An Automated Modeling Method For Multiple Detail Levels Of Real-time Trees

Charalampos Koniaris
Affiliation
VR Department, FHW
Athens, Greece
Charalampos.Koniaris@disney.com

Athanasios Gaitatzes
VR Department, FHW
Athens, Greece
Gaitat@fhw.gr

Georgios Papaioannou
Athens University of Economics & Business
Athens, Greece
Gepap@aueb.gr

Abstract— **Rendering realistic outdoor scenes in real-time applications is a difficult task to accomplish since the geometric complexity of the objects, and most notably of trees, is too high for current hardware to handle efficiently in large amounts. Our method generates trees with self-similarity, and later exploits this property by heavily sharing pre-rendered textures of similar parts of the tree. The intrinsic tree hierarchy of the trees, combined with their self-similarity, allows generation of multiple levels of detail. Here we present the flow of the processing stage, from the collection of the required input data until the export of the models in all their levels of detail as well as related and additional data.**

Keywords : tree; modeling; foliage; rendering; image-based;

I. INTRODUCTION

Since the beginning of real-time 3D graphics in consumer applications, representation of convincing outdoor environments has been a difficult task. That is because of the inherent need of rendering a large number of complex objects on screen, since outdoor environments are rarely empty or barren. One of the most complex objects to be rendered in real-time in an outdoor natural environment is a tree. Trees usually have extremely high geometric complexity and many techniques and tricks have been used over the years to represent them in a visually appealing way. In this paper we present a new technique where we are able to model a large number of different tree types, given their parameters and base textures, and generate models composed of meshes and shared textures for various levels of detail.

II. PREVIOUS WORK

Much work has been done in the field of both modeling and rendering trees. In the field of modeling, Prusinkiewicz et al. in [11] introduced the capability of L-systems to create botanically correct trees using sets of rules. Other methods for modeling are the work of Weber et al. in [16], which has branches and leaves as components and models only trees, and the work of Lintermann et al. in [7], in which one can interactively model any plant type. All mentioned modeling methods create trees of high geometric complexity. In the field of rendering, many techniques have been proposed over the years, both for real-time and offline rendering. The techniques of interest for the current paper are for real-time rendering and can be categorized into image based, geometry based, and techniques using points and lines or volumetric textures.

An interesting technique for visualizing forests proposed by Decaudin et al. in [1], uses volumetric textures of pre-rendered tree clusters, tiles them aperiodically, and applies two different rendering techniques, one for silhouettes and one for an above view of the forest, using slices. Two major drawbacks of this technique are the static lighting and the large amount of texture memory it uses. Another technique by Deussen et al. in [2] uses points and lines for the rendering of lower importance geometry in a scene, but current hardware favors triangle rendering.

Many multi-resolution techniques Rebollo et al. in [12],[13], Hidalgo et al. in [4] and Remolar in [14] have been proposed based on the Foliage Simplification Algorithm (FSA), which uses leaf collapse (two leaves forming a new one which contains both) as a triangle reduction method. The techniques vary mainly in how they select the leaves to collapse. The method of Hidalgo et al. [4] also needs a texture artist to compose a texture atlas of the foliage.

Other techniques focus on more effective approaches for the reduction of geometry by being heavily image-based. Jakulin in [5] presented a method using blended slices, in which the tree foliage is rendered at various angles and depths, and then the resulting images are applied on properly placed quadrilaterals, giving a volumetric feel. The drawbacks of the method are that the planarity of the foliage is significantly visible when viewed from

above, and that for his models, it does not provide levels of detail.

Another technique by Meyer et al. in [9] generates trees using L-systems to exploit the instancing of similar geometry, for interactively rendering trees with shading and shadowing. Lluch et al. in [8] proposed a method where each branching level of a tree is independently rendered by three different views and stored in textures. This method's drawback is it's large footprint in texture memory, especially for complex trees.

Szijarto et al. in [15] proposed the use of 2.5 dimensional impostors, storing the depth information of the rendered textures' actual geometry in their alpha value and processing it later by shaders, and placing randomly the leaf clouds on the canopy. Garcia et al. in [3] proposed the use of impostors and indirect texturing, maintaining the parallax effect when changing view angles. Lacewell et al. in [6] propose the use of billboard clouds, which need large textures to effectively represent trees with very low triangle count.

III. Algorithm Overview

Our algorithm is mostly image-based and relies heavily on its novelty, which is texture sharing, and preprocessing (Figure 1). The main new idea is that, given a carefully crafted (i.e. not completely randomized) tree hierarchy, we can perform render-to-texture to a node & children of a given level, and use/reuse the result in all the nodes in that level, performing the appropriate transformation.

The trees' generation is based on parameters which are given in a per-branching level basis. A branching level is considered the collection of all branches or leaves that have parents belonging to the same level, with the trunk being the first level. The geometry for each branch is not expressed in a global coordinate system but rather a local transformation is maintained, as we later need to replace the detailed geometry with textured impostors depending on the desired level of detail. The pre-rendered textures can represent either a single branching level (a single node in the tree hierarchy), or a hierarchy of branching levels and are created offline by a high resolution model of the tree. The branches and trunk are created using as basis multiple LODs of spline representations, which are also created in the preprocessing phase. The generator can use constrained randomization in the parameters, resulting in a variety of created trees. Also, LODs can be automatically generated for such trees, given a maximum triangle count, using a set of rules of simplification. Finally, soft shadow maps can be computed offline for later use using projective texturing. All the above steps are done in the preprocessing phase, which results in minimal computational and storage use at runtime.

IV. Description - Use

The main idea of the technique is to replace parts of a tree's geometry with shared pre-rendered textures. With a well-defined tree structure, all nodes of a branching level can be replaced by double the number of quadrilaterals (cross-quads), each couple textured with two pre-rendered views of any node and sub-tree of the branching level. This way, LOD levels can be created by gradual replacement of sub-trees with single nodes, each composed of two pre-rendered textured quadrilaterals. Besides sub-tree replacement, branches (single nodes in hierarchy) can also be replaced by quadrilaterals textured with a pre-rendered view of a single branch, giving a greater control to LOD level tuning. Also, branches, loaded from files containing spline data, can be assigned a varied simplified representation of the actual spline, with varied resolution of the branches' circular perimeter shape. This gives even finer control over the creation of LOD levels.

A. *Input Data*

An example application can use a set of input data for the generation of one or more trees and tree types. These data are ASCII spline data files, containing geometric information of a curve for an arbitrary number of detail levels, one or more textures for the bark of a tree, one or more textures for the leaves, and finally script files, containing the sets of parameters for the creation of the tree hierarchy, the texture and the spline data file it uses. Trees can be assigned an arbitrary number of leaf types, which can be distributed randomly, but with controlled weight, on a branch.

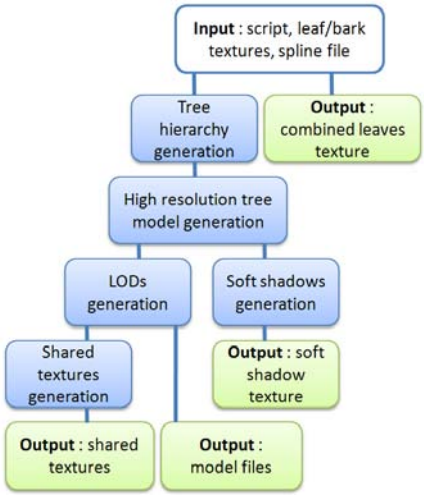

Figure 1. Flow of pre-processing (white = input data, green = output data, light blue = intermediate steps).

Some of the parameters that can be used for the generation of a level in a tree are x, y and z rotations (in the local coordinate system of parent nodes), scaling, radius, number of child links per branch, number of created children per child link, function of length, radius and parent branch length percentages, start and end positions, node type, spline used (if not leaf).

B. Spline LOD Creation

For generating multiple levels of detail for branches and trunk, a method for creating polygonal representations of splines will be used. A type of splines that is convenient for this use is Catmull-Rom splines. The conversion of the splines to a collection of points can be automated, given as parameter the error tolerance of the converted spline. A method that can be used to make the conversion is to compute and add iteratively the maximum distance points to the initial set of control points. An output set of points will be formed by the initial set plus a selection of evaluated points. Each evaluated point whose distance from the line formed by the two points of the output set which enclose it, is maximal, is added and sorted to the set (maximum distance point). Points are added until the maximum distance is less than the appointed error bound. The iterative phase is reflected in the algorithm in table 1. The representations created this way, for the number of points they use, approximate best the actual spline (Figure 2).

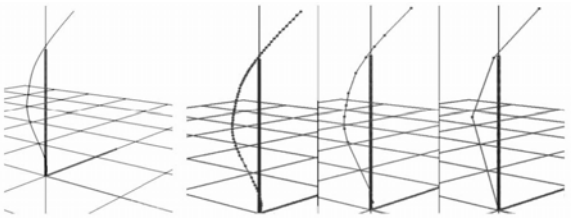

Figure 2. Original spline and three of its' simplified line representations

C. Tree hierarchy generation

Initially, a high resolution model of the tree needs to be created, in order to be rendered to textures for use in the lower resolution models. The method used for the generation of trees is a simplified component - based method, where two types of components can be used, branches and leaves.

A tree is considered as a set of branching levels, each resulting from a previous one, and leading to another. The exceptions are the root level (trunk), which has no parental node, and the leaf level (leaves), which has no child nodes. All internal nodes produce an arbitrary number of children (Figure 3). A transformation is applied to every node in a level, which is the combination of the transformation of its parental node, and the transformation that corresponds to its index in its parent's child-node list and to the set of parameters that define the level in which it belongs. At this stage, randomization in the parameters can occur, but is restrictive to a subset of the nodes in each level - the nodes of a single chosen parental node. Because of the grouping that needs to be done for the rendering of each level & sub-levels, any two instances of a level (containing branch, sub-branches and/or leaves) must be completely identical, so they can later share the same texture.

D. Render to texture

This step requires a high resolution model of a tree, in order to render, parts of it or whole, to textures. For the renderings, an orthographic projection is needed (to avoid perspective distortion), with the frustum having same dimensions as two of the extents of the axis aligned bounding box of the part to be rendered, thus resulting in an optimal sized texture. For each rendering, any node can be used, since by applying its inverse combined matrix, it is relocated to the origin. An issue which arises is the variable thickness of the branch in different levels, which can be partially compensated by scaling the quadrilateral's width on which the output texture should be applied. Due to the fact that all sub-trees of a node share the same internal branch transformations (identical shape), their pre-rendered version can be used as a texture on quadrilaterals which can replace all nodes on the same level. The nodes should be rendered with generally neutral lighting conditions, since lighting can be correct only for the sample node that was rendered, if the light direction is transformed along with node's inverse transformation matrix. Also one rendering can be used for both faces of a quadrilateral, since the usually high similarity doesn't justify the doubled cost in texture memory.

TABLE I. SPLINE REDUCTION ALGORITHM

Initialize output set with spline's control points

While error tolerance is lesser than the maximum distance point

- *Search between every set nodes in the output set for a local maximum distance point*

• *If the global maximum distance is greater than the error tolerance,*

• *Store the maximum distance point, sorted, in the output set*

End while

Figure 3. Example texture with concatenated renders.

Since geometry will be replaced by rendered quadrilaterals, one issue that arises is the planarity of the quadrilaterals that will be very noticeable from certain points of view (when view vector is almost perpendicular to the quadrilateral's normal). This can be improved by rendering into two quadrilaterals, from perpendicular angles (cross-billboards). In this way, the planarity is reduced, but another issue arises. If the curvature of the local root branch is high, parts of the branch deviate enough from its main axis to make it look doubled, when the two rendered textures are applied. This can be further avoided by rendering in the second pass all elements of the first, except of the local root branch. So, for a *n*-level tree, having *k* different branch types, a complete LOD representation would require *(n-1+ k)*2* number of renderings. The viewing boxes have dimensions the x, y & z, y couples of extents of the axis aligned bounding box of any, relocated to the origin, node rendered.

E. LOD information generation

Multiple levels of detail can be achieved with combinations of three types of simplification, mentioned in order of significance/triangle count reduction. They are: reduction from multiple geometric tree levels to textured cross-billboards, reduction from branch geometry to textured quadrilateral and simplification of branch geometry.

A relatively simple system can be constructed to automatically select appropriate detail levels for a given limit on triangle count. Instead of computing the triangle count for all possible combinations, a priority-based approach can be taken. The information needed is: triangle counts for every unique branch type in all of its LODs, number of tree levels and number of children per node. The algorithm begins from the highest resolution model (geometric branches at their highest LOD and textured quadrilateral leaves) and starts simplifying parts till the total triangle count runs below the appointed limit. The priorities that apply to this simplification method are essentially the order in which the various simplifications occur. Also for visual integrity, every LOD level computation can have as starting input detail level the previous computed LOD. With this change, wild deviations of branch LODs can be avoided and no LOD can be the same with another one, since it starts immediately simplifying. Also, two simplifications for the LOD of geometry of the branches can be embedded into the algorithm. First, the closer a level of a branch is to the root level, the larger and more detailed it will be. With this in mind, the LOD generator can select only one

LOD for the closest to the root geometrically represented branch available, and then decrement linearly the detail levels of the splines of all other levels containing geometric representations of branches. Second, the last level in which a geometric representation of a branch is used is usually the most occluded by its surrounding foliage. So, the generator can select for these levels always the lowest resolution geometric LOD available. This effectively reduces the number of rendered triangles, preserving the volume of the branches. Figure 5 shows multiple levels of detail of a generated tree while an example of LOD generator priorities is illustrated in Figure 4.

F. Soft shadow maps

Since high-resolution models are initially required and computations are done offline, it is possible to compute and store several baked soft shadow maps for later use. The pre-computed shadow maps are best used for directional light, since the changes are more subtle than a positional one, and the distance of the light source doesn't matter. A solution to the problem of covering all the potential shadow forms is to compute them at specific equally distributed intervals and fade between them. For an acceptable degree of realism, nine light directions could be used. One (vertical) should be directed towards the negative y-axis and the rest (planar) could form a 45 degree angle with x-z plane and be distributed equally on the appropriate horizontal ring on the surface of the unit sphere. In this way, according to the light's direction, one to three soft shadow textures can be combined (vertical and two planar). The soft shadows can be created by sampling points on a circle 'wrapped' on the unit sphere, with the point on the unit sphere which intersects with a given light direction as center. By sampling more points, the shadows become smoother and more realistic. The bands are the concentric circles formed with center the point of intersection of the light direction with the unit sphere. The lesser the bands and the sampling rate are, the lesser the quality of the soft shadows becomes (Figure 6). As the distance of the sampled points from the center of the circle becomes greater, the amount of shadow contribution decreases, resulting in soft shadow outlines. All shadows can be stored in a single texture, along with the image-encoded center of the base of the trunk (first 4 bytes). They could be used with projective texturing methods for realistic results to be obtained. One issue that arises with this approach is the increased required texture memory for a single tree type.

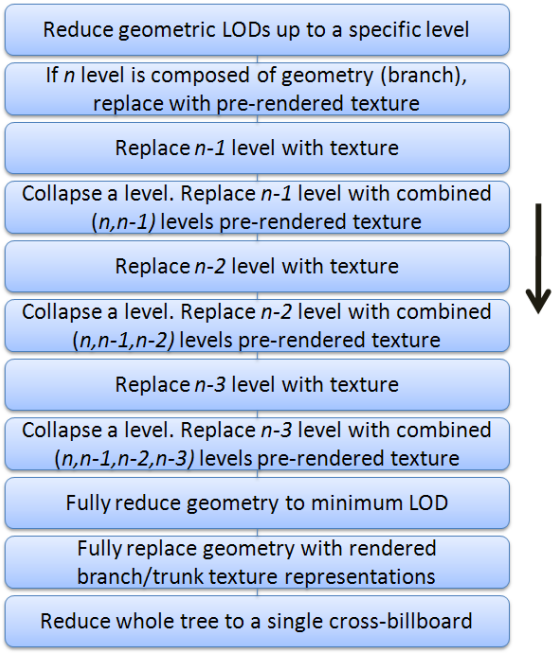

Figure 4. Example LOD generator priorities (top to bottom), limited to four combined level renderings.

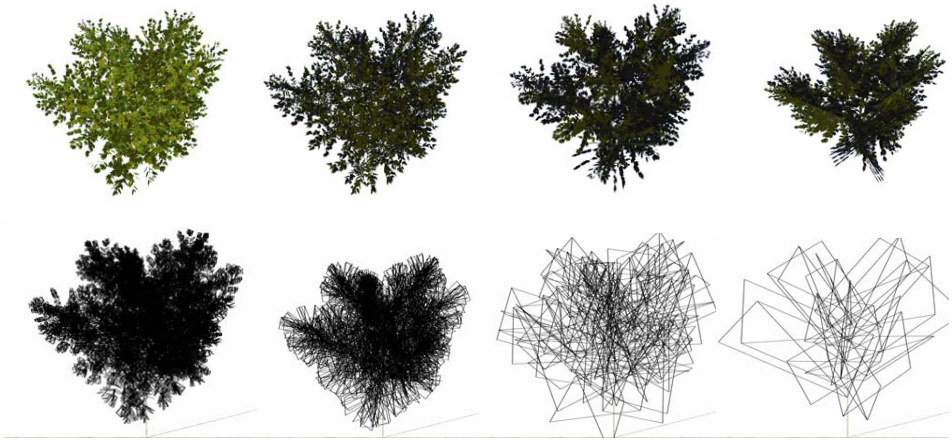

Figure 5. Foliage simplification – fill & wireframe modes. The rightmost three use the same shared pre-rendered texture.

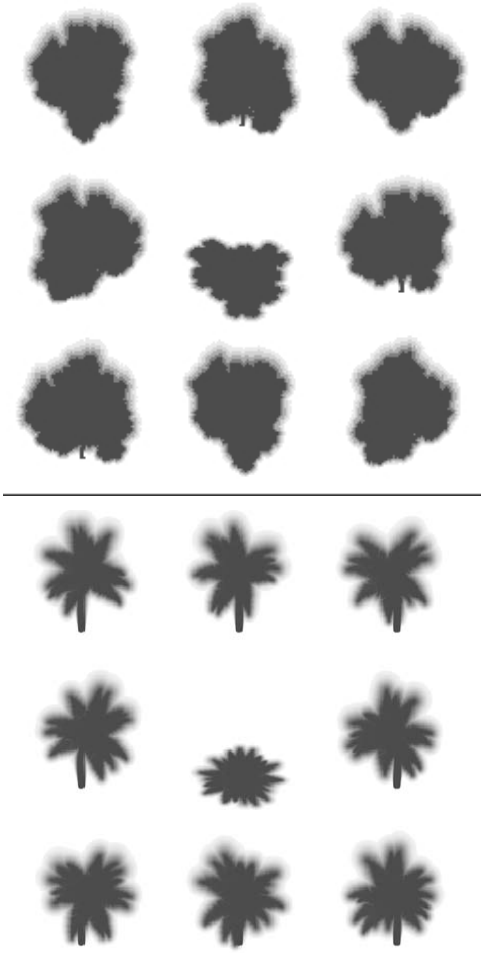

Figure 6. Example low (upper) and high (lower) resolution soft shadow textures

G. Output data

The output data from a preprocessing application, using all the above, can be passed to a real-time application, should be a set of textures and models. The textures should include the original bark and an image containing a concatenation of all the leaf textures, the created rendered textures of the various combined levels and branch types and finally the nine soft shadow textures (or a single atlas). The models can be separated into two groups, one containing all branches which are represented by their actual geometry, and the other containing all the textured representations, either branch, or combined levels or leaves. These two groups can be constructed for each LOD level, so the application should be able to handle easily the models in respect to the drawing method required (no transparency and normal lighting for geometry, transparency and faked lighting for pre-rendered textured quadrilaterals).

V. RENDERING - PERFORMANCE

With the suggested procedure, we can generate meshes (composed of vertices, UVs & normals) with their materials & textures, ready for rendering. This means that any performance metric of the actual rendering won't actually show the performance of the tree models themselves, but the performance of any mesh-rendering technique used. We used the fixed function pipeline, display lists & a simple LOD system to select the appropriate level of a tree based on distance. The results in the test system were the following:

Trees	Triangles	LODs triangle num	Fps
5929	5,500,000	10,640 – 836 – 248 – 4	9.1
5929	960,000	836 – 248 – 4	14.3
5929	370,000	248 – 4	18.2
381	5,500,000	52,000 – 4	30

The system used was an Intel Core2 Quad 2.50GHz CPU, with 2GB DDR3 RAM & a GeForce 9800GTX 512MB graphics card.

The bad performance that was experienced with the 6K trees is because of the inexistent batching, and it can be shown with the last example that the main bottleneck is the number of calls. With appropriate use of hardware instancing we could have obtained much better results.

The texture size used for the above was an uncompressed 512x128 RGBA 32bit texture. Different sizes didn't make difference since only one tree type could be used and the bottleneck was elsewhere. The POT was a requirement which doesn't exist anymore, resulting also in much wasted space, so without this restriction, we could have used hardware supported compression (e.g. DXT1/3/5) along with NPOT sizes to make better use of texture memory.

There are many recent techniques that could have been used given there were no restrictions, such as shaders, vertex buffer objects and hardware instancing, which should further improve the performance and quality of rendering.

VI. SPECIAL CONSIDERATIONS

A. Using randomization for tree generation

The randomization process should be heavily tied to the generation process, in order to obtain correct results. The generation process depends on the randomization of the parameters, for a huge number of tree variations to be possible. The randomization process is limited in effect by the logic of the tree creation. The major factor of tree geometry simplification is the collapse of a branch node and sub-tree, into two rendered quadrilaterals. Since the texture of the quadrilaterals will be shared by all the nodes on the level of the node that collapsed, all randomizations applied to all nodes of the collapsed node's sub-tree must be identical to every set of randomizations in its neighboring nodes. This is better illustrated in Figure 7.

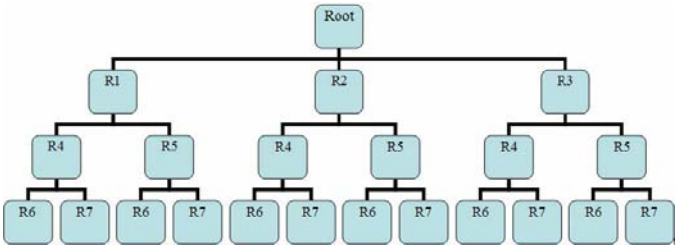

Figure 7. An example tree hierarchy using Rx parameter randomizations

B. Rendered texture filtering

The rendered and leaf textures have alpha components to specify transparency. Two methods that can be used for displaying them are using alpha testing, and pre-multiplied alpha blending. These methods generally do not cause problems, especially if multisampling / sample alpha to coverage is used instead of raw alpha test.

Besides alpha test, back-to-front blending can also be used for the canopy, using the following: First, for each tree canopy (sub-model storing only billboards, mentioned in section 4.7), we can store sets of indices for sample directions on a unit sphere. Then, when rendering each tree, we can select the index set depending on the vector from the camera to the center of the tree. Still this might create problems when trees are close to each other, resulting in sometimes the 'front' trees rendered before the 'back' ones.

VII. CONCLUSION

The method described above is generally a simple to implement way of generating multiple trees of arbitrary complexity. In most cases, convincing models of low triangle counts can be achieved, which are ideal for applications that have a low vertex throughput budget. The only manual work that needs to be done is the creation of scripts and splines. The parameters can be slightly randomized, to produce from the same script different trees each time. This can also be achieved by assigning different input data, especially spline files. Also, since models and their LODs are created offline, there is almost no computation to be done in the client application, which further saves CPU and GPU processing power. In newer graphics cards the whole preprocessing can be ported completely to the GPU resulting in a huge increase in speed, thus enabling the parameters and inputs to be specified interactively, providing immediate feedback to the user/modeler.

The main disadvantage of the presented method is the inability to create specific types of trees, or completely realistic ones, since the transformations are applied only locally (being parent-dependent) and not in a greater or global scope (e.g. global deformations such as tropisms cannot be created). Fortunately, such cases are not very common. Another disadvantage is the inherently bad lighting, since it cannot be copied and transformed as data can. A solution for realistic lighting would be to compute and store instead diffuse and normal maps (and maybe depth maps), but that would require more texture memory and the additional per-pixel computations on the GPU. A last major drawback is the potentially large texture it creates, making prohibitive the use of many different tree types in a virtual environment. Although the resolution of the output textures can be tweaked, if it is set low enough, the visual quality will drop considerably.

Despite the drawbacks, the method can be enriched to produce a more complete tree model-generating solution. Two features that can be added are wind animation and realistic lighting, mentioned above. Also, hardware accelerated features of newer graphics cards could be exploited, especially the programmable pipeline, for better and much faster results in the preprocessing phase as well as the real-time phase. While these generally make the trees more dependent on the client application and used hardware, they offer a significantly better degree of realism.

The system was used successfully, generating tree models which were used for several VR productions for FHW. The only problem was that the script-based method for specifying rotations made the modeling learning curve a little steeper than expected.

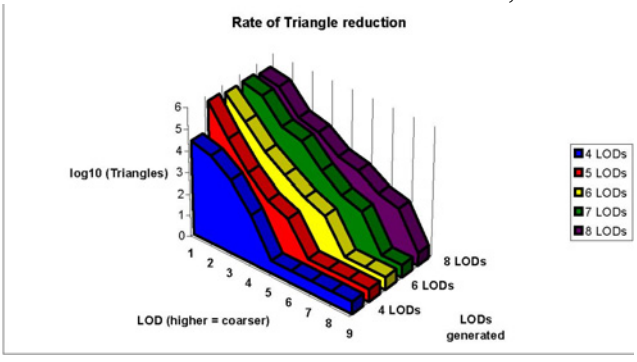

Figure 8. LOD information generated automatically during the creation of 180 trees (13 different species with variations)

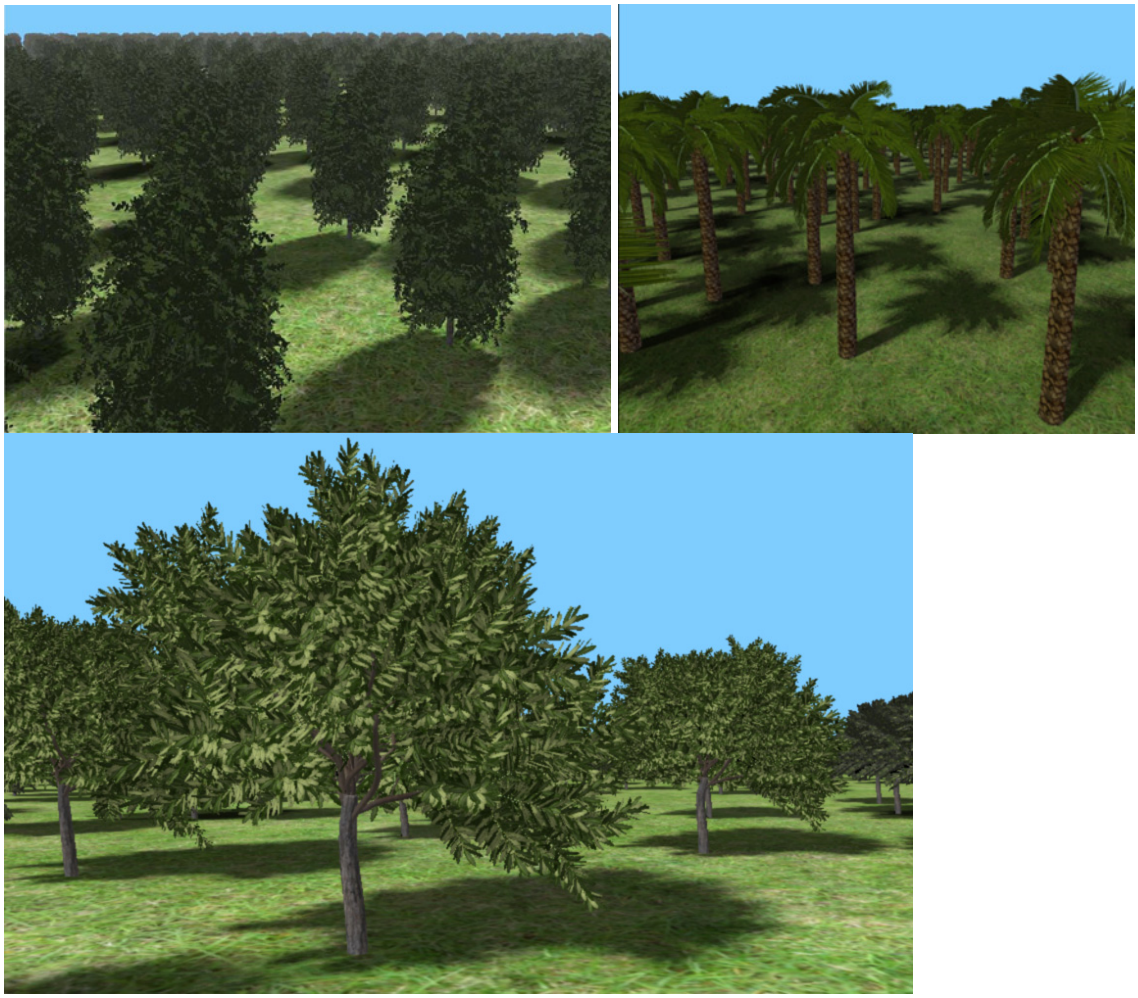

Figure 9. Some simple scenes showing forests populated by poplars, palms & olive trees.

ACKNOWLEDGMENT

Thanks to Dimitrios Christopoulos, for his continuous support in the implementation of various aspects of this work, and Panopoulos Panagiotis for all the texture generation work required. Thanks to Disney for funding all the presentation-related expenses.

REFERENCES

[1] 1. *Philippe Decaudin, Fabrice Neyret.* Rendering Forest Scenes in Real-Time, EG 2004.

[2] 2. *Olive Deussen, Cursten Colditz, Marc Stamminger, George Drettakis.* Interactive Visualisation of Complex Plant Ecosystems, IEEV 2002.

[3] 3. *Ismael Garcia, Mateu Sbert, Laszlo Szirmay-Kalos.* Leaf Cluster Impostors for Tree Rendering with Parallax, EG 2005.

[4] 4. *Jose L. Hidalgo, Francisco J. Abad, Emilio Camahort.* Simplification for Efficient Rendering of Tree Foliage, VIIP 2006.

[5] 5. *Aleks Jakulin.* Interactive Vegetation Rendering with Slicing and Blending, EG 2000.

[6] 6. *J. Dylan Lacewell, Dave Edwards, Peter Shirley, William B. Thompson.* Stochastic Billboard Clouds for Interactive Foliage Rendering, JGT 2006.

[7] 7. *Bernd Linterman, Oliver Deussen.* Interactive Modeling of Plants, CGA 1999.

[8] 8. *Javier Lluch, Emilio Camahort, Roberto Vivo.* An Image-Based Multiresolution Model for Interactive Foliage Rendering, WSCG 2004.

[9] 9. *Stephan Mantler, Robert F. Tobler, Anton L. Fuhrmann.* The State of the Art in Realtime Rendering of Vegetation, VRVIS 2003.

[10] 10. *Alexandre Meyer, Fabrice Neyret, Pierre Poulin.* Interactive Rendering of Trees with Shading and Shadows, EGWR 2001.

[11] 11. *Przemyslaw Prusinkiewicz, Aristid Lindenmayer.* The Algorithmic Beauty of Plants, 1990.

[12] 12. *C. Rebollo, I. Remolar, M. Chover, J. Gumbau.* Hardware-oriented Visualisation of Trees, DGCI 2006.

[13] 13. *C.Rebollo, I. Remolar, M. Chover, O. Ripolles.* An efficient continuous level of detail model for foliage, WSCG 2006.

[14] 14. *I. Remolar, M. Chover, J. Ribelles, O. Belmonte.* View-Dependent Multiresolution Model for Foliage, WSCG 2003.

[15] 15. *Gabor Szijarto, Jozsef Koloszar.* Real-time Hardware Accelerated Rendering of Forests at Human Scale, WSCG 2004.

[16] 16. *Jason Weber, Joseph Penn.* Creation and Rendering of Realistic Trees, SIGGRAPH ’95.