Real-time Hardware Accelerated Rendering of Forests at Human Scale

Gábor Szijártó
Budapest University of Technology and Economics
Magyar Tudósok Krt. 2
H-1117, Budapest, Hungary
szijarto.gabor@freemail.hu

József Koloszár
Budapest University of Technology and Economics
Magyar Tudósok Krt. 2
H-1117, Budapest, Hungary
kolijoco@yahoo.com

ABSTRACT

One of the major challenges in developing techniques for realistic and high performance visualization of outdoor environments is rendering of vegetation. The issue of rendering dense foliage or forests is especially pressing in simulators and other virtual reality based game applications, such as first person shooters. The greatest problem is that convincing modeling of trees, bushes and undergrowth would require great numbers of polygons beyond what graphics hardware can cope with today or in the next few years to come. This paper presents recent extensions and improvements to the authors' previous work on foliage rendering. It reviews some of the landmark techniques used in real-time PC applications, including the authors' novel solution of using 2.5 dimensional impostors for tree rendering, followed by the in-depth discussion of extending the later algorithm to rendering dense forest for human-scale simulation. The method presented takes full advantage of the programmable rendering pipelines available on most of the recent video cards. Numeric results and pictures are presented to illustrate rendering efficiency and visual quality.

Keywords
Tree Rendering, Forest Rendering, Outdoor Simulation, View Dependent Visualization.

1. INTRODUCTION

Rendering some form of vegetation is a must, when it comes to most outdoor scenes in virtual reality environments. The omission of trees and bushes from the virtual tropic or temperate climate zones drastically reduces the feeling of realism, which is one of the key factors in immersing the user in the virtual experience, a key factor in training simulations and games.

The computer and video game industry has become the driving force behind the most recent advances in rendering real-time virtual environments. User demand for more visually intensive, better looking, more immersing software products, and PC systems required to run these applications, has resulted in the availability of very powerful graphics hardware for the mass market. This trend resulted in mainstream video cards today outperforming graphics workstations, even special architectures, considered high-end just a few years ago.

With the launch of NVIDIA's *GeForce* series of cards, the concept of programmable rendering pipelines has been introduced. Though there are still limitations, the most recent boards from NVIDIA (nv3x chips) and ATI (R3x0 chips) [Ati02] finally offer enough flexibility to make even demanding visualization problems feasible. Rendering problems ranging from volume visualization to mass rendering have been adopted for the new generation of hardware accelerators.

The innovations presented in this paper are intended to help take real-time vegetation rendering to the next level by taking advantage of recent graphics hardware. These algorithms have not been developed for high-fidelity vegetation rendering, but for rendering realistic-looking scenes at frame rates appropriate for real time applications, with special focus and consideration for gaming and simulation engine developments.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

2. VEGETATION RENDERING

First Person Simulators

It is often useful to define specific scales of simulation at which a vegetation-rendering algorithm should provide the required level of realism. Most applications can be assigned to one or more of the following categories: insect-, human-, and vehicle scale [Szi03]. This paper focuses on improving forest rendering at human scale for first person simulations.

Image 1: Textured tree models in EPIC's Unreal Tournament 2003 first person shooter.

The first moderate successes in realistic outdoor simulation at human scale emerged only a few years ago. The golden age of the first-person shooter – probably the most popular style of computer game today – still hasn't come to an end. Throughout the ten years of the genre's evolution, outdoor environments with lush vegetations have either been lacking, or have made a rather poor impression. Almost all the popular titles had indoor and urban or industrial settings. Arguably some of the best results where the earliest: Novalogic's voxel based engine, which has since been abandoned in favor of accelerated iterative rendering, being one of the select few examples worth mentioning.

Most graphics engines use static, multi-layered aligned billboards. Though in some rare cases these algorithms yield satisfactory illusions, they are far from what can be potentially achieved with recent hardware.

At the time of this writing the most pleasing visual quality in commercial entertainment software is achieved through the use of basic free-form textured tree models (Image 1) with some Level of Detail (*LoD*) applied. Though the idea is quite straightforward, only in the last few years has hardware become powerful enough to handle the task. Resulting visuals are satisfactory when the trees are further away from the camera. However, due to the simple geometric model used, close-up views look artificial. Also, variations are usually introduced through new models (or through combining model-parts). Thus, increasing the number of trees in a scene quickly bogs performance. The aim of this research has been to develop a core tree rendering algorithm for human and vehicle scale simulation, with possible application in low-altitude (helicopter and glider) flight, land vehicle simulators, and first person games. This paper focuses on techniques that can be used to adopt the algorithm for the latter, human scale simulation of forest environments.

Scientific Applications

Vegetation rendering has long enjoyed its share of scientific interest. The two broad fields associated with the topic are the generation of plants, and their visualization. The former – vegetation modeling – lies outside the scope of this study. Extensive research in the field has yielded a number of publications, and commercial tools for modeling are also available [Pru90].

Visualization seems to be a nut harder to crack. There are two general approaches: geometry-, and image-based methods. As its name suggests, techniques of the former group use geometric representations of the foliage. As it takes roughly hundred thousand triangles to build a convincing model of a single tree, some form of LoD rendering technique must be applied to reduce the polygon count for a given frame to a reasonable level [Pup97][Rem03]. Visually pleasing results can usually only be achieved with rather complex algorithms, or significant memory overheads. The authors suggest that for purposes of games and simulation, geometry based methods are not yet efficient enough.

3. IMAGE-BASED TECHNIQUES

Image-based methods, which mask oversimplified and primitive geometry with various canvases in image space, represent a trade-off of consistency and physical precision in favor of more photo realistic visuals. Because of the inconsistency in literature regarding the exact definitions and usage of the image-based rendering related terms *sprite*, *billboard* and *impostor*, a clarification in the context of this paper follows.

Sprite: a flat face with some static surface properties (usually a simple texture image) always facing the camera from a fixed position in space (Figure 1a)

Billboard: a face that rotates around one fixed axis in space, trying to face that camera as much as possible (Figure 1b).

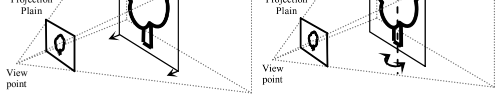


Figure 1a: Sprite Rendering. The textured polygon is always facing the camera. 1b: Billboard with arbitrary position and orientation.

Impostor: a billboard or sprite, whose surface properties are not static. The texture applied is also rendered to at some earlier stage.

In the context of tree-rendering, sprites are analogous to using cardboard cutouts with a tree-like image painted on them, which are always facing the camera. Though resulting visuals are far from satisfactory, this obsolete technique is used to this day to depict smaller plants or plant parts.

Billboards can be regarded as the same cardboard cutouts rotating to face the avatar around an axis dug into the ground. Billboards are the most frequented method for rendering vegetation in many land-based vehicle simulations.

Sets of view-dependent sprites, and complex cutouts are obvious improvements to basic sprites and billboards. The former method simply pre-renders a finite set (usually 4 or 8) of views of the same tree, and presents the one closest in alignment with the actual viewing direction. A popping artifact is visible when there is an alignment change. The latter algorithm uses texture transparency and blending to render more than one view at the same time onto properly aligned surfaces. Both methods fail to deliver quality in close-up views.

Though more advanced techniques using billboarding [Max96], layered depth-images [Sch98], and multi z-buffers [Max96] have been presented through various scientific forums, most are computationally too expensive for implementation in games or simulators. It is important to stress that even though the quality of tree- and foliage rendering makes a huge impact on overall visuals, in an actual product it is but a small part of a complex rendering engine, and must share the available hardware resources with other computation-intensive tasks.

Advantages

The abovementioned techniques all take advantage of the fact that is much faster to render a recorded image of a tree, than to actually process the geometry information describing a tree model. There are a number of reasons for this, the two most important ones being:

- A leaf on a tree only a short distance from the camera is mapped to barely more than one pixel. On today's hardware, rendering a single pixel is much faster than transforming even a single triangle.
- The number of obscured leaves is very large, thus a significant number of transformations would be performed in vain.

Disadvantages

Image based methods suffer from two significant drawbacks: fixed perspective and invariance to motion and rotation.

Luckily, in tree rendering, the fixed perspective is not so disturbing. The tree canopy is a fairly irregular structure, and the human eye is far less sensitive to perspective distortions of irregular shapes than regular ones.

Invariance to motion and rotation is far more disturbing. Leaves are static as the camera moves, while it would be expected for some leaves to appear and others to become obscured by the displacement. This issue has to be addressed in some way to raise rendering quality to an acceptable level. In applications where the camera is usually level with the trees, the trivial idea to store multiple textures corresponding to rotations around the vertical axis come to mind. However, to achieve desired quality and eliminate popping artifacts when changing textures, an unacceptably large number of textures would have to be stored. Popping artifacts are about as disturbing to the human eye as static textures in motion.

From the above it is concluded that to achieve convincing visuals, the geometry of leaves has to be processed to some extent.

4. USING 2.5 DIMENSIONAL IMPOSTORS

In their previous works the authors presented their own image based algorithm for tree rendering. The technique of 2.D dimensional impostors composes the tree canopy of several smaller leaf clouds. A leaf cloud only consists of a fraction of total leaves needed to model the entire tree. The amount of geometry describing a cloud is small enough to handle on a per frame basis. The concept is to process a leaf cloud, render it to a texture, and apply that texture multiple times to render the tree.

Impostor rendering has two stages. The first stage is a view-dependent render-to-texture operation (drawing the impostor), the result of which is used in the second stage usually as sprite or billboard texture.

An arbitrary group of leaves arranged randomly in space and assumed to be positioned in the center of mass of the canopy is rendered to a texture with correct alignment (Figure 2). The texture is then used as an impostor and applied to more sprites to build the canopy. Image quality and feeling of realism increase dramatically, even if the same impostor is used for the entire tree (Image 2).

Depth is introduced to the render-to-texture procedure. In the stage of impostor rendering, depth information is stored in the target textures alpha channel. The result is a 2.5 dimensional impostor. In the implementation the group of leaves rendered is assumed to be at some center point of the canopy, just as before, but the z-near and z-far planes are adjusted to approximate a reasonable bounding box for the rendered group of leaves, as shown on Figure 2. In the final phase of rendering the stored depth values are appropriately scaled and clipped to the final depth buffer before depth testing is performed, yielding a volumetric feel to the textured sprites, which can now inter-lap in a spatially coherent manner.

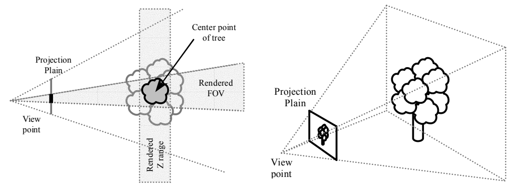


Figure 2: Rendering the impostor texture from a group of leaves assumed to be positioned at the center point of the tree.

The method successfully introduces motion dependence, and by adding depth information (the 0.5 dimension) to the impostors, adds volumetric effect to overlapping textures eliminating virtually all motion artifacts. Depth-consistent impostor rendering results in images where leaf-clouds can correctly overlap with each other, branch geometry, other trees, or any other object in the scene.

Also note that the artificial look resulting from repetition of the same image over many sprites is almost completely eliminated, as volumetric overlapping obscures arrangement to the point where it is almost impossible to discern any single impostor.

Single Tree Implementation

The algorithm proposed above produces very convincing visuals, and can be efficiently implemented using only the GPU on recent video cards (Figure 3).

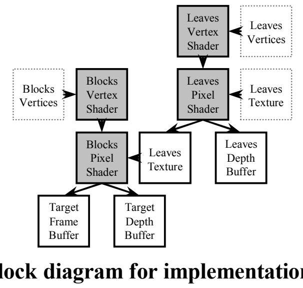


Figure 3: Block diagram for implementation using two vertex, and two vertex programs. Depth information is stored in the alpha channel of Leaves Texture.

The vertex program used in the first stage does nothing beyond the ordinary except for the adjustments to the projection matrix. Thus the center of the tree will become the center of the image, with the aspect ratio matching that of the desired final image. The depth information is stored in the alpha channel of the impostor texture, enabling easy access to it by the second stage.

The vertex shader program in the second stage is used to calculate the correct projected size, position and depth offset of the impostor sprite. The pixel program is responsible for rendering the impostor texture in a depth consistent manner by adding the value from the impostor texture's alpha channel to the depth offset passed by the vertex shader.

Image 2: Branches and leaves with appropriate depth values.

Note that the pixel program to render the blocks requires the very latest hardware to date, as only pixel shaders of version 2.0 and above support direct write access to the depth buffer. Current graphics chipsets supporting this functionality include NVIDIA's GeForceFX 5x00, and ATI's Radeon 9x00 series.

Forest Rendering

This section focuses on forest rendering at the human scale first person simulation (FPS), when the avatar is walking in the forest. As opposed to low altitude flight simulation or some other vehicle scale configurations, where there are many trees visible all being roughly same (small) size, in FPS a few trees close to the camera must be rendered in very high detail while those further away become smaller in projected size but grow in their numbers. Traditional techniques fail to deliver the performance for close-ups, but the 2.5D impostor is perfectly applicable. Because the numerous trees in the distance must be rendered at great speeds, simpler techniques have to be considered. Mixing rendering techniques for any given task introduces the inherent danger of popping artifacts.

The 2.5D impostor is well suited for the task of visualizing distant trees as well as close-ups. As rendering the canopy is a question of filling pixels, performance is inversely proportional to the projected size of textures. This is confirmed to some degree by measurements presented in the next section. Because of this, LoD for the canopy is not a pressing issue. This is fortunate, as geometric LoD techniques usually introduce noticeable morphing or popping on level switches. The 2.D impostor algorithm does not have this drawback. Tree trunks and branches, however, are still rendered from richly detailed polygonal models, so some form of LoD should be applied to them in order to increase performance.

Image 3: Forest.

The brute-force approach is to render individual trees in a loop. However, tests indicate that doing so quickly overtaxes even the most recent graphics hardware. Some form of visibility testing to screen the occluded trees and those outside the view frustum prior to rendering can dramatically increase performance. Inclosing spheres have been implemented for safe frustum culling.

One way to save calculations is not render an impostor foe every individual tree, but to apply the same impostor texture for all trees with similar parameters and orientation. However doing so reduces variations, degrading visual quality. Tests were performed, with results indicating that performance gain is minimal as performance is limited by the stage tiling the impostor textures over the trees in the scene. As a result this technique has been abandoned.

While rendering a single tree is extremely fast on recent graphics hardware, it is still short of what could be achieved with some technological modifications [Szi03]. Writing the depth buffer explicitly from the pixel shader is relatively slow (and even discouraged by developer documentation from hardware manufacturers). Also for pixel programs doing so, the depth check is performed after execution of the code. This means that only shader has accessed all associated textures and performed its sometimes expensive set of calculations does the architecture decide whether or not the pixel is displayed at all. While this is logical, some support for “early elimination” based on minimum z-offset, or more generally, any parameter passed directly from the vertex processing stage would be welcome, as complex shaders would not have to be executed for occluded pixels. In case of the 2.5D impostors the z-offset (the nearest possible projected position the pixel could occupy before the depth value from the impostor texture’s alpha channel is added) could be used to discard a significant portion of calculations for a scene. In dense forests, where sometimes entire trees have to be rendered even though only a few of their leaves are visible through occluding canopies, early elimination would have an even greater positive impact. Estimates based on test measurements indicate a five- to tenfold potential increase in rendering speeds for an average forest scene. Other tests showed that similarly the stencil test is also performed after shader program execution.

As already mentioned, test results indicate the performance bottleneck to be rendering the impostor textures to the frame buffer (Blocks Pixel Shader, refer to Figure 3). The idea is to simplify this stage for distant, obscured, and almost occluded canopies. We implemented a pixel program that did not write the depth buffer, but rather used simple alpha blending in rendering the impostors, thus giving up depth consistency for overlapping leaf clouds. Doing so introduces a popping artifact, when one texture “jumps” ahead of another as a canopy is rotated. However, qualitative tests indicate that the artifact is indistinguishable for trees sufficiently far away from the viewer. On current hardware such trees can be rendered up to five times faster using alpha blending instead of 2.5D impostors, with almost no impact on overall visual quality (See Results).

In order to achieve top visual quality for close-ups, highly detailed geometry was used to model tree trunks, branches and even leaves. For distant trees maintaining the same detail is more than a luxury. In typical FPS scenes most probably only the trunks of distant trees are visible. For this reason it is reasonable not to use optimization based LoD techniques, but rather simply to decimate smaller branches for lower detail levels (cut them off as the camera withdraws). Doing so elements of geometry are discarded that are not visible anyway, and no popping or morphing artifacts are introduced. For distant trees, it is sufficient to use a single triangle to model a leaf, as projected leaf size nears sub-pixel proportions.

Forest Implementation

Instead of simply extending the algorithm used for single tree rendering, all stages of processing and rendering were revised with special consideration to hardware capabilities.

The first phase is sorting. This is necessary, because using alpha blending for textures on distant trees requires a back-to-front rendering order. The vertex and pixel shaders for the first stage of tree rendering (Leaves Vertex Shader and Leaves Pixel Shader, refer to Figure 3) are almost identical for the alpha, and 2.5D techniques. The only difference being that while the 2.5D version writes a depth value to the alpha channel, the alpha version simply passes a constant 1.0 for target pixels with leaves projected onto them. Because of the negligible difference, execution speed for the two versions of the first stage shaders is almost identical.

The second phase is impostor generation. Switching between render targets is a costly operation in terms of performance. Thus, instead of rendering on a per tree basis with one switch for every tree in the visible scene, all impostors are rendered to a single large render target capable of holding 16x16 different impostor images, which, as qualitative testing has shown is enough for practical purposes.

All "other" objects, such as trunks and branches are rendered in the third phase.

Finally, in the fourth phase, the impostor blocks are rendered. First high detail 2.5D impostors for trees close to the camera in a front-to-back order. Next the remaining low detail alpha blended impostors are rendered back-to-front.

Note that in a pure 2.5D impostor implementation, the last two phases would also be interchangeable.

5. RESULTS

The two leading hardware manufacturers, ATI and NVIDIA both offer fairly solid support for OpenGL and DirectX. For the research involved in this paper, Microsoft DirectX 9.0 [Mic02] has been the platform of choice. DirectX supports accelerated hardware through various versions of pixel and vertex program profiles. Programs or shaders are written in an assembly-like language. NVIDIA introduced its C for Graphics (Cg) [Nvi02] programming language to provide developers with easy to use C-like syntax to develop advanced shaders. The Cg toolkit available free of charge from NVIDIA's homepage provides tools to compile Cg code to the DirectX shader language, or the NVIDIA specific register-combiner extensions for OpenGL. For this research vertex and pixel programs were developed in Cg, compiled to DirectX shaders, which were fine-tuned by hand when modifications were deemed necessary.

Tests were run on an AMD Athlon XP 2600+ based system with an ATI Radeon 9700Pro (R300) video card. The application's sensitivity to processor speed was not significant, as all geometry and object data were static and uploaded to the video card, and the video accelerator was able to perform all calculations on-board, reducing system processor load to a minimum.

A highly detailed tree model was used. Leaves were modeled using 46 polygons each, the branches and the tree trunk were built from 19834 faces. 70 impostor textures were rendering to every tree, and there were 512 leaves in every impostor. The corresponding polygon count for one tree thus multiplies up to over 1.5 million polygons (35,000 leaves). Using the 2.5D impostor technique however, barely 40,000 polygons were actually processed during rendering. The low detail tree model uses 390 polygons to model the trunk and branches, while 4 polygons were used to model a single leaf. The corresponding polygon count thus equals about 143,750 polygons. The default resolution of the test application (Image 4.) was 512x512, and impostors were rendered to 128x128 textures. Test results were obtained by omitting branch and trunk rendering, as those are handled independently from the algorithm.

Qualitative testing was not overly rigorous: a few people, some of whom were computer graphics specialists, were asked their opinion on visual quality and comparisons.

Single Tree Results

The first tests were run to show the performance relative to pure geometry based rendering. Of course, the results are hardware dependent, as pure geometry based rendering tends to be vertex shader limited, while the impostor-based algorithm is limited by the pixel shader. The ATI R3X0 chipset used, has four vertex shading units, which also boosted the highest performance on the market at the time of writing, thus the pure geometry results may safely be regarded as „best-case" values among video cards.

	1024	512	256	128
🟦 2.5D impostor	391	455	501	525
🟪 pure geometry	26	51	100	194

Table 1: Pure geometry vs. 2.5D impostor rendering without branches. 512x512 render target resolution.

Withdrawal from a tree results in decreasing the number of the pixels rendered to. This has no impact on the sensitivity of the pure geometry algorithm. Comparing tables 1 and 2 we find that rendering the same number triangles to four times the number of pixels barely impacts the performance of the 2.5D impostor algorithm. It was also found that the rendering speed of the impostor algorithm is about linearly sensitive to the image size rendered to. This characteristic has already been mentioned in Section 3.5. It can be concluded that that rendering the leaves to the impostor texture is not contributing significantly to rendering in case of the model used.

	100%	75%	50%	25%
🟦 1024 leaves	391	519	656	813
🟥 512 leaves	455	644	750	1170
🟨 128 leaves	525	785	1150	1736
⬜ 1 leaf	554	850	1261	1999

Table 2: "Withdrawal" test rendering to 512x512 render target.

Forest Results

Image 4: 100 Trees in the scene.

Tree parameters were configured based on results from the previous subsection: 512 leaves per impostor, 70 impostors per tree. A forest of 1024 trees was generated. As mentioned in the Forest Implementation Subsection, the number of different impostors (and thus the number of trees) for a scene is limited to 256. As illustrated by Image 4, this number large enough to ensure that the forest does not "end" after about 100 meters, but seems convincingly deep. In fact, qualitative testing indicates that even a limit of 100 impostors would yield acceptable visuals. Tests were run using the 256 limit nonetheless.

The next issue explored was the compromise between quality and performance by limiting the number of trees closest to the camera to be rendered using high resolution models and 2.5D impostors, while low resolution models and alpha blended impostors were to be used for remaining ones. Again, qualitative testing indicated that rendering low detail trees, after the first 15-20 high detail ones does not impact visual quality when compared to a pure high detail image (Refer to Image 5).

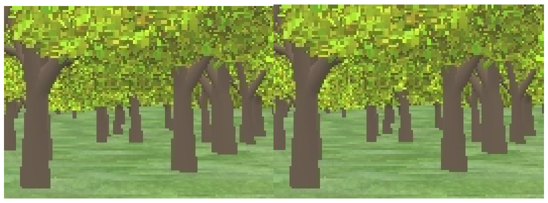
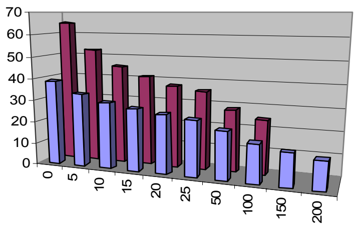


Image 5: Comparison. All the trees in the left picture, but only the twenty closest trees in the right one are rendered in high quality.

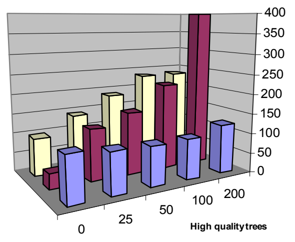
We explored the impact of the number high quality trees in a scenes consisting of 100 and 200 trees respectively. As show on Table 2, rendering only the first 25 trees in high quality doubles performance compared to the pure high quality scene barely impacting visual quality. Also note that rendering all low quality trees only boosts an additional 50% relative to the 25 mark while degrading visuals to an unacceptable level. The chart also illustrates that performance is not inversely proportional to the total number of trees, because omitted distant trees are represented by much smaller textures than those drawn in close-up in either case.



	0	5	10	15	20	25	50	100	150	200
🟦 200 trees	38,8	33,5	30,2	28,4	26,7	25,4	21,7	17,4	14,7	12,8
🟥 100 trees	65	52,6	45,3	41,1	37,4	35,6	28,2	24,5		

Table 2: Frame rates for rendering a total of 100 or 200 trees respectively, with the first 0, 5, 10, etc. drawn in high detail.

Finally the contribution of the individual rendering phases to overall performance was evaluated. Computational times for the relevant phases were measured on the scene of 200 visible trees. Note that precise timing is near impossible as the graphics hardware operates independently from the processor, and the processor is only responsible for running driver components during rendering, which optimally do little more than manage data traffic between processor and GPU. In case of the tested algorithms, the processor is only subjected to the bare minimum of computations. The program was tweaked to wait for the GPU, thus enabling some time measurements without introducing a significant performance lag.



	0	25	50	100	200
🟦 Textures	114	104	101	101	124
🟥 Impostors	39	130	158	220	400
🟨 Objects	97	142	186	233	233

Table 3: Computation times in ms for rendering phases 2 to 4. Refer to Forest Implementation Subsection.

As expected, the second phase (impostor generation, Textures), is not sensitive to the number of high resolution trees, as shaders in that stage are almost identical for both detail modes (see Forest Implementation Subsection).

In the pure high quality case, impostor rendering is limiting performance, while in the pure low quality case, object rendering and impostor generation are the bottlenecks.

It is deducted the object geometry detail could be reduced further to improve performance, but this might result in degrading visuals, so the authors advise against it. Note that current game applications use much less detailed models for foliage. Another viable option is reducing the number of trees. Doing so results in a linear increase in object rendering and impostor generation, however impostor rendering speeds are improved only slightly.

Measurements confirm that modifications to hardware as suggested in the Forest Rendering Subsection would have a dramatic impact on performance. A tenfold boost on rendering speed is estimated. With the added feature the high quality 2.5D impostor could be implemented even more efficiently than the low detail alpha blending method. The same frame rates achieved in all low quality cases could be achieved, possibly even surpassed using all high quality canopies.

6. CONCLUSIONS

This paper presented the application of 2.5D impostor rendering to forest rendering. The technique renders high detail foliage without popping/morphing artifacts associated with geometry based LoD methods, and without artifacts upsetting the motion parallax, inherent in basic image based methods. A first person simulation of a convincing forest can be rendered at 25 frames per second, with further room for hardware and application specific optimization and fine-tuning.

Image 6: Low altitude flight.

As proposed acceleration techniques only exploit the fact, that trees are densely packed and obscure each other to some degree, the technique presented also performs formidably for low altitude simulations.

If the technique is to be extended to high altitude simulation as well, another approach must be considered as projected size of impostors may become as low as single pixels, resulting in noisy canopies when the camera moves. Also impostor generation is expected to become the performance bottleneck. A smooth transition from low to high altitude must also be implemented.

Future work will focus on introducing more variation to vegetation, conifer rendering, and engine integration.

7. ACKNOWLEDGEMENT

This work has been supported by the National Scientific Research Fund (OTKA ref. No.: T042735), IKTA ref. No.: 00159/2002, the Bolyai Scholarship and the Slovene-Hungarian Fund.

8. REFERENCES

[ATI02] ATI Technologies Inc., RADEON™ 9700 Pipeline Overview, 2002, www.ati.com

[Lin99] Lintermann, B., Deussen, O.. Interactive modeling of plants. IEEE Computer Graphics and Applications, 19(1), 1999.

[Max96] Max, N., Ohsaki, K.. Rendering trees from precomputed Z-buffer views, Eurographics Workshop on Rendering 1996, pp. 165-174, 1996.

[Mic02] Microsoft Corporation, DirectX 9.0, 2002

[Nvi02] NVIDIA Corporation, Cg Language Toolkit, 2002

[Pru90] Prusinkiewicz, P., Lindenmayer, A., The algorithmic beauty of plants, New York, Ed. Springer-Verlag, 1990.

[Pup97] Puppo, E., Scopigno, R., Simplification, LOD and Multiresolution – Principles and Applications, Eurographics'97, Tutorial Notes, 1997.

[Rem03] Remolar, I., Chover, M., Ribelles, J., Belmonte, Ó., View-Dependent Multiresolution Model For Foliage, WSCG 2003, pp.370-378, 2003.

[Sch98] Schaufler, G., Per-object image warping with layered impostors. Eurographics Rendering Workshop 1998, pp. 145-156, 1998

[Szk95] Szirmay-Kalos, L. Theory of Three-Dimensional Computer Graphics, Publishing House of the Hungarian Academy of Sciences, 1995.

[Szi03] Szijártó, G., József, K., High Resolution Folaige Rendering for Real-time Applications, SCCG 2003, Budmerice, Slovak Republic