An Image-Based Multiresolution Model for Interactive Foliage Rendering

Javier Lluch
Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
46022 Valencia, Spain
jlluch@dsic.upv.es

Emilio Camahort
Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
46022 Valencia, Spain
camahort@dsic.upv.es

Roberto Vivó
Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
46022 Valencia, Spain
rvivo@dsic.upv.es

ABSTRACT
This paper presents a new method for realistic real-time rendering of tree foliage. Some approaches to this problem have been presented before but the quality of their results was not maintainable with respect to changes in view vector and observer distance. Our method is based on a hierarchy of images obtained from pre-processing the botanical tree structure (an L-system) and storing the information in a texture data tree without increasing rendering time. The texture tree is traversed for each frame and an appropriate set of images is extracted and blended with the previous image set. The number of polygons is dramatically reduced – thus enabling interactive visualization and smooth transition between levels of detail. Our method can be easily applied to computer games and visual interactive applications containing vegetation.

Keywords
Plant and tree modeling and rendering, multiresolution modeling, image-based rendering.

1. INTRODUCTION
Tree modeling is a field that has lately received much attention from the computer graphics community. Techniques like L-systems [Prus90], modeling by components [Lint99], and commercial applications such as OnyxTREE [onyx03] and AMAP [deRe88] suggest the possibility of interactive rendering scenes made up of tens, or hundreds, of trees. Still, these modeling methods produce too many polygons, making them unfeasible for interactive rendering of large scenes with many trees.

One way of modeling trees more efficiently is by separating the model for the trunk and the branches from the model for the leaves. In this paper we use L-systems to model the trunk and branches. Given an L-system we derive it using production rules and obtain a string that can be graphically interpreted [Prus90]. Interpretation produces a multiresolution model that enables accelerated visualization of the geometry of the trunk and branches.

For the tree foliage we propose a model that uses a set of pre-computed images for rendering. These images replace the leaves contained in a bounding box representing a group of branches. The images are organized in a hierarchical fashion representing different levels of detail (LODs). The method that computes this multiresolution representation is fully automatic. We obtain the best results when the size of the leaves is small compared to the size of the plant.

Our model supports tree rendering at different LODs depending on viewer distance. The visual quality of the renderings does not depend on camera orientation or viewer distance. The coarsest LOD is represented by a bounding box containing the leaves of the entire tree. The finest LOD is made of a set of texture-mapped polygons, one for each leaf.

This representation supports progressive transmission by sending a stream of images corresponding to the different LODs. For the trunk and the branches a text file containing the L-system can be sent with just a few bytes. The L-system can then be interpreted at the destination. Also, we can

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

use other techniques to model them [Lluc01] [Lluc03].

Using our multiresolution representation, we can render several hundreds of trees at interactive rates. A frame rate of 15 fps is achieved by carefully choosing the LOD to be rendered for each tree. Our method creates a complex model in less than a minute. The model occupies less than 250Kbytes in compressed format. A tree can be as leafy as desired because additional leaves do not increase the rendering time.

This article is organized as follows: Section 2 reviews previous work in interactive tree rendering, Section 3 introduces the method we use to obtain the data structures and images used for rendering, Section 4 describes our rendering algorithm, and Section 5 presents the results obtained with our method. Conclusions and future work are discussed in the last section of the paper.

2. BACKGROUND

There are many real-time and virtual reality graphics applications that attempt to immerse the user in an outdoor scene. The scene can be made up of numerous natural elements such as plants and trees. For a more realistic effect, the scene usually includes a lot of detail. For example, it is important that the leaves of the trees be represented realistically – just in case the viewer gets closer.

The most common acceleration technique for tree rendering uses *impostors*. An impostor replaces the tree, or part of it, with one or more textured polygons. There are two types of impostors. A billboard is an image texture mapped onto a polygon that is always facing the viewer. Alternatively, we can use two perpendicular polygons texture-mapped with a transparent texture. The main problem with these two approaches is that they lose realism as the user gets closer to the tree.

A different model common in MMORPGs (Massively Multiplayer On-line Playing Games) compromises between the use of geometry and the use of impostors. The method requires user intervention to model the trunk and branches. For the trunk it uses a few cylinders. For the branches it uses a few textured polygons. For the leaves, it uses one billboard for each branch.

Schmalstieg [Schm96] presents a method that supports direct rendering of L-systems by transforming the rewriting system into a directed cyclic graph. The method does not use an intermediate polygonal model making it very memory efficient. Once the graph is generated, it can be traversed from left to right to visualize the model. Only one model is needed to create a set of different individuals of the same species. It is enough to define the main features of the species and use some random values in the derivation process. Problems with this method appear when obtaining the LODs. LOD generation requires that a sub-graph (representing a tree branch) be replaced by a single primitive with an adequate color and shape. This is a complicated task that has to be done manually.

Meyer and Neyret propose a method that renders volumetric textures interactively [Meye98]. This approximation is based on slicing a piece of 3D geometry into a sequence of layers. A layer is a rectangle containing the shaded geometry of a given slice. These layers are later used as transparent textures in the rendering process. An object is made of a triangular mesh with texture coordinates and height vectors associated to each vertex. It also uses a volumetric pattern formed by a set of RGBA textures representing very thin horizontal volume slices. A problem occurs when the observer is located in a place where the lines of sight with the height vectors form an angle close to 90º. In this case, holes appear between different slices and realism is lost. The solution is to create a group of slices for each of the main directions, tripling the number of necessary textures.

Jakulin presents a technique that combines geometry for trunk and branches with images for leaves [Jaku00]. The crown is visualized by using a multilevel representation, where each level is made of a texture called a slice. For each tree, there are several groups of levels used to simplify rendering from different viewpoints. Transitions between LODs are made by controlling the texture opacity as a function of the angle formed with the look vector. This makes LOD transitions smoother and reduces the number of slices used by the model. Slices in the same group are parallel and equidistant. A tree is modeled using six slice groups forming a 60º angle between them, and containing five slices each. A total of 30 textures are thus used to model the leaves of a tree. This method is only valid for an observer located on the ground – and from ten to fifty meters away. Therefore, it is impossible to render fly-by's without losing realism.

3. DATA STRUCTURES

We present a new model for efficient plant and tree rendering. Given a tree represented by an L-system, the multiresolution modeling process begins by deriving a parametric chain. The chain is interpreted and a data structure is created containing the relevant information for rendering. In this Section we describe this data structure and the algorithm that builds it.

Description

The data structure is a tree made of nodes and edges. Each node has a link to its parent node, its children list and its sibling list. There is a special node, called root, with no parent or siblings. Figure 1 shows an example of this data structure for a simple tree. We use it to model the trunk and branches of any plant or tree.

Associated to each node we store a graphics primitive: for instance, a cylinder, a sphere or a polygon. Each node also contains information about the bounding box of its sub-tree. Bounding boxes are given by its two extreme points, *min* and *max*. They bound the current branch and all of its children.

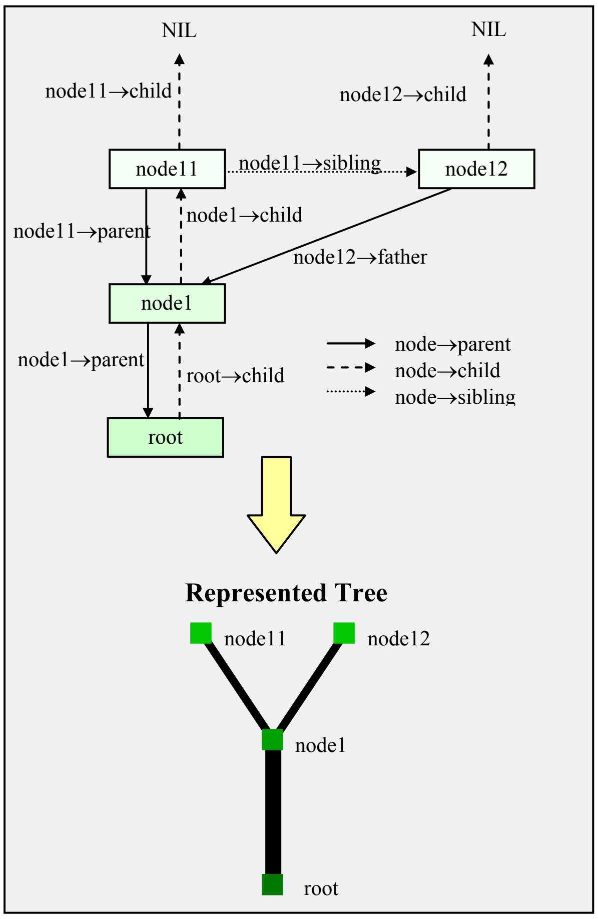


Figure 1. Tree hierarchical data structure.

Construction

We use stack-based turtle graphics [Prus90] to build a tree. We add a node every time a new module of the input string is interpreted. The module may have a graphical meaning, like a cylinder representing a branch, or it may encode a stack operation. There are two stack operations: *PUSH* that saves the turtle state in the stack and increases the tree level by one, and *POP* that recovers the stack state and decreases the tree level by one.

The bounding box hierarchy is built while building the tree data structure. *PUSH* and *POP* modules represent the beginning and end of a new branch and, therefore, a new bounding box. At any time during the interpretation, an *open box* is a box whose dimensions are not final, because the children of its associated branch have not yet been completely traversed. The most recently created box is called *last box*. Finally, *level* gives the current depth in the graph.

3.2.1 Generating the Bounding Boxes

The following algorithm computes the bounding boxes associated to a tree's parametric string:

```
Process the modules of a parametric string
  In case module is:
    PUSH              Create a new box
                      Give initial values to min, max
                      Mark the box as last and open
                      Increment level
    POP               Close the last box
                      Save the box
    CYLINDER, FORM
                      Update min, max of all open boxes
    FORWARD           Modify (advance) the turtle position
    TURN              Modify (rotate) the turtle orientation
```

The algorithm runs as follows. When the interpreter finds a *PUSH* module, a new *open* box is created and initial values are assigned to its endpoints. From then on, all *open* box dimensions are updated for every module parsed, until a *POP* module is encountered. Once the process is finished, each node has an associated bounding box. Note that L-systems produce well-formed parametric chains, with equal numbers of *PUSH* and *POP* modules. Therefore, the bounding box associated to a node includes all the bounding boxes associated to its children, grandchildren, etc.

After running this algorithm we are left with a bounding box hierarchy containing the tree's geometry. Now we calculate a set of images representing the geometry located inside of each suitable box.

3.2.2 Generating the Pre-Computed Textures

We use images (textures) to replace the geometry of a tree's leaves. When a bounding box is closed, its dimensions define a volume containing a branch of the tree and its children branches. So, we use the bounding box to generate a set of images that represent the geometry of the leaves located inside the box. At rendering time, we replace the leaves' geometry with images.

We generate an orthographic projection for each side of a box. We place the camera at the center of the target face, with the look vector facing the center of the box. Then, we project the leaves inside the box onto the target face, and store the resulting image as a texture map. At the end of the process, we have generated six images per bounding box. This may require a large amount of storage, especially if we generate one set of images per branch. The size of the images is also relevant. Most graphics cards have a texture memory of 32Mbytes – 64Mbytes for high-end cards. Hence, we need to find a compromise between texture size and model quality. Figure 2 shows an example.

3.2.3 Texture Size

Graphics cards typically impose two conditions on the selection of a texture's size. First, both the horizontal and the vertical dimensions must be the same. Second, they have to be powers of two. We avoid textures that have dimensions larger than 128, since they are too big and require too much texture memory. We also avoid textures smaller than 64, since they are too small and introduce distortions when projecting the foliage. Consequently, we use textures of 64 and 128 square pixels. 64-pixel textures are used to represent four times more trees at a lower resolution. 128-pixel textures are used to represent higher-quality trees.

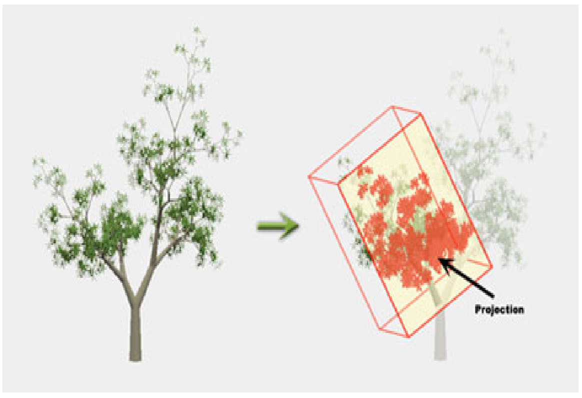


Figure 2. Generating a pre-computed texture to replace the leaves of a tree branch

Bounding boxes at the same tree level correspond to the same LOD. Boxes at higher levels include the boxes at lower levels. If the textures of a bounding box are rendered, the boxes inside of it are not rendered. If the observer is located far enough, the leaves will be rendered using the textures computed for the box bounding the whole tree. As the observer approaches the tree, smaller bounding box textures representing different branches are rendered.

We choose a constant texture size for all levels. A texture does not depend on the size of the box it represents. Using 128x128 textures, we can store an entire binary tree of depth ten in roughly 96 Mbytes of memory. Now we focus on the task of reducing the number of levels whose textures need to be calculated.

3.2.4 Number of Levels

We limit the number of levels using an adjustable strategy that takes into account the tree's topology. We compute the ratio between the volume of a node's box and the volume of its parent's box. If the ratio is smaller than a threshold, we do not generate textures for the node's box. We use this threshold to adjust the number of LODs and the number of textures for a given tree. The choice of threshold is a tradeoff between realism and the amount of storage used for textures.

4. RENDERING ALGORITHM

Interactive tree rendering is an expensive task due to the geometric complexity of tree models. Our algorithm solves this problem by using images to replace a tree's leaves. In this section, we discuss our rendering algorithm and comment on some acceleration and visual improvement techniques.

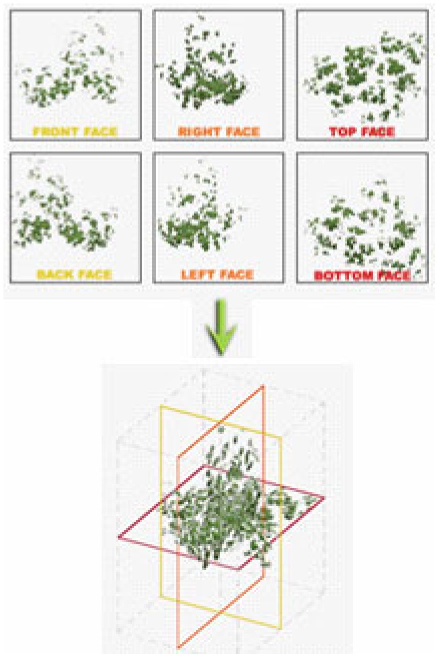


Figure 3. Rendering the textures associated to a bounding box.

Consider the case when the viewer is relatively far from the tree. In that case, we replace the geometry of all the leaves by six texture-mapped polygons. The textures correspond to the bounding box of the root of the tree. The polygons are arranged in a cross, as illustrated in Figure 3. Only three polygons are actually visible to the viewer; the other three are back-face culled.

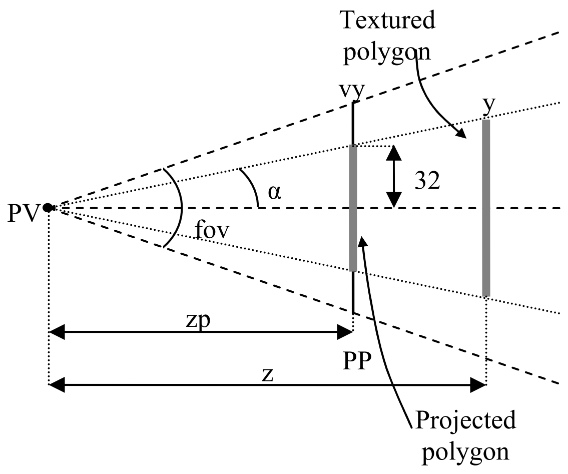
```
Leaf drawing algorithm
// Using geometry: cylinders, cones, and/or polygonal meshes
root_node := read tree root
child_node := left child of root_node
While (child_node <> NIL) ∧
      (level of child_node ≤ top detail level) Do
  obtain distance from viewer to child_node box
  If distance > lod distance Then
    draw child_node box
// Using texture-mapped polygons
// We have rendered a box for this branch, so we are
// done
// Now we continue with its next sibling, if it exists
    While (child_node <> NIL) ∧
        (right sibling of child_node = NIL) Do
        child_node := parent of child_node
    End While
    If (child_node <> NIL) ∧
        (child_node <> root_node) Then
        child_node := right sibling of child_node
    End If
  Else
// We continue rendering children
    If (left child of child_node <> NIL)
        child_node := right child of child_node
    Else
        While (child_node <> NIL) ∧
            (right sibling of child_node = NIL) Do
          child_node := parent of child_node
        End While
        If (child_node <> NIL) ∧
            (child_node <> root_node) Then
          child_node := right sibling of child_node
        End If
    End If
  End If
End While
```

Figure 4. Leaf rendering algorithm.

As the viewer gets closer to the tree, we render texture-mapped polygons associated to finer LODs of the tree. The polygons replace the leaves stemming from a branch and its children branches located inside of the LOD's bounding box. Figure 4 shows this leaf rendering algorithm. Note that we have a distance associated to each LOD. That distance determines when that LOD is to be rendered.

Computing the Distance Associated to an LOD

Box textures have an optimal display distance that depends on their projected area in the final image. That area is maximized when the projection happens along the look vector. We can determine the distance between the viewer and the box so that the projected area is 128 or 64 pixels. That way, if the image is located at that distance, we do not have to scale it before projecting it. The distance that switches from one LOD to the next one can be obtained from the distance that maximizes the projected area of the texture. The projected area will be reduced if the object is closer to the viewer or the projection is not perpendicular to the viewing direction (see Figure 5). Distances that produce changes in the LODs are pre-computed and stored for each bounding box. That way no extra computation is necessary at rendering time.



$$\tan(fov/2) = \frac{vy/2}{zp} \rightarrow zp = \frac{vy}{2 \cdot \tan(fov/2)} \Rightarrow$$

$$\frac{y/2}{z} = \frac{32}{zp} \rightarrow z = \frac{y \cdot zp}{2 \cdot 32} = \frac{y \cdot vy}{128 \cdot \tan(fov/2)}$$

Figure 5. LOD distance computation

Improvements on the Algorithm

Our algorithm produces two types of artifacts. 'Popping' occurs across textures in a box and when transitioning from one LOD to another. Additionally, our renderings look much like billboards with no sense of volume inside the bounding boxes. One option to solve these problems uses volumetric textures, as proposed by Meyer and Neyret [Meye98]. Their solution divides the volume of a box into parallel slices, each with an associated texture. It requires, however, a much larger amount of texture memory, thus reducing the number of trees that can be represented.

Instead we propose using six diagonally aligned textures per box. The textures are arranged as illustrated in Figure 6. Our approach improves the sense of volume inside the boxes without requiring additional storage. We also reduce the popping artefacts by alpha-blending the texture-mapped polygons associated to a box. Alpha blending is also used to achieve smooth transitions between LODs.

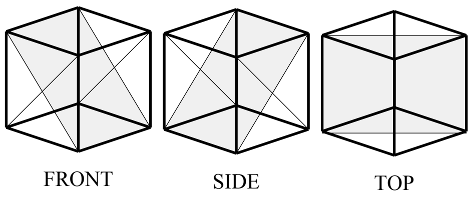


Figure 6. Diagonally aligned textures.

5. RESULTS

We developed two applications to test our method. The first application supports modeling plants and trees using L-systems. It also generates the sets of images that represent the leaves of the trees at different LODs. The second application is an interactive renderer of our tree models.

We run our tests on a Pentium III at 500 MHz with 256 Mbytes of RAM and a GeForce 2 MX graphics card. Generating all the images associated to a tree takes less than one minute. The images occupy between 4 and 6 Mbytes of storage. If we use a 32 Mbyte graphics card, we can store up to 8 complete tree models. Note that this is not necessary as we only need to store the LODs currently in use at any time. A model can be progressively transmitted by sending the definition file of the L-system followed by the images compressed and sorted by LOD.

Figure 7 shows a tree model rendered using geometry and the three image-based approaches discussed in this paper. Note that there are barely any visual differences between the first and the last image. Still, the leaves in the first image require 20000 texture-mapped polygons, while the last image only requires 50 polygons.

Figure 8 contains a close view of a tree. The leaves closest to the viewer are represented by one polygon each. All the other leaves are rendered using pre-calculated images. Figure 9 illustrates another feature of our method, namely, that the models can be rendered from any direction without loss of realism. Finally, Figure 10 compares the frame rate of our approach with the frame rate of a geometry renderer. Note that our approach can render scenes with a couple of hundred trees at interactive rates.

6. CONCLUSIONS AND FUTURE WORK

In this paper we present an image-based multiresolution approach to interactive tree rendering. Our method builds the LODs of a tree by grouping branches into a bounding box hierarchy. Associated to each bounding box we store six images representing the leaves contained in the box. The amount of memory used by this image representation is independent of the number of leaves of the tree. Our method automatically builds these tree models in less than one minute.

At rendering time we select and display the right LODs and related images using a viewer distance criterion. For the same tree model, leaves closer to the viewer may be rendered at a higher LOD than leaves located further from the viewer. Unlike previous approaches, our tree models are view-independent and we can render hundreds of them at interactive rates. Our models can also be progressively transmitted. The images that belong to the different LODs can be sent separately, starting with the coarsest LOD. In a low bandwidth scenario, we can send an L-system definition file occupying a just a few bytes. After that we can send the images or we can generate them locally at their destination.

We are currently working on a multiresolution tree model that combines our image-based representation for the leaves with a procedural multiresolution representation for the trunk and branches [Lluc03]. Our goal is to apply the resulting modeling techniques to interactive computer graphics applications like computer games, simulation, interactive walkthroughs and fly-by's, and virtual and augmented reality.

7. ACKNOWLEDGEMENTS

This work was partially supported by grant TIC2002-0416-C03-01 of the Spanish Ministry of Science and Technology, and by a grant of the *Programa de Incentivo a la Investigación 2003* of the Universidad Politécnica de Valencia.

8. REFERENCES

[Jaku00] A. Jakulin. Interactive Vegetation Rendering with Slicing and Blending EUROGRAPHICS 2000

[Lint99] B. Lintermann and O. Deussen. Interactive modelling of plants. IEEE Computer Graphics and Applications,19(1), January/February 1999.

[Lluc01] J. Lluch, M. J. Vicent, S. Fernández, C. Monserrat. The Modelling of branched structures using a single polygonal mesh. Proceedings on IASTED International Conference Visualization, Imaging and Image Processing Conference, September 2001

[Lluc03] J. Lluch, E. Camahort, R.Vivó, "Procedural Multirresolution for Plant and Tree Rendering", 2nd International Conference on Virtual Reality, ComputerGraphics, Visualization and Interaction in Africa. AFRIGRAPH 2003. ACM SIGGRAPH, pp 31-38, 2003.

[Meye98] A. Meyer and F. Neyret. Interactive volumetric textures. Eurographics Rendering Workshop 1998, 157–168, June 1998.

[Prus90] P. Prusinkiewicz, A. Lindenmayer, "The algorithmic beauty of plants", New York, Ed. Springer-Verlag, 1990

[Schm97] D. Schmalstieg, M. Gervautz, Modeling and Rendering of Outdoor Scenes for Distributed Virtual Environments, Proceedings of ACM Symposium on Virtual Reality Software and Technology 1997 (VRST'97), pp. 209-216,Lausanne, Switzerland, Sep. 15-17, 1997

[onyx03] www.onyxtree.com

[deRe88] P. de Reffye, C. Edelin, J. Françon, M. Jaeger and C. Puech, Plant Models Faithful to Botanical Structure and Development, in Computer Graphics (SIGGRAPH '88 Proceedings), 22(4), pp. 151-158, August 1988

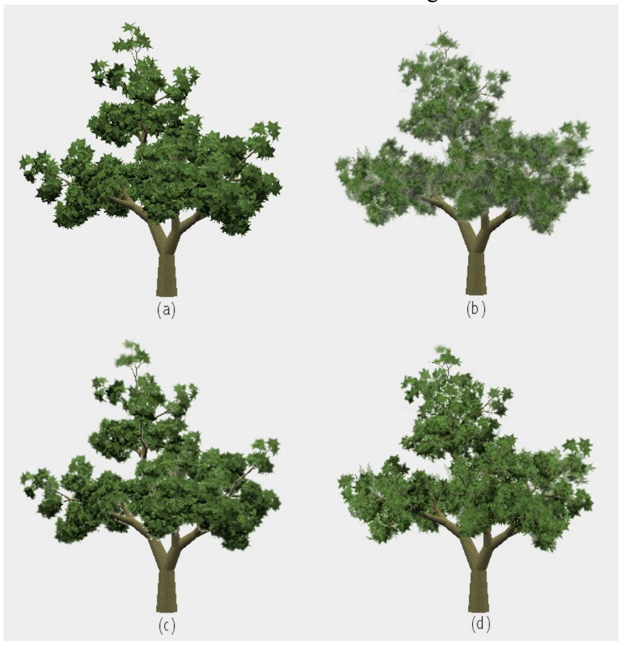


Figure 7. A tree whose leaves are rendered using (a) geometry only - 20.000 polygons, (b) axis aligned textures with 20 pol., (c) volumetric textures with 200 pol., and (d) diagonally aligned textures with 50 pol.

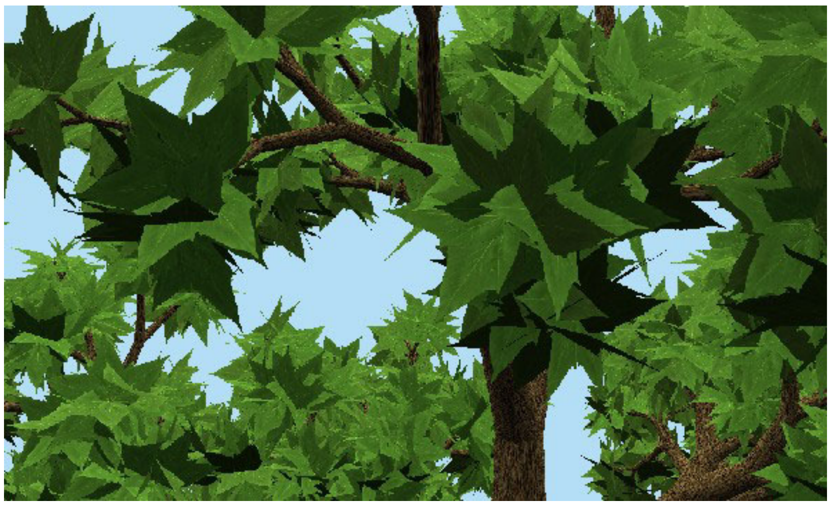


Figure 8. A close view of a tree

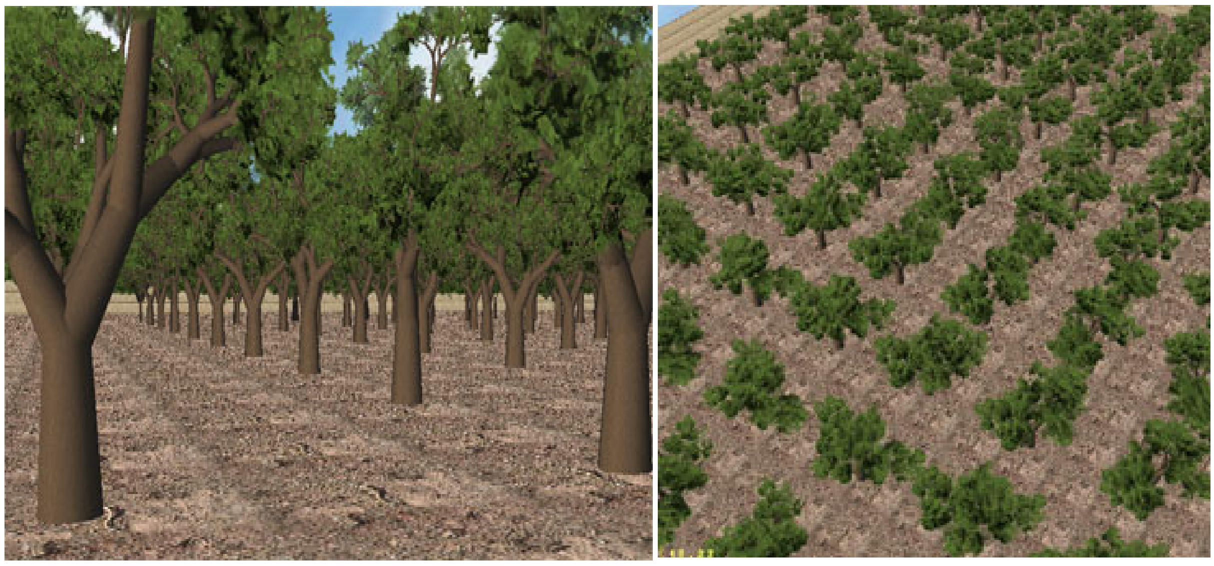


Figure 9. Trees rendered from different viewpoints.

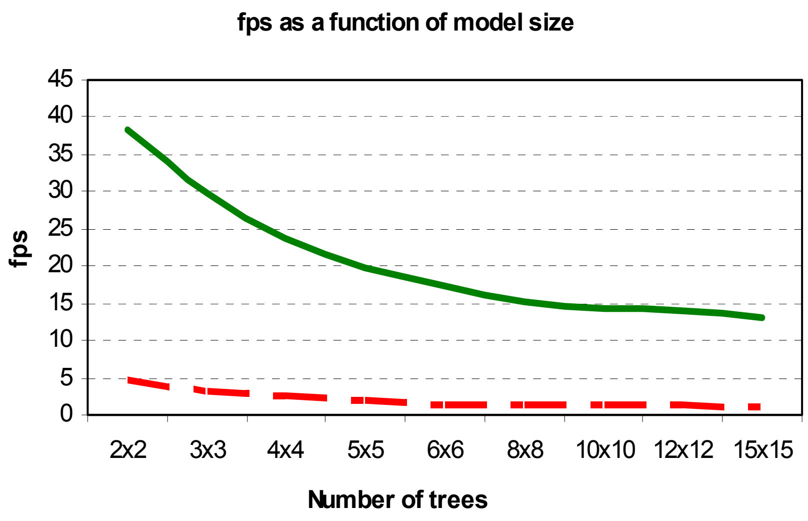


Figure 10. Frame rate as a function of the number of trees when rendering using geometry only (red) and using our image-based multiresolution approach (green).