Visibility-based Forest Walk-through Using Inertial LOD Model

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

Charles E. Hughes
School of Computer Science
University of Central Florida
Orlando, Florida
ceh@cs.ucf.edu

Rendering large scenes with many complex objects places prohibitive requirements on modern 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 LOD management techniques for achieving interactive frame rates for a forest walk-through. The proposed framework selects the LOD based on object visibility, in addition to the projected size. Run-time visibility computation is used to efficiently schedule dynamic scene modification, such as interactive forest growing. We also propose an inertial LOD model to minimize popping artifacts – rather than instantly switching the LODs, discrete LODs are smoothly blended over a number of frames. The proposed techniques can be readily adapted for scenes other than forests, providing support for simulations involving both static and dynamic environments, whether synthetic natural (e.g., dense vegetation) or man-made (e.g., urban landscapes).

Keywords:
Level-of-detail, visibility, occlusion, walk-through.

1. Introduction

Rendering requirements for walk-throughs of large scenes with many complex objects exceed the capabilities of even the fastest graphics hardware. Thus, techniques for managing scene and object complexity are necessary in order to achieve interactive frame rates. The two major contributions of this paper address these issues.

First, we propose an inertial level of detail (LOD) model. 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 [[17] chapter 2]. We describe an implementation for modern graphics hardware that requires two passes for each object in transition.

Second, we propose a visibility-based walk-through framework. The framework renders objects in the 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 at run-time for each frame and makes use of hardware-assisted occlusion queries. Thus, a precomputation step is not necessary and the scene can be modified dynamically. We illustrate this by increasing the age for the entire forest, while prioritizing tree growth by their visibility.

The tree model is briefly reviewed in Section 2. The inertial LOD model, as well as

the underlying discrete and continuous LOD models, are described in Section 3. The visibility-based framework is proposed in Section 4, which also includes experimental timings. Conclusions and future work are listed in Section 5. The proposed techniques are not restricted to rendering forest walk-throuhgs. The inertial LOD model is applicable to any objects, while visibility-based walk-through framework can utilize any LOD model.

2. Tree Model

While computer modeling and rendering of trees and forest scenes has been studied quite extensively [[4],[8], ,[19],[20],[22],[23],[24],[28] and others], little research has been published on traversing forest scenes at interactive frame rates [[9],[21],[15],[20],[25]]. Rendering photo-realistic forest scenes requires minutes to hours of computation per frame. In order to be useful for visual simulation, training, and scientific applications, a forest walk-through framework may have to satisfy a number of additional requirements:

- Each tree should be unique. Instantiating affinely transformed copies of the same tree is unacceptable in some applications. Of course, to increase performance such system can always be scaled down by using a only a few unique specimen.
- The exact same tree must be rendered, given the same viewpoint position and direction. Thus, tree structures cannot be generated randomly when their positions fall within the viewing frustum and discarded when they are no longer visible.

To generate trees for the proposed forest walk-through framework we have modified the stochastic L-system 5, described by Prusinkiewicz *et al*. [[24]]. L-systems were chosen because they provide a compact representation of a tree as well as the methods necessary for advanced biological simulation. The rules of the system are as follows:

ω: $FA(1)$

p_1: $A(k) \rightarrow /(\varphi)[+(\alpha)FA(k+1)] - (\beta)FA(k+1)$: $\min\{1, (2k+1)/k^2\}$

p_2: $A(k) \rightarrow /(\varphi) - (\beta)FA(k+1)$: $\max\{0, 1-(2k+1)/k^2\}$

The initial string is specified by the axiom ω, 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, while / denotes rotation around the y-axis. The angles for the rotations are specified in the parentheses (α = 32°, β =20°, φ = 90°). 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. For more details on L-systems and their interpretation refer to [[24]].

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.

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

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

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.

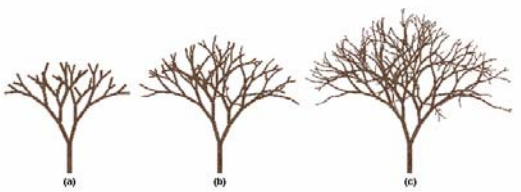

Figure 1. Branch structures: 6, 8, and 12 productions

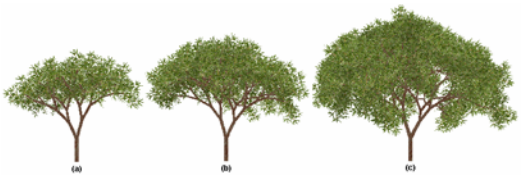

Figure 2. Foliated structure: 6, 8, and 12 productions

3. Hierarchical Level of Detail

A simplifying assumption is made 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.

3.1. Discrete LOD Model

We propose a hierarchical scheme for computing tree levels of detail, similar to Max's approach [[19]]. Human perception experiments [[26],[27]] suggest that the structure of lower level branches is critical to memorization and recognition. Thus, in the k^{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.

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° 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, we observed minimal discrepancies due to occlusion and distance to objects rendered at lower levels of detail.

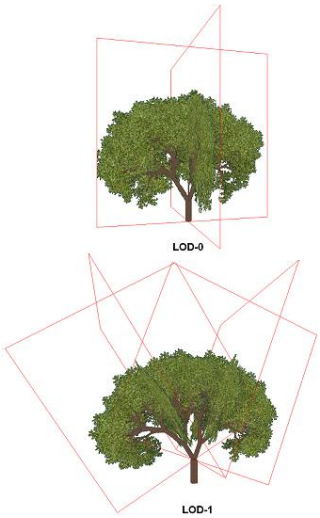

Figure 3: Cross-polygons for LOD-0 and LOD-1

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° angle to the viewer. Orientation of individual impostors is less significant for higher levels of detail.

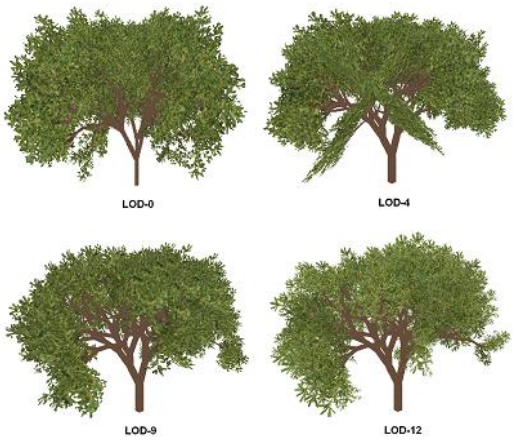

Figure 4: Four levels of detail for a 12-production tree

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.

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

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

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 [[15]]. Assuming the EXT_blend_func_separate OpenGL extension, the following states are enabled when objects are rendered in the front-to-back order (the framebuffer is initially set to (0, 0, 0, 0) color):

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

Note that the back-to-front ordering is less likely to produce correct results than front-to-back order: the first pass combines the source fragment with the result of all the fragments previously rendered to that location. Thus, it very unlikely that a fragment rendered in the second pass is properly "inserted" as its opacity should depend only on the opacity of the corresponding fragment rendered in the first pass. This problem is greatly reduced (if not avoided altogether) by rendering the objects in a loose front-to-back order. 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 [[10]].

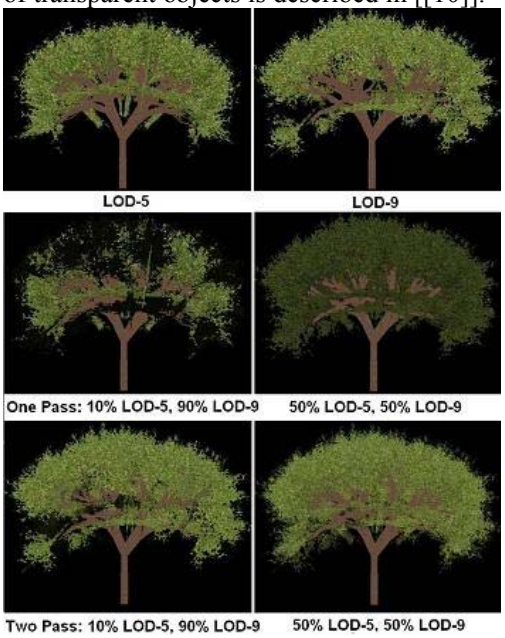

Figure 5: Continuous LOD

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.

3.2. 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 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.

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: *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 [[7],[21]]. and require different LOD management approaches. The walk-through viewer is arbitrarily close to some objects, which need to be rendered at the highest level of detail in order for the smallest elements to be distinguishable. We propose a visibility-based LOD management framework.

A large body of research exists for visibility-based object culling [[5]]. 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 [[6],[16]] and culling takes place before any rendering. Some examples of object-based methods include Prioritized-Layered Projection algorithms [[16]], Binary Space Partition algorithms, and algorithms using shadow frusta [[14]]. 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 [[11]] and Hierarchical Occlusion Buffer [[29]]. Related methods were proposed by Bartz *et al.* [[2],[3]] and Hey *et al.* [[12],[13]].

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.

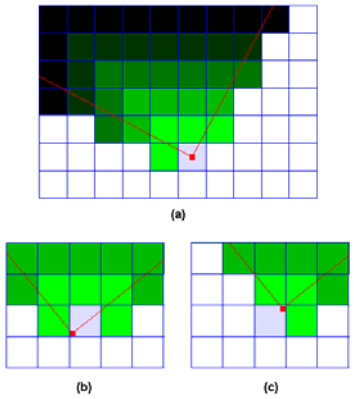

Figure 6: Terrain grid and view frustum

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 blue grid cell contains the viewpoint and non-white grid cells are either intersected or contained by the viewing frustum. There are 8*D* 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 cells: the viewing direction is the same in Figures 6b and 6c but the left-most cells are two indices apart due to different viewing positions.

Visibility Computation

A number of graphics cards implement occlusion queries [[2],[3]] which 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 that 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 be used to approximate visibility.

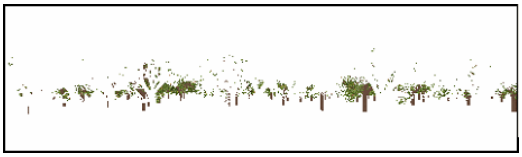

Figure 7: Contribution of the trees in grid cells distance 5 through 10 away from the viewpoint

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 10. 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 the lower-level branches, justifying the hierarchical LOD model. The following framework utilizes 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.

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.

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 proposed 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.

Figure 8: A forest scene at full level of detail

Figure 9: A forest scene using the proposed framework

Figure 10: A forest scene showing levels of detail

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 3. Note that even though discrete LOD results in the highest performance, in practice it exhibits the most evident popping artifacts.

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

Table 3: *Walk-through performance*

In addition to LOD selection, visibility information can be used to efficiently schedule dynamic scene modification. We added forest growing into the walk-through application, allowing the user to adjust the number of productions applied to all trees at run-time. In order to grow a tree by one production, a new linked-list is created by scanning through the existing modules and applying the rewriting rules to *A* modules. The number of modules is exponential in the number of production rules applied, making forest growing a time-intensive task. Rather updating all the trees in the scene first before rendering the next frame, the proposed framework grows each tree (if necessary) only after its visibility is determined. If visibility is below 55%, the tree is grown by one production per frame until the desired age is reached. If visibility is above 55%, the tree is grown to the desired age in one frame. Experiments were conducted by growing a 12-production forest to either 15 or 18 productions. Framerate was averaged over 200 frames of circular motion immediately after the user chose to increase the age. Table 4 lists the framerates and total times (in seconds) for traversals of forests being grown to age 15. Framerate was measured for traversal while the forest was being grown. Timing was repeated over the same path after the trees have reached the desired age. The right-most column lists the times required to grow all 400 (or 1600) trees to 15 productions without any rendering. Table 5 lists same measurements for growing forests from 12 to 18 productions. All traversals use the inertial LOD model. Note that times to grow the entire scene dramatically exceed the total times for 200 frames of combined traversal and growing. Thus, visibility-based dynamic scene-modification results in substantial time savings.

Num. trees	Traverse and grow		Grow then traverse		Grow
	FPS	time	FPS	time	time
400	6.80	29.42	8.59	23.30	45.11
1600	2.92	68.56	6.08	32.89	654.66

Table 4: Forests grown from 12 to 15 productions

Num. trees	Traverse and grow		Grow then traverse		Grow
	FPS	time	FPS	time	time
400	5.07	39.48	6.94	28.81	154.47
1600	1.73	115.95	4.08	49.08	2255.05

Table 5: Forests grown from 12 to 15 productions

5. Discussion and Future Work

An inertial LOD model as well as a visibility-based walk-through framework were proposed in this paper. 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. Forest walk-through was used as a sample application, with the proposed framework achieving speedup up to 20 over view frustum culling alone. Furthermore, the proposed framework used visibility information to dynamically grow the forest at run-time, dramatically increasing performance when compared to the naïve approach of growing the entire forest first. 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] Andujar, C., C. Saona-Vazquez, I. Navazo, P. Brunet. 2000. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum* 19(3), 2000.

[2] D. Bartz, M. Meiβner, T. Hüttner. Extending graphics hardware for occlusion queries in OpenGL. *Proc. of Workshop on Graphics Hardware 98*, 1998, 97-104.

[3] D. Bartz, M. Meiβner, T. Hüttner. OpenGL-assisted occlusion culling for large polygonal models. *Computer and Graphics* 23(5), 1999, 667-679.

[4] N. Chiba, K. Muraoka, A. Doi, J. Hosokawa. Rendering of forest scenery using 3D textures. *The Journal of Visualization and Computer Animation* 8, 1997, 191-199.

[5] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics* 9(3, 2003), 412-431.

[6] S. Coorg, S. Teller. Real-time occlusion culling for models with large occluders. *1997 Symposium on Interactive 3D Graphics*, 1997, 83-90.

[7] P. Decaudin, F. Neyret. Rendering Forest Scenes in Real-Time. *Eurographics Symposium on Rendering*, June, 2004, 93-102.

[8] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems, *Proc. of SIGGRAPH 98*, 1998, 275-286.

[9] O. Deussen, C. Colditz, M. Stamminger, G. Drettakis. Interactive visualization of complex plant ecosystems. *Proc. of IEEE Visualization 02*, 2002, 219-226.

[10] C. Everitt. Interactive order independent transparency. NVidia whitepaper, 2002.

[11] N. Greene, M. Kass, G. Miller. Hierarchical z-buffer visibility. *Proc. of SIGGRAPH 93*, 1993, 231-240.

[12] H. Hey, R. F. Tobler. Lazy occlusion grid culling. *Technical Report TR-186-2-99-09*, Vienna University of Technology, March 1999.

[13] H. Hey, R. F. Tobler, W. Purgathofer. Real-time occlusion culling with a lazy occlusion grid. *Technical Report TR-186-2-01-02*, Vienna University of Technology, January 2001.

[14] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, H. Zhang. Accelerated occlusion culling using shadow frusta. *Proc. of the 13th Annual ACM Symposium on Computational Geometry*, 1997, 1-10.

[15] Jakulin. Interactive vegetation rendering with slicing and blending. *Proc. of Eurographics 2000*, Short Presentations, 2000.

[16] J. T. Klosowski, C. T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics* 7(4), 2001, 365-379.

[17] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002.

[18] D. Marshall. Multiresolution rendering of complex botanical scenes, *Proc. of Graphics Interface 97*, 1997, 96-104.

[19] N. Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Eurographics Workshop on Rendering 96*, 1996, 165-174.

[20] N. Max, O. Deussen, B. Keating. Hierarchical image-based rendering using texture mapping hardware. In *Eurographics Workshop on Rendering 99*, 1999, 57-62.

[21] Meyer, F. Neyeret, P. Poulin. Interactive rendering of trees with shading and shadows. *Proc. of Eurographics Workshop on Rendering*, 2001.

[22] R. Měch, P. Prusinkiewicz. Visual models of plants interacting with their environment, *Proc. of SIGGRAPH 96*, 1996, 397-410.

[23] P. Oppenheimer. Real time design and animation of fractal plants and trees. *Proc. of SIGGRAPH 86*, 1986, 55-64.

[24] P. Prusinkiewicz, M. James, R. Měch. Synthetic topiary, *Proc. of SIGGRAPH 94*, 1994, 351-358.

[25] Remolar, M. Chover, Ó. Belmonte, J. Ribelles, C. Rebollo. Geometric simplification of foliage. *Proc. of Eurographics '02*, 2002, 397-404.

[26] V. K. Sims, J. M. Moshell, C. E. Hughes, J. E. Cotton, 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, J. Xiao. Salient characteristics of virtual trees. *Proceedings of the Human Factors and Ergonomics Society* 45, 2001, 1935-1938.

[28] J. Weber, J. Penn. Creation and rendering of realistic trees. *Proc. of SIGGRAPH 95*, 1995, 119-128.

[29] H. Zhang, D. Manocha, T. Hudson, K. Hoff. Visibility culling using hierarchical occlusion maps. *Proc. of Siggraph 97*, 1997, 77-88.