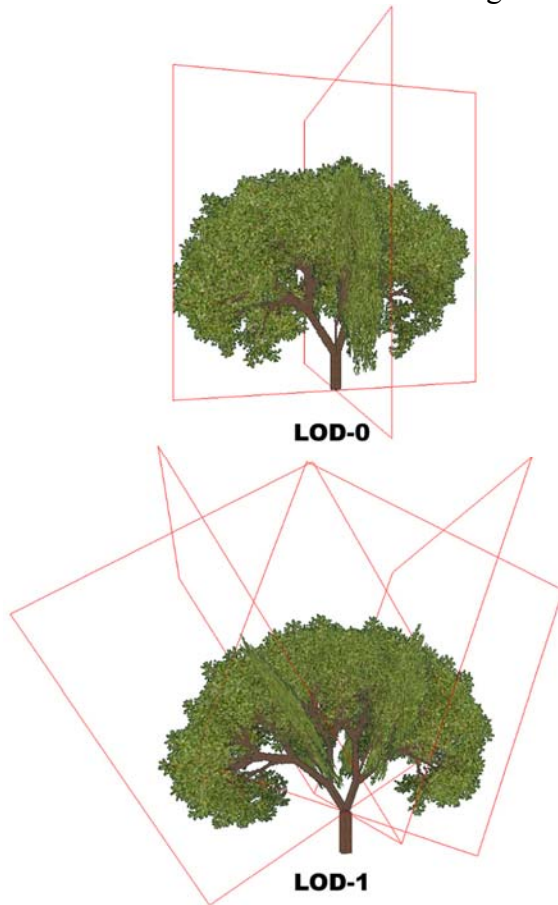
a decision is made at run-time whether to render triangles or the point-approximation. The number of points used in the approximation is also determined at run-time. Each list is ordered randomly to ensure that the overall shape of the object is preserved no matter how many points are rendered. Thus, the method produces no popping artifacts. Complex plants are stored in shallow oct-trees, with a separate point list for each cell. Marshall [20] used a non-uniform tetrahedral subdivision of 3D space to render images in three steps: tetrahedra visibility computation, subdivision refinement, and rendering the plant geometry (or approximating it with Gouraud shaded tetrahedra that contain it). The method was implemented in software and required minutes to render each frame of an 80-tree scene. Remolar *et al.* [31,32] proposed a multi-resolution simplification scheme by iteratively merging the leaves of a tree. Popping artifacts occur, though the approach maintains the over-all shape of a tree.

Several texture-based methods for reducing the geometric complexity of trees have been proposed. Jakulin [22] approximated a tree with *slicings*, each consisting of multiple layers of textured polygons. Several slicings are pre-computed from multiple views of a given tree. To reduce the popping artifacts, each tree is approximated by blending the two slicings that most closely match the viewing direction. Meyer, Neyeret, and Poulin [23] render a tree as a 3-level hierarchy, each level composed of transformed instances of the lower level elements. A set of bidirectional textures is pre-computed for a representative element of each hierarchy level. Separate textures are obtained for direct and ambient lighting. A tree is approximated by selecting and blending 5 textures most closely matching the viewing direction and lighting conditions. The method requires a large amount of storage for bidirectional texture maps, as well as a long pre-computation step. Max *et al.* [24,25] proposed to reconstruct trees from texture maps that contain normal and depth information in addition to RGBA color for each pixel. Several texture maps are pre-computed from a number of viewing directions, evenly distributed on a bounding sphere of a tree. A tree is approximated by rendering only the texture maps most closely matching the viewing direction. During rendering, each texel is transformed using the pre-computed depth information. In [24] texture maps were pre-computed for each hierarchical level of a tree, rather than for the entire tree only. The approach required hours of pre-computation and seconds to render an approximation for a single tree. However, rendering performance would likely be dramatically improved using modern graphics hardware with fragment shaders.

Among the approaches discussed above, only Max [24,25] utilizes a tree hierarchy. Thus, the other approximations assume static scenes and associated limitations. For example, swaying in the wind or breaking off branches cannot readily be implemented. Furthermore, all the approaches described tacitly assume that complex scenes are obtained by instantiating multiple copies of a few distinct plants. For the approach proposed by Deussen *et al.* [19], a scene with a large number of unique trees would require a prohibitive amount of storage for all point lists. Similarly, the approaches in [19,24,25,23] would require large amounts of texture memory to store the images for each unique plant.

### **3. Hierarchical Level of Detail**

In what follows, we make the simplifying assumption that a tree consists of branch segments and leaf clusters. The *level* of a branch segment is the production in which the corresponding module was added. Any level- $k$  branch segment is connected to a single level- $(k - 1)$  segment, or *parent*, and may have multiple *children*, or level- $(k + 1)$  segments connected to it. We say that a tree is *rooted* at the level-0 branch segment, which by construction does not have a parent. Given any branch segment we define the *subtree* to be the set of all successor segments and associated leaf clusters. A *k-subtree* is a subtree rooted at a level- $k$  branch segment.



**Figure 3.** Cross-polygons for two LODs

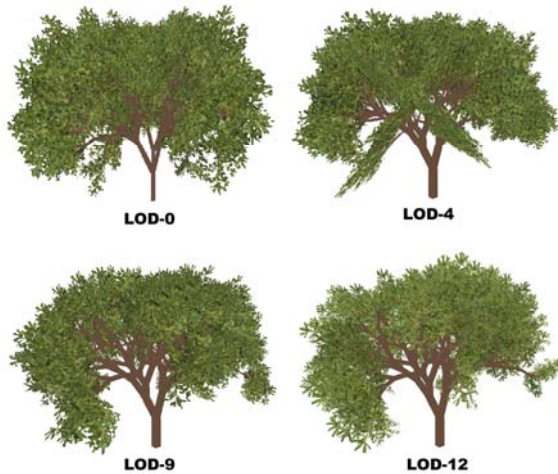
### 3.1. Discrete LOD Model

We propose a hierarchical scheme for computing tree levels of detail, similar to Max's approach [24]. Human perception experiments [26,27] suggest that the structure of lower level branches is critical to memorization and recognition. Thus, in the  $k^{\text{th}}$  level of detail, LOD- $k$ , replaces each  $k$ -subtree with a textured cross-polygon impostor. Each instance of the impostor is transformed in 3D space just as the  $k$ -subtree it approximates. Two sample LODs are shown in Figure 3, outlining the boundaries of the textured cross-polygon impostors. The lowest level of detail, LOD-0 replaces the entire tree with a single cross-polygon, which is rendered over five times faster than the full tree.

The LOD- $k$  textures are rendered from several views of an arbitrary  $k$ -subtree. In our implementation two texture maps are generated from two perpendicular views of the subtree. Each texture map is a 256x256 image, scaled appropriately at run-time by the

graphics hardware. The same set of textures is used for all trees of the same species. Since the silhouette of a subtree is the same from any two views at an angle of  $180^\circ$  to each other, we used the same texture for both sides of a quadrangle. While these simplifications provide only a rough approximation of the fully detailed object, due to occlusion and distance to objects rendered at lower levels of detail, the differences are negligible.

Four sample levels of detail for a 12-level tree are shown in Figure 4, LOD-0 and LOD-12 being the lowest and the highest level of detail, respectively. Each LOD was rendered at the same angle and distance from the viewer. Note that the trunk of LOD-0 appears thinner because both polygons in the impostor are at a  $45^\circ$  angle to the viewer. Orientation of individual impostors is less significant for higher levels of detail.



**Figure 4.** Four levels of detail for a 12-production tree

Times to render a single tree at varying levels of detail are listed in Table 2. The times were averaged over 100 randomly generated trees.

**Table 2.** Time to render a single 12-production tree at different levels of detail

LOD	time (ms)	FPS	speedup
0	0.54	1840.43	5.39
4	0.89	1122.59	3.29
8	1.41	708.04	2.07
12	2.93	341.56	1.00

### 3.2. Continuous LOD Model

To reduce the popping artifacts when switching between discrete LODs, we extend the discrete model to a continuous one. The continuous model renders two LODs, linearly interpolating their translucencies. Due to the hierarchical nature of the LOD model both LODs share branch segments: the higher LOD needs to be fully rendered, but only the impostor cross-polygons need to be drawn for the lower LOD. Thus, the polygon count is only slightly higher than that of the higher LOD alone.

Implementation of the continuous LOD model currently requires two rendering passes since blending two objects often results in an image that is darker than a rendering

of a single opaque object [22]. Assuming the OpenGL EXT\_blend\_func\_separate extension, states are enabled as shown in Table 3 when objects are rendered in the front-to-back order (the framebuffer is initially set to (0, 0, 0, 0), transparent black).

**Table 3.** Blending operations for two-pass rendering

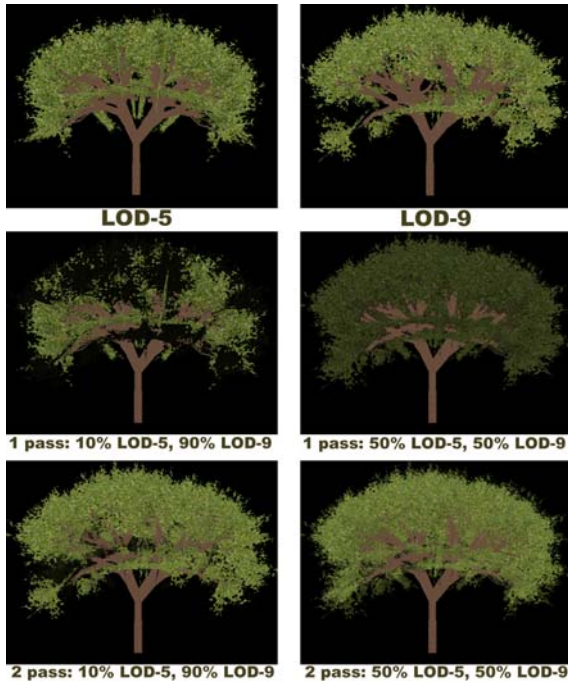
Pass	Depth		Alpha blending		
	Function	Mask	Comp.	Source	Destination
1st	LEQUAL	TRUE	RGB	SRC_ALPHA	ZERO
			Alpha	ONE	ZERO
2nd	GREATER	FALSE	RGB	SRC_ALPHA_SATURATE	ONE
			Alpha	ONE	ONE

The order in which objects are rendered during these passes, closest first (front-to-back) is essential so occluding parts are available when a fragment is being considered. In our experiments, two-pass rendering resulted in between 60% and 70% increase in rendering time. If the hardware were capable of selecting a blending function based on the outcome of the depth test, only a single pass would be required. A multi-pass technique for order-independent rendering of transparent objects is described in [28].

The results for the LOD-5 and LOD-9 are shown in Figure 5. The two-pass approach results in correct brightness and avoids dark spots, which are unavoidable with a single pass due to the depth test.

### 3.3. *Inertial LOD Model*

While the continuous LOD model nearly eliminates popping artifacts when visibility of an object changes smoothly, popping still occurs when visibility changes abruptly. For example, visibility can drastically change when a viewer emerges from inside a tree canopy. In such a case it is possible that the current frame is blending discrete LOD-*a* with LOD-*b*, whereas the previous frame required blending LOD-*c* and LOD-*d* (where *a*, *b*, *c*, *d* are all distinct). To mitigate popping effects in such cases, we propose the *Inertial LOD* model.



**Figure 5.** Continuous LOD

Whenever a change in detail is needed, the inertial LOD model interpolates the blend factor of the continuous LOD over a number of frames (even if the visibility does not change during those frames). Thus, two LODs are maintained, referred to as *past* and *target*. Once the transition is complete only the discrete target LOD is rendered, avoiding the two-pass overhead inherent in the continuous model. The target LOD may need to be updated while the transition is in progress. Let  $a$  and  $b$  be the past and target LODs, respectively, and  $a < b$ . If the new target LOD  $c$  is less than  $a$ , then  $b$  becomes the past LOD; otherwise  $a$  remains the past LOD. The case where  $a > b$  is processed similarly. The blend factor has to be adjusted considering its current value (as opposed to resetting it) so as to avoid abrupt changes in the past LOD.

#### 4. Visibility-based Walk-through Framework

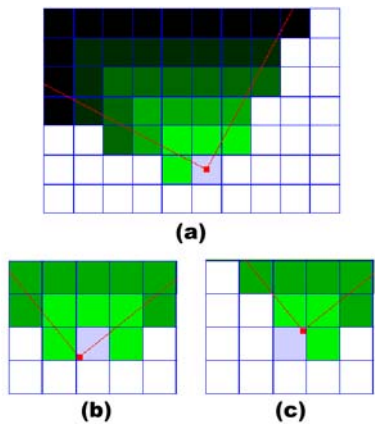
Walk-through applications are fundamentally different from fly-overs [29,23] and require different LOD management approaches. The walk-through viewer is arbitrarily close to some objects, requiring them to be rendered at the highest level of detail in order for the smallest elements to be distinguishable. We address this via a visibility-based LOD management framework.

A large body of research exists for visibility-based object culling [30]. The methods fall into two general categories: *object* and *image* based. Object-based methods clip and cull primitives against a set of pre-selected objects or occluders [31,32] and culling takes place before any rendering. Some examples of object-based methods include Prioritized-Layered Projection algorithms [32], Binary Space Partition algorithms, and algorithms using shadow frusta [33]. Object-based algorithms perform best in the presence of a small number of portals or large occluders, which makes them unsuitable for a forest walk-through since the objects making up trees (branch segments and leaf clusters) are

many and tend to be relatively small. Image-based visibility methods cull in window coordinates, thus rendering is required. The z-buffer algorithm is the most common example of image-based culling. In order to cull primitives or even objects, depth tests are performed on the projections of the bounding boxes. Computation is accelerated by hierarchical methods, such as Hierarchical Z-Buffer [34] and Hierarchical Occlusion Buffer [35]. Related methods were proposed by Bartz *et al.* [36,37] and Hey *et al.* [38,39].

While the above methods were designed to cull polygons, we adopt a similar approach to select levels of detail. We propose an image-based level of detail selection method for rendering large scenes at interactive frame rates. Given an object, the appropriate level of detail is chosen at run-time and is based on visibility and projected size. The objects within the frustum are rendered in the front-to-back order, which enables interactive visibility computation. Furthermore, this approach does not require a costly pre-computation step.

To facilitate front-to-back sorting from any viewing position, the terrain is divided into a two-dimensional rectangular grid and objects are distributed among the grid cells. The system could readily be extended to employ a quad-tree, which would improve performance for non-uniform distributions. Snapshots of the grid, the viewpoint and viewing frustum are shown in Figure 6. The light grey grid cell contains the viewpoint and non-white grid cells are either intersected or contained by the viewing frustum. There are  $8D$  cells at distance  $D$  from the viewpoint, where distance is the radius of a "square" circle. A radial coordinate system is utilized to traverse only the cells within the view frustum for each distance. Care must be taken when computing the boundary of cells. This is evident in Figures 6b and 6c, where the viewing direction is the same but the leftmost cells are two indices apart due to different viewing positions with the same cell.



**Figure 6.** Terrain grid and view frustum

#### 4.1. Visibility Computation

The majority of modern graphics cards support occlusion queries [36,37] that, given some primitives, determine how many resulting fragments would pass the depth/stencil test without actually modifying the render target. The visibility of a given object can be

computed by issuing two queries with different depth functions – one passes the fragments that are “in front” of the current z-buffer; the other passes the fragments that are behind. The sum of the queries’ results approximates the projected size of an object after clipping. It is an approximation since some pixels are counted twice due to self-occlusion within an object. For example, when rendering the full level of detail of a 12-production tree, an average of 2.21 fragments (the maximum was 11) were written to each pixel position that was modified. In practice, the lowest LOD of an object is used to approximate visibility.



**Figure 7.** Contribution of the trees in grid cells distance 5 through 10 away from the viewpoint

Contribution to the final image by the objects (in this case trees) in grid cells between distances 5 and 10 (inclusively) from the viewer are shown as non-white pixels in Figure 7. Since it is difficult to distinguish the features of individual objects (trees in this case), lower LODs could be used to increase performance. Furthermore, the most prominent features are the lower-level branches, justifying the hierarchical LOD model. The following framework utilizes the OpenGL occlusion query extension to select LODs at run-time:

1. Render the objects in the viewpoint cell and all cells at distance 1 at the highest level of detail.
2. **While** the highest LOD in the previous step is greater than LOD-THRESHOLD **do**  
     **For each** cell at distance  $d$  within the frustum **do**  
         **For each** object in the current cell **do**  
             Compute visibility  
             Select LOD  
             Render the chosen LOD

The while loop in step 2 iterates as long as the highest level of detail selected in the previous step is above a user-specified threshold, LOD-THRESHOLD.

#### 4.2. *Experimental Results*

The testing hardware and software are described in Section 2 above. The results were averaged over 200 consecutive frames of circular movement through forest scenes, starting at the center. Scenes were rendered at 800x600 resolution. Four levels of detail were selected based on visibility and projected size (in pixels):

- LOD-12. Visibility: 55% to 100% and size >10K.
- LOD-8. Visibility: 20% to 55% and size > 10K.
- LOD-4. Visibility: 10% to 20% or 1K < size <10K.
- LOD-0. Visibility: 3% to 10% or 50 < size < 1000.  
no rendering for under 3% visibility or size < 50.



**Figure 8.** A forest scene at full level of detail



**Figure 9.** A forest scene using the walk-through framework



**Figure 10.** A forest scene showing levels of detail

The framework rendered each frame until all trees in the previous iteration of the while loop were rejected (visibility lower than 3% or fewer than 50 fragments contributed). On average, trees up to distance 12 were rendered in our experiments. The same 400-tree forest scene is shown in Figures 8 through 10. Each tree is rendered in full level of detail in Figure 8, while the walk-through framework was utilized to render Figures 9 and 10. In Figure 10, LOD-0 trees are colored blue, LOD-4 trees are colored

bright green, LOD-9 trees are colored gold, while LOD-12 trees are textured without any additional coloring.

Using the framework with inertial LOD model on forests with 100, 400, 1600, and 2500 distinct trees, respective speedups of 1.9, 4, 13.7, and 20.9 were achieved when compared to the view-frustum culling alone. The inertial LOD model was set to complete the transition between discrete LODs in 10 frames. Average frame rates for view-frustum culling only, discrete LOD, continuous LOD, and inertial LOD are summarized in Table 4. Note that even though discrete LOD results in the highest performance, in practice it exhibits the most evident popping artifacts.

**Table 4.** Walk-through performance

Number of trees	Average FPS			
	cull	disc.	cont.	inert.
100	10.36	25.29	14.73	19.35
400	3.09	17.65	8.89	12.47
1600	0.82	16.93	9.56	11.27
2500	0.52	15.67	7.23	10.9

## 5. Discussion and Future Work

An inertial LOD model and a visibility-based walk-through framework were presented in this paper. The inertial LOD model can be used to minimize popping artifacts when switching between discrete LODs. This is critical for objects, such as trees, that are not amenable to progressive surface-simplification methods. A forest walk-through was used as a sample application, with the framework achieving speedups approaching 20 times that seen with view frustum culling alone. A two-pass hardware-efficient implementation of the hierarchical LOD model was described. This technique would require only a single pass (implying a 40% increase in performance) if graphics hardware allowed choosing the blending function based on the outcome of the depth test.

An interesting direction for future work is to extend the visibility computation to consider object fragmentation in the final image. Consider two equally sized instances of the same object that is 50% visible. The instance which is visible as one contiguous block of pixels may need a higher LOD than the one that is visible as a collection of small disjoint blocks. Such LOD-selection technique would require fast reading of the framebuffer as well as efficient image processing techniques.

## 6. References

- [1] Sextant suite of tools for rapid urban 3D database construction, mission planning, and briefing. Object Raku Technology, Inc. <http://www.objectraku.com/>
- [2] M. Zyda, A. Mayberry, C. Wardynski, R. Shilling, and M. Davis. "The MOVES Institute's America's Army Operations Game." *Proceedings of 2003 Symposium of Interactive 3D Graphics* (2003): 219-220.
- [3] W. D. Wells. *Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations (GENETICS)*. Ph.D. dissertation, Naval Postgraduate School (2005).

- [4] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann (2002).
- [5] Cohen, D. "Computer simulation of biological pattern generation processes." *Nature*, 216 (1967): 246-248.
- [6] Honda, H. "Description of the form of trees by the parameters of the tree-like body: effects of the branching angle and the branch length on the shape of the tree-like body." *Journal of Theoretical Biology*, 31 (1971): 331-338.
- [7] S. Ulam. On some mathematical properties connected with patterns of growth of figures. *Proc. of Symposia on Applied Mathematics*, 14 (1962): 215-224.
- [8] A. Lindenmayer. "Mathematical models for cellular interactions in development, I & II." *Journal of Theoretical Biology*, 18 (1968): 280-315.
- [9] R. Měch, and P. Prusinkiewicz. "Visual models of plants interacting with their environment." *Proc. of SIGGRAPH 96* (1996): 397-410.
- [10] P. Prusinkiewicz, M. James, R. Měch. "Synthetic topiary." *Proc. of SIGGRAPH 94*, pp. 351-358, 1999.
- [11] Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz P. "Realistic modeling and rendering of plant ecosystems." *Proc. of SIGGRAPH 98* (1998): 275-286.
- [12] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. "Plant models faithful to botanical structure and development." *Proc. of Siggraph 88* (1988): 151-158.
- [13] P. Oppenheimer. "Real time design and animation of fractal plants and trees." *Proc. of SIGGRAPH 86* (1986): 55-64.
- [14] Bloomenthal, J. "Modeling the mighty maple." *Proc. of SIGGRAPH 85* (1985): 305-311.
- [15] J. Weber, J. Penn. "Creation and rendering of realistic trees." *Proc. of SIGGRAPH 95* (1995): 119-128.
- [16] Aono, M., and Kunii T. "Botanical tree image generation." *IEEE Computer Graphics and Applications*, 4(5) (1984): 10-34.
- [17] Chiba, N., Muraoka, K., Doi A., and Hosokawa J. "Rendering of forest scenery using 3D textures." *The Journal of Visualization and Computer Animation* 8 (1997): 191-199.
- [18] I. Remolar, M. Chover, Ó. Belmonte, J. Ribelles, and C. Rebollo. "Real-time tree rendering." Technical Report DLSI 01/03/2002, Departamento de Lenguajes y Sistemas Informáticos, Univeristat Jaume I (2002).
- [19] Deussen, O., Colditz, C., Stamminger, M., and Drettakis G. "Interactive visualization of complex plant ecosystems." *Proc. of IEEE Visualization 02* (2002): 219-226.
- [20] D. Marshall. "Multiresolution rendering of complex botanical scenes." *Proc. of Graphics Interface 97* (1997): 96-104.
- [21] I. Remolar, M. Chover, Ó. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage. *Proc. of Eurographics 02* (2002): 397-404.
- [22] A. Jakulin. "Interactive vegetation rendering with slicing and blending." *Proc. of Eurographics 2000*, Short Presentations (2000).
- [23] A. Meyer, F. Neyeret, and P. Poulin. "Interactive rendering of trees with shading and shadows." *Proc. of Eurographics Workshop on Rendering* (2001): 183-196.

- [24] N. Max. "Hierarchical rendering of trees from precomputed multi-layer z-buffers." In *Eurographics Workshop on Rendering 96* (1996): 165-174.
- [25] N. Max, O. Deussen, and B. Keating. "Hierarchical image-based rendering using texture mapping hardware." In *Eurographics Workshop on Rendering 99* (1999): 57-62.
- [26] V. K. Sims, J. M. Moshell, C. E. Hughes, J. E. Cotton, and J. Xiao. Recognition of computer generated trees. *Proceedings of the Human Factors and Ergonomics Society* 46 (2002):2215-2219.
- [27] V. K. Sims, J. M. Moshell, C. E. Hughes, J. E. Cotton, and J. Xiao. Salient characteristics of virtual trees. *Proceedings of the Human Factors and Ergonomics Society* 45 (2001): 1935-1938.
- [28] Everitt, C. "Interactive order independent transparency." NVidia whitepaper (2002).
- [29] Decaudin, P., and Neyret F. "Rendering forest scenes in real-time." *Eurographics Symposium on Rendering* (2004): 93-102.
- [30] Cohen-Or, D., Chrysanthou, Y., Silva, C. T., and Durand F. "A survey of visibility for walkthrough applications." *IEEE Transactions on Visualization and Computer Graphics* 9(3) (2003): 412-431.
- [31] Coorg, S., and Teller, S. "Real-time occlusion culling for models with large occluders." *1997 Symposium on Interactive 3D Graphics* (1997): 83-90.
- [32] J. T. Klosowski, and C. T. Silva. "Efficient conservative visibility culling using the prioritized-layered projection algorithm." *IEEE Transactions on Visualization and Computer Graphics* 7(4) (2000): 365-379.
- [33] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. "Accelerated occlusion culling using shadow frusta." *Proc. of the 13<sup>th</sup> Annual ACM Symposium on Computational Geometry* (1997): 1-10.
- [34] Greene, N., Kass, M., and Miller, G. "Hierarchical z-buffer visibility." *Proc. of SIGGRAPH 93* (1993): 231-240.
- [35] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. "Visibility culling using hierarchical occlusion maps." *Proc. of Siggraph 97* (1997): 77-88.
- [36] Bartz, D., Meißner M., and Hüttner, T. "Extending graphics hardware for occlusion queries in OpenGL." *Proc. of Workshop on Graphics Hardware 98* (1998): 10-34.
- [37] Bartz, D., Meißner, M., and Hüttner T. "OpenGL-assisted occlusion culling for large polygonal models." *Computer and Graphics* 23(5) (1999): 667-679.
- [38] Hey, H., and Tobler R. F. "Lazy occlusion grid culling." *Technical Report TR-186-2-99-09*, Vienna University of Technology (1999).
- [39] Hey, H. R. Tobler, F., and Purgathofer W. "Real-time occlusion culling with a lazy occlusion grid." *Technical Report TR-186-2-01-02*, Vienna University of Technology (2001).

## Acknowledgements

The research presented here is partially supported by the National Science Foundation (SES0527675), the Army Research, Development and Engineering Command, Orlando, and the VIRTE (Virtual Technology and Environment) project funded through the Office of Naval Research.

## **Author Biographies**

**Dr. Charles E. Hughes** is a Professor and the Associate Director of the School of Electrical Engineering and Computer Science at the University of Central Florida (UCF). He also holds academic appointments in Digital Media, Modeling&Simulation, Text&Technology and the Institute for Simulation & Training (IST). He is the Director of the Media Convergence Laboratory, an interdisciplinary research group located at IST. His research interests are in interactive computer graphics, and virtual and mixed reality. He applies his work in mixed reality to entertainment, training, education, performance assessment and rehabilitation. Dr. Hughes received his BA in Mathematics from Northeastern University in 1966, and his PhD and MS in Computer Science from Penn State University in 1970 and 1968, respectively.

**Dr. Paulius Micikevicius** is a Developer Technology Engineer at NVIDIA. Prior to joining NVIDIA, he was an Assistant Professor of Computer Science at Armstrong Atlantic State University. As a post-doctoral researcher, he led the development of a mixed reality 3D graphics engine at the Media Convergence Laboratory at UCF. His research interests include parallel computing, interactive graphics, algorithms and graph theory. Dr. Micikevicius holds a Ph.D. in Computer Science from the University of Central Florida, and a BS in Computer Science from Midwestern State University.