# FAST RENDERING OF LEAVES

C. Rebollo, J. Gumbau, O. Ripolles, M. Chover, I. Remolar
Dept. Lenguajes y Sistemas Informáticos
Universitat Jaume I
Castellón, Spain
email: {rebollo,jgumbau,oripolle,chover,remolar}@uji.es

**ABSTRACT**
One of the main problems of rendering outdoor scenes in real time is the representation of trees and plants. The modeling tools for vegetal species generate polygonal models with such a great geometric complexity that they need efficient techniques for achieving an interactive visualization. In this article we present a LOD model based on the geometric representation of leaves that eliminates bus traffic when changing the level of detail and enables instancing as many trees as desired. This model allows for a highly realistic visualization, skeletal animations for wind effects and also the use of traditional illumination techniques for polygonal models. The proposed representation permits to adapt automatically and in a continuous manner the level of detail, obtaining view-dependent resolutions. The basic idea consists in initially uploading to the GPU the vertices of the high resolution model and obtaining the desired approximation by rendering the appropriate vertices. This model improves the spatial cost and the rendering speed of any previous model based on the geometric representation of leaves. With respect to the models based on images, our solution offers a higher visual quality and also the possibility of including animations.

**KEY WORDS**
Real-time rendering, level of detail, natural phenomena

## 1 Introduction

Interactive visualization of trees and plants poses nowadays one of the hardest problems in representation of natural scenes. The great complexity of these models makes it necessary to use techniques that simplify the problem, making it accessible and exploiting to the maximum the features of the graphics hardware. Following this idea, the most important models that have been developed in recent years can be classified into two general groups: geometry-based models and image-based models.

### 1.1 Image-based models

These solutions are usually based on using billboards, volumetric textures or billboard clouds.

Employing billboards for the representation of trees is a technique that offers good visual results for objects which are far from the viewer. Despite these good characteristics, this solution suffers from noticeable effects of parallax [1]. It is also possible to find in the literature works that use textured polygons and divide the trees in slices [2] or in different sets of leaves [3], and then they choose the most appropriate ones depending on the characteristics of the scene. These algorithms present ghosting effects and aliasing when the viewer is close to the trees.

Figure 1. A close-up view of the *Aesculus hippocastanum*.

Algorithms based on volumetric textures are not adequate for representing trees in short distances [4, 5, 6, 7], but offer good results for visualizing dense forests from a bird's-eye view.

Finally, it is important to comment on billboard clouds [8], which suffer from aliasing when the tree is close to the viewer and also present different problems for animating leaves. These aliasing effects might be avoided by applying several textures, but at the expense of overloading the rasterization stage of the GPU [9]. The generation of forests is based on a set of discrete levels of detail that might produce popping effects.

### 1.2 Geometry-based models

It is possible to distinguish two different tendencies between these models: using multiresolution models over the polygonal model or resorting to primitives like points or lines.

On the one hand, multiresolution models entail a high data traffic between the CPU and the GPU to update the level of detail of the objects [10]. In addition, these approaches are not useful for instantiating several times the same tree because of the data structures they work with. Some authors have improved these solutions, dividing the foliage into different sets of leaves and updating only the affected groups when changing to a different resolution [11].

On the other hand, the approaches based on points and lines often represent trunks with lines and leaves with points, adapting the density and the size of the points depending on the scene requirements [12, 13]. This kind of representation is effective for distant objects, but visually unacceptable for close objects.

Other works are based on the representation of trees with points and triangles. Some authors sample each object to obtain a list of points [14]. In runtime, and according to the projected size of each object, it is decided whether the model should be visualized using triangles, or whether it is better to use the point-based approach. It is also possible to find models that organize the triangles and the points in a regular spatial structure to use the different levels of detail established during the visualization process [15].

### 1.3 Model overview

Given that existing solutions present different problems, we decided to develop a new multiresolution model based on the geometric representation of the foliage, the *Fast Leaves Model (FLM)*. On the one hand, discrete algorithms suffer from disturbing popping artifacts. On the other hand, the main issue with applying continuous multiresolution techniques to solve that problem is the cost of recalculating the geometry every time a change in the level of detail happens.

In order to overcome both problems, we have created a continuous and view-dependent model that reduces the extraction time to the minimum by needing only simple formulas to obtain the desired approximation. This way, the cost of changing the level of detail is almost inexistent. For rendering the different resolutions, it is only necessary to initially upload the vertices of the foliage to the GPU, and traverse the same array with different increments depending on the level of detail we want to render. This initial set of vertices of a tree species might be used by as many instances as desired. On top of that, the use of geometry offers realistic representations in short distances. Figure 1 presents the quality of a close-up view of a tree rendered using our solution.

The presented representation is easily integrated in the graphics pipeline, allowing us to apply traditional techniques for collisions and for animating and illuminating the trees. In this case, the animation of the leaves is carried out in the GPU through skeletons. The illumination and the cast shadows are generated in the rasterization stage by means of shadow maps.

It is important to mention that we will use two different multiresolution models for managing the level of detail of a tree. The trunk and the branches will be rendered using a multiresolution model for manifold meshes, which is based on the use of triangle strips. This model, known as *LODStrips* [16], is a continuous multiresolution model of uniform resolution that exploits the possibilities of the graphics hardware. The leaves will be rendered using the multiresolution model we are presenting in this paper, whose basic characteristics will be explained in the following sections.

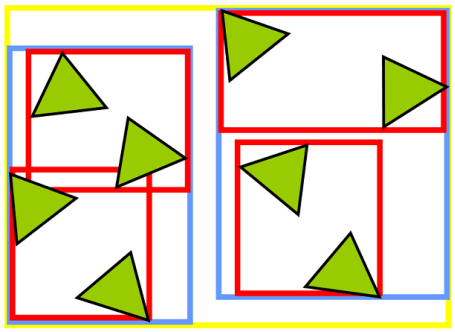

Figure 2. Hierarchical structure of a set of leaves.

## 2 Fast leaves model

The basic idea of the model is to use the set of vertices of the original model to represent the leaves through all the approximations. With this objective in mind, the first step consists in transforming the set of leaves initially defined as quadrilaterals into triangles. This conversion reduces the size of the array of vertices in a 25% and, on top of that, the construction of the model is also simplified.

The second step consists in rearranging the vertices by means of a hierarchy of bounding boxes. The next step is to find a simplification sequence that fits our representation. Finally, the vertices that define the triangles, with their texture coordinates and normal vectors, will be uploaded to the GPU for obtaining an interactive visualization. These steps will be explained in the following sections.

The input trees of the multiresolution model presented may be developed with any CAD application that models vegetal species. In this paper, the models used for the validation of the multiresolution model have been built using XFrog [17]. This commercial software represents the trunks and the branches as triangle meshes, while leaves are rendered as textured quadrilaterals. Animations to generate wind effects can be obtained by adding skeletons to the trees that have been constructed.

### 2.1 Sorting algorithm

To obtain an adequate simplification sequence, we will construct a hierarchical structure of axis-aligned bounding

boxes (AABB) (see Figure 2). This structure is necessary for our simplification process, but will also be useful for the frustum culling. This algorithm could be easily extended to use oriented bounding boxes (OBB), which would offer more precise simplifications.

```
Simplify (leaves, AABB) {
  if (leaves>2) {
    Axis=LongerAxis(AABB);
    Divide(Axis,leaves,leavesL, leavesR);
    AABBL=CalculateAABB(leavesL);
    AABBR=CalculateAABB(leavesR);
    if (MoreVisible(LeavesL)) {
      Simplify (LeavesL,AABBL)
      Simplify (LeavesR,AABBR)
    }
    else {
      Simplify (LeavesR,AABBR)
      Simplify (LeavesL,AABBL)
    }
    Write (leaves);
  }
}
```

Figure 3. Pseudo-code of the simplification algorithm.

The algorithm, presented in Figure 3, works as follows:

- Initially, the AABB adjusted to the original set of leaves is calculated.
- Following the longer diagonal of the AABB, the set of leaves is divided into two subsets with an even number of leaves.
- The AABB of those two subsets of leaves is calculated, and the algorithm repeats this subdivision process recursively.
- Finally, when the recursion has finished and there are only two leaves left, the sequence of leaves is constructed.

Every time we obtain two subsets after dividing the previous one, the algorithm has to choose the subset to start the recursion with. The criterion followed for this decision is based on the visibility of each one, as the algorithm will choose the subset of leaves which is more visible. This choice is made using a metric based on the point of view [18]. The algorithm orders the leaves according to their importance. The most visible leaves of the tree will always appear in the beginning of the sequence, and the inner ones will be at the end and, therefore, will be simplified first.

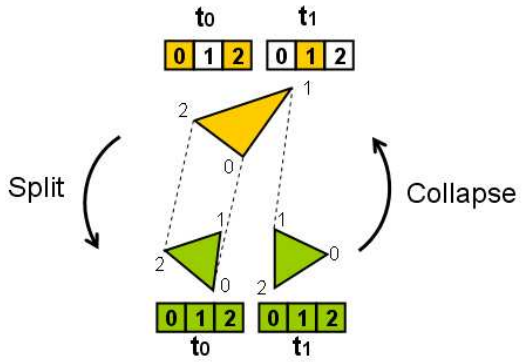


Figure 4. Simplification operation.

### 2.2 Simplification algorithm

Once the sequence of ordered vertices has been obtained, the process is ready to carry out the simplification step. With the intention of minimizing the information that is necessary to represent the different levels of detail, the simplified leaves must share the vertices of the original ones. The leaf simplification operation takes two initial leaves and obtains an only one (see Figure 4). Following the example given in that figure, the two initial leaves would be defined by the triangles $t_0$ and $t_1$. After the collapse operation, the resulting leaf would be rendered using vertices 0 and 2 from the first leaf ($t_0$), and vertex 1 from the second leaf ($t_1$). This way, the new leaf will be rendered using the same array of vertices, but making increments of 2. It is important to mention that, in this example and in the following ones, values 0, 1 and 2 that are used to represent a triangle are referred to the first, second and third vertex of that triangle respectively. This simplification, although not being as precise as the one presented in [19], is perfectly adapted to our representation, and produces good visual results.

Figure 5 shows the resulting sequence of leaves obtained after the simplification algorithm has been applied. The leaves coloured in green ($t_0$ to $t_7$) represent the original set of leaves following the simplification order. The rest are the leaves obtained after different collapse operations. All the leaves will be rendered using the array of vertices belonging to the high resolution model, choosing the correct increment to traverse the array. For obtaining the leaves of the initial set, it is necessary to use an increment of 1. Every time we want to obtain the leaves from a higher level of the hierarchy, we must multiply the increment by two.

### 2.3 Rendering algorithm

One of the main goals of the rendering algorithm we have developed is to offer a continuous transition between the different levels of detail. If we need to move to a coarser level of detail, the rendering algorithm eliminates two

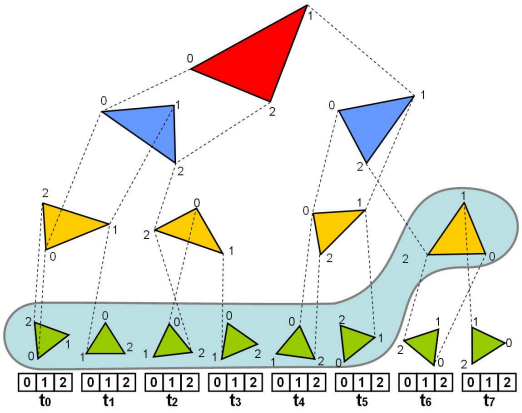


Figure 5. Sequence of original leaves and binary tree obtained after the simplification.

leaves of the current approximation, and adds the leaf that replaces them. On the contrary, if we wanted to move to a more detailed approximation, the rendering algorithm would apply the inverse operation.

In order to render the desired approximation, the array of vertices is traversed just once, using the correct increments to choose the appropriate vertices. As we have commented before, every level of our hierarchy of leaves needs a different increment. In general, the algorithm will render leaves from two different levels and therefore it will be necessary to use two different increments: one for the first part of the array, and the other one for the rest of the vertices. For example, to eliminate the right-most leaf of the tree represented in Figure 5, it is enough to draw the vertices of the triangles $t_0$ to $t_5$ with an increment of 1, and the vertices of the triangles $t_6$ and $t_7$ with an increment of 2 (drawing vertices 0 and 2 of the triangle $t_6$, and 1 of $t_7$).

As it has been commented before, we have developed our multiresolution model with the intention of offering variable resolution of the crown of the trees. The initial division of the foliage is made using the hierarchy of AABB obtained in the simplification process, which allows us to organize the set of leaves in different groups, each one having their own array of leaves. This way, it is possible to apply different resolutions to different groups of leaves, visualizing with more detail the zones of the tree located closer to the observer, while the back zones have a lower detail. The criterion used to select the suitable level of detail is the Bounding Box projection. This error is calculated as the relation between the projected area of the different bounding boxes and the total area of the screen. The result will be a value between *0* and *1*, where *0* would refer the coarsest level of detail and *1* would be the highest detail.

Figure 6 presents the pseudo-code of the rendering algorithms. This algorithm takes as input the data that defines the leaves at their highest resolution. Every set of leaves will be rendered if it is visible and will adapt its level of detail according to its Bounding Box projection. The *RenderLeaves* function will traverse the array of vertices using two increments, as in the worst case we will draw leaves from two different levels of the binary tree. For the correct performance of the algorithm, it is necessary to know the number of vertices to draw with each one of the increments (#v1 and #v2), the value of the increment (step) used in the first section (the second increment is obtained multiplying by two the first one) and the position in the array where we must switch between those two different increments (the first increment is applied starting from the beginning of the array). These values are obtained with the *Compute* function that applies straightforward formulas with constant execution time. The *Render* function uses these parameters to tell the GPU which vertices to render, drawing the foliage with at most two GPU calls.

```
RenderFoliage (Foliage) {
  for (∀ LeaveSet ∈ Foliage) {
    if (notCulled(LeaveSet)) {
      LOD=projectedAABB(LeaveSet);
      RenderLeaves(LeaveSet,LOD);
  }
}

RenderLeaves (LeaveSet,LOD) {
  if (LODChange) {
    Compute(#leaves,#v1,#v2,step,init2);
    Render(leaves,0,#v1,step);
    if (#v2 != 0) {
      Render(leaves,init2,#v2,step*2);
    }
  }
}
```

Figure 6. Pseudo-code of the rendering algorithms.

In Figure 7 it is shown an example of the simplification of an initial set of seven leaves. This example will allow us to analyze the way this algorithm works. The trace of the rendering process of the different levels of detail is presented in Figure 8, where for each level of detail are shown the drawn triangles, and the two different sections of the array that are traversed (the vertices that are drawn have been coloured). When the number of leaves of an approximation is odd, the last leaf is no taken into account and it is not rendered, and the leaves that are simplified are the two leaves that come before. For every step we can see the leave that appears, the two leaves that disappear and also the vertices that are involved. The traversal of the different sections of the array of vertices is done with the appropriate increment. As we can see, the number of leaves visualized decreases in one every time we change to a coarser level of detail. All the information that is necessary to traverse the array of vertices is shown in Table 1.

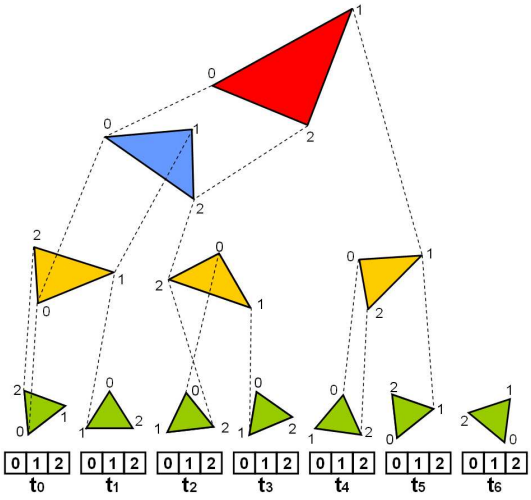

Figure 7. Example of the binary tree obtained after the simplification of a set of seven leaves.

| #leaves | #v1 | #v2 | step | step*2 | init2 |
|---|---|---|---|---|---|
| 7 | 21 | 0 | 1 | null | 0 |
| 6 | 15 | 3 | 1 | 2 | 12 |
| 5 | 9 | 6 | 1 | 2 | 6 |
| 4 | 3 | 9 | 1 | 2 | 0 |
| 3 | 9 | 0 | 2 | null | 0 |
| 2 | 3 | 3 | 2 | 4 | 0 |
| 1 | 3 | 0 | 8 | null | 0 |

Table 1. Trace of the algorithm.

## 3 Results

In order to prove the validity of the model, our aim has been to verify not only the low storage cost and the fast extraction and visualization process, but also the quality of the resulting approximations. All the experiments were carried out using Windows XP on a Dell PC with a processor at 2.8 Ghz, 2 GB RAM and an nVidia GeForce 7800 graphics card with 256MB RAM.

Figures 9 show different levels of detail (100%, 50% and 25% of the highest detail) of one of the tree species used in our tests. These approximations prove that the simplification maintains the shape and the appearance of the original tree.

With the aim of showing how the visual quality is maintained with FLM, Figure 10 gives visual results of a forest formed by 100 trees of eight different species amounting for more than 15 millions of polygons. In this Figure, the first image represents the forest obtained with our multiresolution model, the second one offers the highest detailed view of the scene, and the third one shows the difference between those images. It can be seen how, despite reducing the polygons rendered in a 30%, the visual

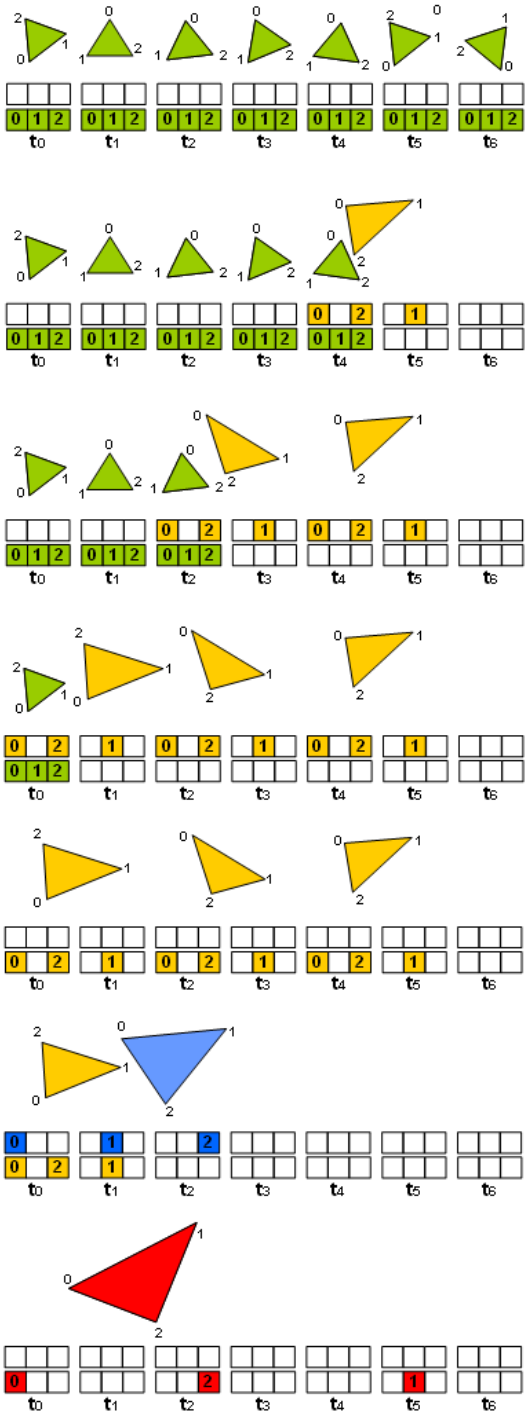

Figure 8. Trace for the example set of leaves.

(a) 38124 leaves. (b) 19062 leaves. (c) 9531 leaves.

Figure 9. Results obtained for the tree model *Oela Europaea*.

appearance is assured as much as possible, as only those trees whose projected Bounding Box go beyond a threshold will reduce their level of detail. Thus, it can be observed how the instances located in the background have a higher error and how this error is almost visually imperceptible.

It is important to underline that we have maintained the same visual quality in the temporal tests performed.

### 3.1 Storage cost

The use of the approach presented in this paper *(FLM)* offers a low storing cost as it does not store the indices of the leaves. This way, FLM just needs the data related to the vertices and a small piece of information for managing the level of detail. If we decided to render more than one instance of the same tree, the vertices information might be shared between all the instances. It is important to comment that other models like [11] increase the storing cost as they need to store the original geometry data and a noticeable quantity of additional information. On top of that, instantiation is not well supported by these models as, despite sharing the vertices array, all the information related to leaves and to manage the level of detail must be replicated for every instance we might want to use.

### 3.2 Temporal cost

To test the rendering speed of the approach we are presenting, we decided to render a forest formed by a different number of instances of the *Fraxinus Ornus Manna* tree, which is composed of more than 28,000 leaves. In these tests, shown in Table 2, we compare the speed of our solution against the results of rendering the whole geometry without any kind of level-of-detail management. It can be easily seen how FLM increases the rendering speed, obtaining nearly three times more FPS for all the cases.

| | Number of instances | | | | |
|---|---|---|---|---|---|
| | 25 | 50 | 75 | 100 | 200 |
| No LOD | 45 | 21 | 14 | 11 | 5 |
| FLM | 115 | 52 | 38 | 27 | 16 |

Table 2. FPS of scenes with different number of trees.

## 4 Conclusions and future work

In this paper we have presented a new continuous multiresolution model for the representation of the crown of a tree, which is adapted perfectly to the characteristics of the current graphics hardware. The extraction process, which is one of the main problems of continuous multiresolution models, has been reduced to just straightforward formulas. In addition, the data that must be sent through the bus every time a change in the level of detail happens is eliminated. This variable resolution model adjusts the number of leaves in accordance to the position and distance to the viewer. Among its main features, we want to underline the possibility of using the same data structures in the GPU for different instances of the same species, allowing for the creation of multiple instances of the same tree species without increasing the memory cost.

As it has been defined as a model based on triangles, it is perfectly adapted to the graphics pipeline allowing skeletal animations to generate wind simulation and other advanced effects of illumination that can be obtained with pixel shaders.

Nowadays, we are working in a scene manager that decides for every instance which is the most adequate level of detail, as well as the quality of the approximations used to calculate the cast shadows and when should they be generated.

(a) Scene with FLM managing the level of detail.

(b) Scene with the full detailed geometry.

(c) Difference image between a) and b).

Figure 10. Visual comparison of the FLM model.

## Acknowledgements

This work was supported by the Spanish Ministry of Science and Technology with grants TIN2004-07451-C03-03 (MATER) and FIT-350101-2004-15, the European Union in the project IST-2-004363 (GameTools: Advanced Tools for Developing Highly Realistic Computer Games) and FEDER funds.

## References

[1] Speedtree, interactive data visualization inc. `http://www.idvinc.com/speedtree/`, 2005.

[2] A. Jakulin. Interactive vegetation rendering with slicing and blending. In A. de Sousa and J.C. Torres, editors, *Proc. Eurographics 2000 (Short Presentations)*. Eurographics, 2000.

[3] J. Lluch, E. Camahort, and R. Vivo. An image based multiresolution model for interactive foliage rendering. *Journal of WSCG'04*, 12(3):507–514, 2004.

[4] A. Meyer and F. Neyret. Interactive volumetric textures. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, pages 157–168. Springer Wein, 1998.

[5] A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*. Springer-Verlag, 2001.

[6] P. Decaudin and F. Neyret. Rendering forest scenes in real-time. In *Rendering Techniques '04 (Eurographics Symposium on Rendering)*, pages 93–102, 2004.

[7] A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23(3):720–727, 2004.

[8] A. Fuhrmman, E. Umlauf, and S. Mantler. Extreme model simplification for forest rendering. In P. Poulin E. Galin, editor, *Eurographics Workshop on Natural Phenomena*, pages 57–66, 2005.

[9] I. Garcia, M. Sbert, and L. Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *Proc. Eurographics 2005 (Short Presentations)*. Eurographics, 2005.

[10] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage. *Journal of WSCG'03*, 11(2):370–378, 2003.

[11] C. Rebollo, I. Remolar, M. Chover, and O. Ripolls. An efficient continuous level of detail model for foliage. *Winter School of Computer Graphics*, 2006.

[12] P. Oppenheimer. Real time design and animation of fractal plants and trees. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 55–64. ACM Press, 1986.

[13] J. Weber and J. Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128. ACM Press, 1995.

[14] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 219–226. IEEE Computer Society, 2002.

[15] G. Gilet, A. Meyer, and F. Neyret. Point-based rendering of trees. In P. Poulin E. Galin, editor, *Eurographics Workshop on Natural Phenomena*, 2005.

[16] J. F. Ramos and M. Chover. Lodstrips: Level of detail strips. In *International Conference on Computational Science*, pages 107–114, 2004.

[17] Greenworks: Organic software. `http://www.greenworks.de/`, 2005.

[18] P. Castell, M. Sbert, M. Chover, and M. Feixas. Techniques for computing viewpoint entropy of a 3d scene. In *Lecture Notes in Computer Science 3992*, pages 263 – 270, 2006.

[19] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage. In I. Navazo and Ph. Slusallek, editors, *Proc. Eurographics 2002 (Short Presentations)*. Eurographics, 2002.