SIMPLIFICATION FOR EFFICIENT RENDERING OF TREE FOLIAGE

Jose L. Hidalgo, Francisco J. Abad and Emilio Camahort
D. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, Spain
email: jhidalgo, fjabad, camahort@dsic.upv.es

ABSTRACT

Tree modeling and rendering is an integral part of many modern computer graphics applications. Unfortunately, tree models are highly detailed and require a lot of geometry information. To solve this problem both image- and geometry-based simplification techniques have been proposed. These techniques build multiresolution representations that either have large storage requirements or do not allow viewing the trees from a close distance. We present a multiresolution model that can be efficiently stored in the GPU and produces highly realistic views of trees at close range. Our model supports a rendering algorithm that only requires two render operations to display any level of detail of any tree. We propose a simplification method targeted at this rendering algorithm. This simplification method produces continuous levels of detail of the leaves of each tree. We show that our algorithm can render scenes with as many as several million trees modeled using our representation.

KEY WORDS
Tree modeling and rendering, multiresolution representations, geometric simplification, GPU-based rendering.

1 Introduction

Reproducing realistic natural scenes has always been a difficult challenge in real-time rendering applications. Natural scenes contain different species of trees and modeling each of them usually requires a huge amount of geometric information. Representing a scene made of several models at full resolution is not possible, even with modern GPUs. Additional techniques are required to render those scenes at interactive rates.

The two most used approaches to tree and plant rendering are geometry- and image-based techniques. The goals of both types of techniques are to render a large number of trees and, at the same time, to obtain the most realistic images. Image-based techniques usually start working on a complete geometric model of the tree. This is because the geometric representation achieves the highest level of detail, at the expense of higher complexity. Therefore, research on efficient geometric algorithms is necessary in both fields.

This paper presents simplification and rendering algorithms suitable for close tree exploration. They allow real-time level of detail changes of trees with no time penalty. Furthermore, all the trees of the same species share the same data structure. That is, we can instantiate millions of trees based on a few geometric models with very low memory overhead.

This paper is organized as follows. First, we briefly present current techniques used to represent trees. Then, we present our rendering algorithm and simplification method. The paper ends with some results and conclusions.

2 Background

The two main goals in applications that reproduce tree species are: to achieve high quality images of a single tree to explore it, and to obtain high quality renderings of forests with different trees. We try to fulfill both goals.

In [1, 2] Decauding et al. and Beherendt et al. introduced different techniques to render dense forests based on 3D textures and volumetric algorithms. These techniques work well for viewing a forest from far away, but artifacts appear when the viewer gets closer to the trees. This is useful in applications like flight simulators, but does not work when the user is allowed into the forest.

Techniques for rendering single trees, like image-based and geometry-based techniques, can be extended to render forests with some limitations. Current research has shown that using just the geometric representation of a polygonal tree is not feasible. Therefore simplification or multi-resolution techniques are needed [3, 4].

2.1 Geometry- and Image-Based Representations

Most of the techniques used to represent trees and plants are either geometry-based [5] or image-based [6, 2]. Other authors have used particle systems [7].

One of the most important problems that image-based techniques have is parallax simulation. When the user looks around a tree, she or he should see different levels of leaf depth, moving at different speeds. In [8] the authors solve this problem using static planar impostors that replace sets of leaves located nearly on the same plane. Still,

most image-based techniques present problems when rendering close views of the tree.

To solve this problem, we use geometry-based methods. One solution creates a small number of discrete resolution levels called levels of detail (LODs). Discrete LODs are useful for real-time applications and videogames since they have acceptable GPU memory requirements and their models do not need to be updated. Their drawback is the *popping* effect (sudden changes in the image) that occurs when the LOD changes.

Continuous LOD algorithms solve this problem by allowing smooth transitions between LODs. They are more difficult to generate and handle, but produce no popping artifacts [9].

It is very difficult to create a complete real application using a purely geometric approach. Usually, interactive applications need a balanced use of geometry and image-based techniques, such as impostors [10, 11].

2.2 Leaf Simplification Techniques

The techniques used to simplify general geometric models usually fail to simplify sparse geometries, like tree foliage. Simplification techniques are based on the collapse of several polygons into a single one, or simply on the removal of polygons from the model. When the model is composed of isolated polygons, the simplified models are usually unacceptable.

Remolar et al. present a simplification algorithm designed for sparse geometric models [12]. This method is based on a simplification operation that collapses those two leaves that minimize a distance function. The distance function is based on the distance between the two leaves and the number of original leaves represented by each LOD node. This operation takes two leaves and creates a new leaf with area similar to the originals'. The method reuses the vertex information of the leaves and therefore adds no new geometric information to the model.

Zhang and Blaise introduce a similar approach that adds new parameters like area similarity, deformation measures and a diameter penalty to the distance function [13]. The new distance function can be fine tuned by adjusting several weights.

In [5] two geometry extraction algorithms are presented. They include both a variable and a uniform technique to extract the polygons that compose a given LOD. Variable extraction algorithms allow the user to direct the extraction using a given criterion like, for example, camera position. On the other hand, uniform extraction increases or decreases the level of detail globally.

Both techniques require every instance to store a list with its active nodes. Given the hierarchical data structure derived from the distance function used to decide whether two leaves should collapse, and given the specific criteria, each node is checked to decide whether it has to be collapsed or split. Then, the changes are applied and the nodes are updated.

This algorithm can not be used in scenes with millions of trees because each instance needs information of its own state and even if vertex information is stored in the GPU, indices can not be shared between instances. On the other hand, only the geometric location of each vertex can be shared among several leaves, and new normals and texture coordinates should be computed for each new leaf. Therefore, these multiresolution techniques can not be used in applications for real-time rendering of forests with a large number of trees.

3 Our Approach

The goal of the simplification method is to preserve the original appearance of the tree, for every viewpoint. Furthermore, the appearance of a tree depends on its distance to the viewer. Therefore, we define a good simplification criterion as one that minimizes the change in the appearance of the tree when it is represented at the viewing distance associated to the LOD.

Our simplification algorithm is based on the Foliage Simplification Algorithm (FSA) [5, 12]. It repeatedly selects and collapses pairs of leaves that minimize a distance function. Therefore, each simplification step reduces by one the number of leaves in the tree model. Our approach improves on this simplification algorithm and implements a more efficient rendering algorithm.

We build a continuous LOD model for the leaves of a tree. Since the selection process to determine which pair of leaves is going to be collapsed and the collapse process are independent, we will address each one of them separately. A multiresolution representation for the branches of a tree can also be built by using traditional geometric simplification.

3.1 Selection Criteria

In order to determine which pair of leaves should be collapsed we find the pair of leaves that minimizes a distance function. This function depends on:

- the euclidean distance between the leaf centers,
- coplanarity: the angle between the leaves' normals,
- number of leaves: the difference between the actual number of leaves represented by each leaf; initially, each leaf in the model represents only one leaf (i.e., itself); after collapsing two leaves, the resulting leaf represents the sum of the number of leaves accounted for by the original leaves,
- difference in area: takes into account the difference between the area of each leaf, to avoid large leaves collapsing with much smaller leaves, and
- interiority: it is the distance from the midpoint between both leaves and the axis of the tree; this criterion favors the collapse of the inner leaves of the tree that, when seen from far away, have less impact on its appearance.

Each of these factors is weighted by an adjustable parameter to increase or reduce its importance. Each step in the simplification process involves computing the distance function between every pair of leaves and selecting the pair that minimizes the function.

In order to maintain the aspect of the tree, we impose limits on the choice of pairs of leaves. Using several thresholds, we prevent pairing of leaves that are too far from each other or oriented in substantially different directions. At the end we obtain a few polygons representing all the leaves of the tree. Each polygon corresponds to a set of leaves located close to each other.

3.2 Generating a New Leaf

In this Section we present how to create a new leaf from the two leaves selected in the previous step. In previous works, the data structure was oriented to reuse the vertices of the collapsed leaves in the new leaf [12, 13], thus reducing the amount of memory required by the model. On the other hand, generating a new leaf using this method has several drawbacks. First, the new leaf will generally not be a quad with four coplanar vertices, thus degenerating the geometry of the leaves in the process. Also, each vertex stores its texture coordinates. If we choose the four vertices that maximize the resulting area, the correct texturing of the new leaf can not be guaranteed. Finally, vertices usually store their normals as well. Reusing old vertices in the new leaves produces incorrect shading.

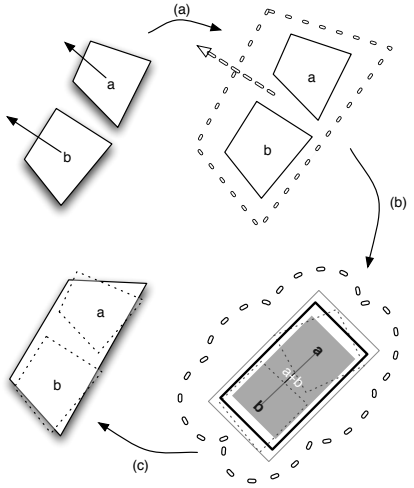

Figure 1. Generating a new leaf: *(a)* the two collapsing leaves are projected onto a new plane; *(b)* the size and orientation of the new quad are selected; and *(c)* the new quad is obtained.

Our algorithm allows representing the new leaves as new polygons that do not share any previous vertex information. This solves the above drawbacks.

The following algorithm generates a new leaf from two leaves selected by our distance function

1. The center of the new leaf is located at the midpoint between the collapsed leaves' centers, weighted by the number of leaves represented by each leaf.
2. The orientation of the new leaf (its normal) is the weighted mean of the normals of the collapsed leaves.
3. Once the plane of the new leaf is defined, we orthographically project the vertices of both leaves onto it (see Figure 1). The line that passes through the projected centers of the leaves defines the main axis of the new leaf. The secondary axis is perpendicular to the main axis. This defines the quad that will contain both projected leaves.
4. Since leaves may or may not overlap, the area of the new leaf will most likely be different from the sum of the areas of the original leaves. To avoid a large increase or decrease in the area of the new leaves, we set the new leaf to have the average area between the sum of the areas of the original leaves and the area of the bounding quad that encloses both projections.
5. The texture coordinates of the new leaf are set taking into account the foliage density represented by the leaf, using a pre-computed texture atlas. We assign a complex texture to a polygon that represents several leaves, instead of using huge single leaves as in the original model. To avoid artifacts when changing LODs, the texture atlas is more detailed for the finest LODs (see Figure 2).

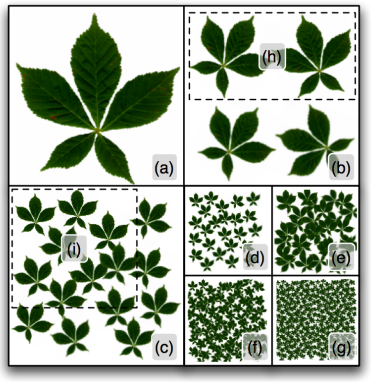

Figure 2. Pre-computed texture atlas. *(a)* to *(g)* contain texture images for different leaf LODs.

3.3 Generating the Texture Atlas

In our current implementation we use a texture atlas composed by an artist taking into account the model requirements. This atlas is labeled so that, given a leaf density and

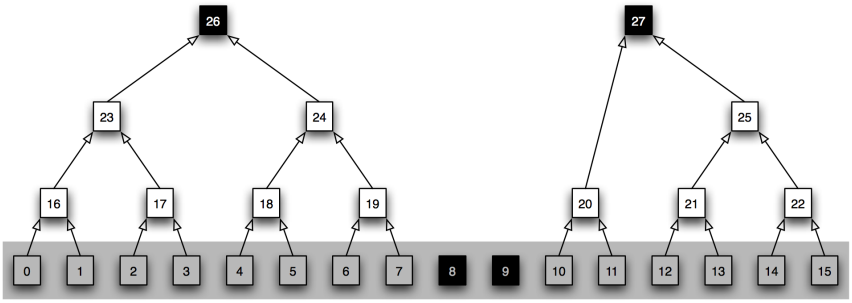

Figure 3. Example of a simplification chain. Each leaf is labeled with the order in which it was generated. The highlighted leaves (at the bottom, in gray) represent the geometry of the original model. The nodes with black background and white label represent the lowest LOD. Each node points to the node it collapses to.

an LOD, the proper subregion of the atlas is selected to texture map the LOD's quad. The problem is how to generate the texture atlas automatically.

Our approach starts by simplifying the leaves without considering textures. Then we build the atlas by taking into account leaf collapses with similar numbers of leaves. Higher LOD leaves require computing better quality textures, as these leaves are viewed from closer distances. We compute the distribution of the distances between collapsed leaves and the distributions of the relative orientations of the same leaves. Then we decide which are the most significant distances and orientations and use them to render the first level of atlas textures. Each texture contains a pair of leaves with one of those distances and orientations.

As we simplify lower LODs we start taking into account the density of leaves within each quad. By density we mean the leaves' projected area divided by the supporting quad's area. We determine the most significant densities of leaves and render textures with those densities. The texture atlas thus obtained has texture maps good for rendering collapsed leaves with different distances and orientations, and quads with different densities of leaves.

4 Rendering Algorithm

The input of our algorithm is a generic hierarchical simplification chain. It is stored in a forest data structure, composed of one or more binary trees, as seen in Figure 3. We transform the hierarchical data structure into a linear data structure that allows uniform LOD extraction. After building the data structure, we store it in the GPU memory as long as needed, since it is a static structure that does not need to be modified. This reduces the amount of CPU-to-GPU communication when rendering lists of polygons whose information is already stored in the GPU. Furthermore, all the instances of the same tree type share the same data structure. Each instance only needs to know its own LOD at rendering time. This means that there is no geometry extraction involved in the process. Thus the switch from an LOD to another has no additional cost.

Figure 4 shows the process of linearizing the hierarchical simplification chain shown in Figure 3. First, the vector is initialized with the roots of the (binary) forest. The black line in the middle separates the leaves into two groups: the original geometric model (on the left) and the new leaves created by collapse operations (on the right). Indices p and q point to the first free position for each type of leaf. The algorithm processes backwards the new leaves, distributing its children in the proper part of the vector, depending on its type.

In parallel we build the LOD table, where each entry contains 3 integers a, b, c. This table indicates which part of the vector should be rendered for each LOD. Our rendering algorithm is able to draw any LOD with just two render operations of two subvectors of the geometry vector: $[0, a]$ and $[b, c]$. Here 0 and b are the offsets within the vector and a and c are the lengths of the subvectors.

There are several advantages in this approach. First, all the information is shared among all the instances of the same species, and there is no redundant information. Second, it is not necessary to update the data structure for each change of LOD, because it already stores every LOD. Third, this model only needs to use vertices (it does not use indices) because the render operations are linear. And last, the vertices of the new leaves are independent, so they can be computed freely in each collapse, without the limitations imposed by reusing the original vertices. This allows us to properly compute both the texture coordinates and the normals of each new leaf.

4.1 Extended Rendering Algorithm (ERA)

The simplification process described above obtains a continuous level of detail model that allows us to represent all the leaves in a tree for a given LOD. However, it seems unreasonable to represent the whole tree at the same resolution, when the viewer is only facing one side of the tree. If we divide the initial set of leaves into sectors (see Figure 6) we can apply the simplification process independently to each of these sectors.

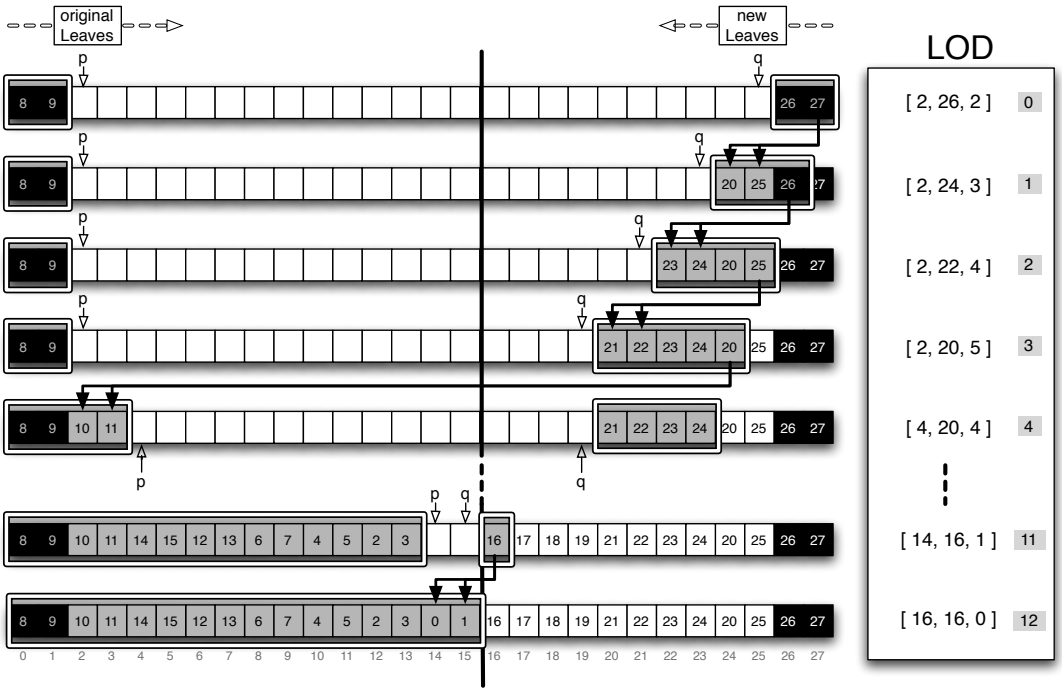

Figure 4. Leaf vector and LOD table evolution for the example in Figure 3. The first row shows the vector after initialization and its entry in the LOD table. The following rows show the evolution of the algorithm and how the LOD table is filled. In each row, the leaves that compose that LOD are encircled.

Once the model is simplified into independent sectors, the rendering algorithm can be easily extended to perform two render operations per sector. With this approach, we can represent the visible part of the tree with higher detail. Our technique improves on variable LOD representations because it does not require geometry extraction [5].

This sector-based technique supports rendering different tree sectors at different resolutions. For example, visible outer sectors will usually be rendered with a higher resolution than occluded and innermost sectors. Furthermore, by allowing sectors of different sizes, this technique solves the problem of non-uniform trees, where the foliage distribution is heterogeneous, with sparse clusters of leaves in certain areas of the tree. Note that this technique works even if the viewer is above, looking down at all sectors of the tree.

5 Results

We have implemented our representation and rendering algorithm. They both can be used with different simplification algorithms. Our implementation uses OpenGL and Vertex Buffer Objects (VBOs) to store the multiresolution representation of the foliage of the trees. A VBO is a GPU memory area that stores vertex information like position, normal and texture. For each leaf we store a quad and for each LOD a set of quads. Changes in LODs only require two VBO rendering operations. This enables interactive rendering of scenes with millions of different tree instances. Figure 5 shows three different screenshots of a scene containing up to five million trees modeled with our representation. Our algorithm processes only the foliage, in order to render efficiently a complete tree other simplification algorithms should be applied to branches and trunk.

6 Conclusions

We have presented new algorithms for simplification, construction and rendering of tree models. Our simplification algorithm produces continuous LODs for the leaves that make up the foliage of a tree. All the trees of a species can be represented by a single quad array stored in the GPU memory. The different LODs are made of texture mapped quads that can be efficiently accessed for rendering. Rendering a tree specimen only requires displaying two arrays of quads stored in the GPU. Changing an LOD requires no communication between the CPU and the GPU. These features allow instantiating millions of trees and rendering hundreds of thousands of trees.

Acknowledgements

This work was partially supported by grant TIN2005-08863-C03-01 of the Spanish Ministry of Education and Science and STREP project IST-004363 of the 6th Framework Program of the European Union.

Figure 5. Three different views of a scene with up to five million trees rendered with our algorithm.

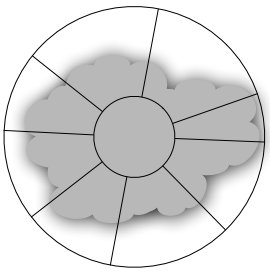

Figure 6. Top view of a tree divided into non-uniform sectors; the center sector contains the innermost leaves.

References

[1] P. Decaudin and F. Neyret, "Rendering forest scenes in real-time," in *Eurographics Symposium on Rendering*, pp. 93–102, june 2004.

[2] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen, "Realistic real-time rendering of landscapes using billboard clouds," in *EUROGRAPHICS*, vol. 24, 2005.

[3] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *SIGGRAPH '98*, pp. 275–286, ACM Press, 1998.

[4] G. Szijártó and J. Koloszár, "Hardware accelerated rendering of foliage for real-time applications," in *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, (New York, NY, USA), pp. 141–148, ACM Press, 2003.

[5] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte, "View-dependent multiresolution model for foliage," *Journal of WSCG (WSCG'2003)*, vol. 11, no. 1, pp. 370–378, 2003.

[6] A. Candussi, N. Candussi, and T. Höllerer, "Rendering realistic trees and forests in real time," in *Eurographics 2005, Short papers*, (Dublin, Ireland), 2005.

[7] D. Marshall, D. S. Fussell, and I. A. T. Campbell, "Multiresolution rendering of complex botanical scenes," in *Proceedings of the conference on Graphics interface '97*, (Toronto, Canada), pp. 97–104, Canadian Information Processing Society, 1997.

[8] I. Garcia, M. Sbert, and L. Szirmay-Kalos, "Leaf cluster impostors for tree rendering with parallax," in *Eurographics 2005, Short papers*, (Dublin, Ireland), 2005.

[9] C. Zach, S. Mantler, and K. Karner, "Time-critical rendering of discrete and continuous levels of detail," in *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, (New York, NY, USA), pp. 1–8, ACM Press, 2002.

[10] E. Sayer, A. Lerner, D. Cohen-Or, Y. Chrysanthou, and O. Deussen, "Aggressive visibility for rendering extremely complex foliage scenes," in *5th Korea-Israel Bi-National Conference on Geometric Modeling and Computer Graphic*, 2004.

[11] S. Dobbyn, J. Hamill, K. O'Conor, and C. O'Sullivan, "Geopostors: a real-time geometry/impostor crowd rendering system.," *ACM Trans. Graph.*, vol. 24, no. 3, p. 933, 2005.

[12] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo, "Geometric simplification of foliage," in *Eurographics'02 Short Papers*, pp. 397–404, 2002.

[13] X. Zhang and F. Blaise, "Progressive polygon foliage simplification," in *Plant Growth Modeling and Applications*, (Beijing, China), pp. 182–193, Tsinghua University Press, October 2003.