A Clustering Framework for Real-Time Rendering of Tree Foliage

C. Rebollo, I. Remolar, M. Chover, J. Gumbau, O. Ripollés
Departamento Lenguajes y Sistemas Informáticos
Universitat Jaume I, 12071 Castellón, Spain
Email: {rebollo,remolar,chover,gumbau,oripolle}@uji.es

Abstract— **Real-time rendering of vegetation is currently a problem in need of a solution. The lack of plants and trees reduces the realism of outdoor scenes. The large number of polygons that form this kind of objects implies that current hardware cannot achieve interactive rendering of outdoor scenes. This paper deals with this problem and it presents a multiresolution scheme that allows us to represent the whole geometry of the trees and forests using both uniform and variable levels of detail. The method presented here models the trees using two multiresolution approaches, due to the different characteristics of the geometry that forms them. The trunk is modelled with a solution oriented towards representing continuous meshes, and the foliage is modelled with the multiresolution model *Level of Detail Foliage*, presented in a previous work. Both of them have been designed to take advantage of the graphics hardware by adapting the data structures and the rendering algorithms to make the visualisation time efficient. The main characteristic of *Level of Detail Foliage* is the fact that it classifies the leaves that form the foliage in independent clusters in order to improve the visualisation time. In this paper, it has been efficiently implemented and extended to allow us to change the level of detail in a variable manner, by adapting the resolution of the crown of the tree to certain criteria determined by the application.**

Index Terms— **real-time rendering, clustering of leaves, tree representation, multiresolution, hardware graphics, level of detail.**

I. Introduction

Interactive applications currently developed are usually set in outdoor scenes. In these environments, trees and plants are necessary to make the scenes more similar to the real world. Many works on modelling vegetal species have appeared up to now, being possible to obtain very realistic tree models. However, the problem arises when real-time rendering is required by the application, as the more realistic the vegetation objects are, the greater the number of polygons they contain. Figure 1 shows an example of a tree generated by one of the most used commercial application, Xfrog [1]. It can be appreciated how the representations obtained can be highly realistic.

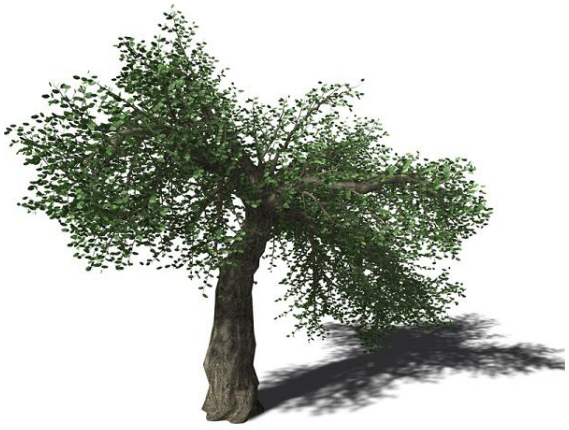

Figure 1. Tree *Quercus Suber* modelled using Xfrog. 20.281 leaves.

Depending on the technique used to solve this problem, methods can be divided into two important groups: image-based and geometry-based rendering. The model presented in this paper is categorized within the second group. Geometric representation has many advantages, the most important of which is that trees do not lose realism even when the camera is extremely close to the object. Another important advantage is that geometry can be stored either in the main memory or directly in the graphics card, thus taking advantage of current graphics hardware. Moreover, using geometry to represent objects makes it possible to obtain shadows and different lighting effects, as well as greater acceleration in rendering. Geometry-based approaches have used several techniques to achieve interactivity, such as replacing the basic display primitive (triangles) by points and lines, or using multiresolution modelling techniques.

Geometric models which apply multiresolution techniques have proven to be a good solution for visualising objects made up of a vast number of polygons in real-time applications. These models adapt the geometric detail of the objects to the capacity of graphics systems. Using this technique, objects are represented by means of multiple resolutions, with varying complexities, called levels-of-detail (LoDs). The application can visualise the object using the most suitable LoD and therefore avoid, for instance, wasting time on visualising imperceptible

This paper is based on "Hardware-Oriented Visualisation of Trees" by C. Rebollo, I. Remolar, M. Chover, J. Gumbau, which appeared in the Proceedings of the 21st International Symposium on Computer and Information Sciences (ISCIS), Istanbul, Turkey, November 2006.

This work was supported by the Spanish Ministry of Science and Technology (TIN2004-07451-C03-03 and FIT-350101-2004-15), the European Union (IST-2-004363) and FEDER funds.

details. Multiresolution models can provide levels of detail of uniform resolution or levels of detail of variable resolution.

General level-of-detail models do not work properly with the representation of trees because of the characteristics of their geometry [2]. They can be easily applied to the trunks, as they are modelled as continuous meshes, but are not suitable for the set of isolated polygons which is used for the foliage. This paper presents a new hardware-oriented multiresolution approach to represent the geometry of these vegetal species. Two different multiresolution models are used to represent the tree objects: *LodStrips* [3] for the trunk and branches, and *Level of Detail Foliage* LoDF [4] for the foliage. Both of them have been designed within a hardware-oriented approach, thus considerably reducing the visualisation time of the LoDs that are required. In order to adapt the resolution of the different parts of the foliage depending on certain criteria, the data structures and retrieval algorithms of LoDF have been both extended. Finally, these models are compared to the ones used in [5] to represent trees, showing how this approach reduces the extraction and visualisation time even when hardware storage is not applied.

This paper presents the following structure. After reviewing previous work in section II, the multiresolution model used to represent the trunk and branches is set out in section III. Next, section IV analyses the solution proposed to represent the foliage. The data structure designed for this model and the retrieval algorithms for uniform and variable levels of detail of the foliage are presented. In section V, the forest representations are discussed. Section VI offers the results of comparing this new method against the one used in [5] and, finally, in Section VII some ideas for future research are outlined.

II. Related Work

Extensive research has been carried out to offer real-time visualisation of detailed vegetal species, adapting the number of polygons used to represent those plants to the requirements of graphics hardware. As it was said in the previous section, research into interactive visualisation of vegetal species can be grouped into two broad directions: works that use images or works that use only geometry to represent the plants.

- **Image-based rendering**. This is one of the commonest methods to represent trees because of its simplicity. Impostors [6] are the most popular example of image-based rendering. This method replaces the geometry of the object with an image of it textured on a polygon immersed in the scene. Nevertheless, it presents some disadvantages, such as, for example, the loss of realism when the object is close to the viewer. Max [7] adds depth information to the precalculated images. This information allows them to recalculate different views from the stored images of the scene. Other authors [8] [9] obtain 2D images from volumetric textures and combine them depending on the position of the camera. Some works, such as Shade et al. [10] and Remolar et al. [11], divide the scene into zones depending on the distance from the object to the viewer. Objects farther away from the camera are represented by an image and objects near the viewer are depicted by geometry. García et al. [12] solve the parallax problem by using textures that group sets of leaves. Since 2005, some works have appeared that represent the tree using billboard clouds, as those presented by Behrendt et al. [13], Fuhrmann et al. [14] and Lacewell et al. [15].
- **Geometry-based rendering.** This approach does not lose realism when the viewer moves towards the model, but the number of polygons that form the tree objects makes it necessary to use certain techniques to obtain interactive visualisation. Most of the works published to date change the display primitive for points or lines [16] [17] [18]. Works such as the ones presented by [2] [19] allow us to interactively adapt the number of points depending on the importance of the object in the final rendered image.
  In recent years, several works based on multiresolution models have appeared. Meyer et al. [20] and Lluch et al. [21] use a representation based on multiresolution models of images. Remolar et al. [11] presents a multiresolution representation of the tree based exclusively on isolated polygons. Authors have called this representation *VDF*. It is focused on tree foliage and it adapts the number of leaves that form the foliage in real time. Rebollo et al. [4] [22] improve this representation adapting the data structures to the graphics hardware.

III. Trunk Representation

The trunk and the branches are represented by a continuous mesh, where the polygons are connected to each other. In our scheme, they are modelled using *LodStrips* [3]. Every multiresolution model needs a simplification algorithm to obtain the sequence of operations that will enable a progressive refinement of the original mesh. In this case, the selected algorithm will be QSlim [23], which uses the vertex collapse as its simplification operation. LodStrips is entirely based on optimised hardware primitives, triangle strips. This drawing primitive offers an efficient representation of the connectivity of a mesh and reduces the amount of information that is sent to the hardware, thus resulting in an improvement of the performance. LodStrips uses this primitive for both storing and rendering the model, and manages the level of detail by performing fast strip updating operations. This continuous approach noticeably improves on previous models, in terms of storage and visualisation cost. The basic data structure of this model is composed of three elements:

- A set of multiresolution strips.
- An array of vertices, which store a pointer to the vertex they are collapsed to.

- A structure to store all the information related to know how to update the strips to reflect a LOD change.

The main characteristics of LodStrips that have encouraged its selection as a continuous multiresolution model for the trunk are:

- Smooth transitions between levels of detail, avoiding visual artifacts.
- Exploitation of mesh connectivity by means of triangle strips, resulting in less storage needs and a faster rendering.
- A fast level of detail extraction routine, which is key for an efficient multiresolution model.

IV. Foliage Representation

The common representation of foliage is based on isolated polygons that represent the leaves. In the presented scheme, it is represented using the multiresolution model LoDF. The main properties of LoDF can be summarized in:

- The foliage is divided into independent clusters.
- It is based on the Foliage Simplification Method [24].
- It allows us to manage only a subset of the leaves that form the multiresolution model.
- Different levels of detail can be visualised in real-time, with an uniform or variable resolution.
- The model takes advantage of the graphics hardware in order to reduce the extraction and visualisation time of a level of detail.
- LoDF stores appearance attributes.

A multiresolution representation is generally constructed from two main elements: the original geometry of the object and the different approximations given by a simplification method. LoDF is based on the *Foliage Simplification Algorithm* (FSA) [24], which provides the coarsest approximation F_{n-1} from the original foliage F_0. The basic simplification operation of this algorithm is *leaf collapse*: two leaves are collapsed into a new one. In the multiresolution model the inverse operation has been performed in order to increase the level of detail. The refinement operation has been called *leaf split*: one leaf is divided into the two leaves that formed it. These operations condition a hierarchical structure based on trees, as shown in Figure 2. The sequence of leaf-collapse operations obtained from FSA is processed to build the new multiresolution representation F^r (Figure 3).

The LoDF data organisation is a forest of binary trees, where the root-nodes are the leaves that form F_{n-1}, the coarsest approximation, and the leaf-nodes are the leaves of the original tree model, F_0. In the example, F_0 is formed by 9 leaves, and F_{n-1} is made up of 3 leaves.

Let F and F^r be the original and the LoDF representation. They can be defined as:

$$F = \{V, L\} \tag{1}$$

$$F^r = \{V^r, L^r\} \tag{2}$$

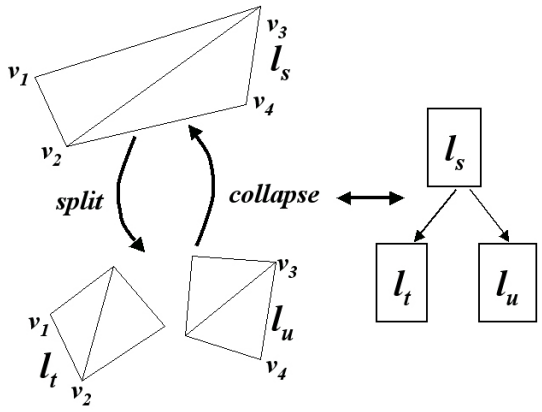

Figure 2. Example of a collapse and split operations and the resultant hierarchical structure.

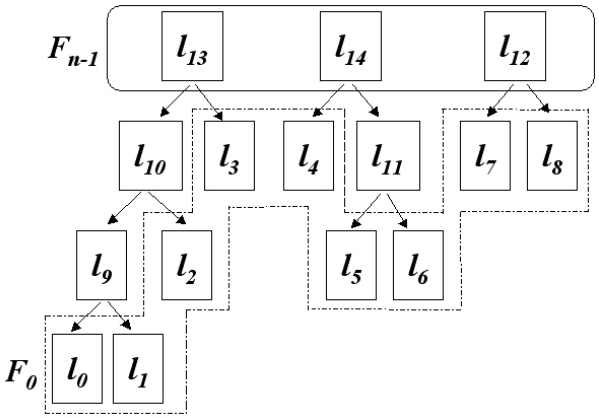

Figure 3. Example of an F^r structure.

being V and L the set of vertices and leaves forming the original object, and V^r y L^r the set of vertices and leaves that represent all the different levels of detail stored in the multiresolution structure F^r.

Regarding the vertices and the leaves of the object, it can be said that the most detailed representation F_0 is formed by the original vertices and the leaves

$$V = V_0 \tag{3}$$

$$L = L_0 \tag{4}$$

Due to the characteristics of the simplification method FSA, no new vertices are added to the multiresolution structure in order to represent the different levels of detail F_i. Then, it can be said that F^r is formed by:

$$V^r = V_0 \tag{5}$$

$$L^r = L_0 \cup l_0 \cup ... \cup l_{n-2} = L_0 \ \cup \ \bigcup_{i=0}^{n-2} l_i, n \geq 1 \tag{6}$$

where l_i is the leaf included in F_i in order to represent the F_{i+1} approximation.

The main characteristic of F^r is that the foliage is divided into independent clusters. As each simplification

operation creates a new leaf, each leaf can only belong to one binary tree. This way, every set of leaves in a binary tree of the data organisation forms one cluster. Following with the previous example, Figure 4 shows the three clusters representing it. These groups determine the data structure used in our model.

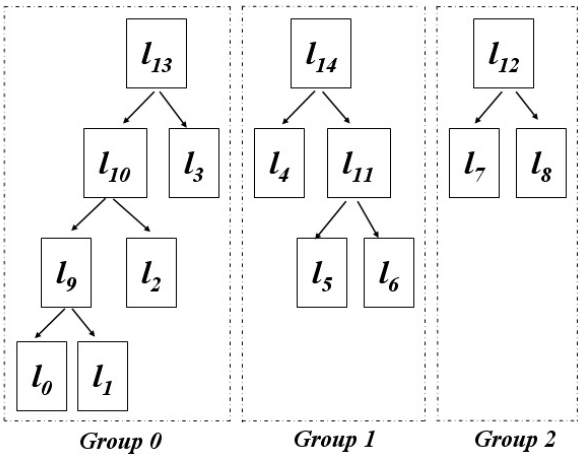

Figure 4. Data stored in clusters 0, 1 and 2 for the data structure shown in Figure 3.

A. Data Structure of the Foliage

The basic data structures of the LoDF model are shown next. In order to improve the visualisation time, some data are stored in the Graphics Process Unit, GPU. These data structure are:

```
struct Vertex {
        float coord; };
struct GroupHw {
        int vertices; };
struct Visualised_leaf {
        GroupHw *Groups; };
```

The rest of the data are stored in the Central Process Unit, CPU. They are:

```
struct Leaf {
        int vertices[4]; };
struct List {
        int leaf_number;
        int next; };
struct Group {
        int Init_group;
        int Active_leaves;
        struct List *Leaves; };
struct GroupLeaf {
        struct Group *Groups; };
struct Changed_group {
        int Group_number; };
struct LoDF {
        Vertex *Vertices;
        Leaf *Leaves;
        GroupLeaf *GroupLeaves;
        Changed_group *Changed_groups;
        Visualised_leaf *Visualised_leaves; };
```

The main data structure is an array of clusters, called *GroupLeaves* (Figure 5). Each cluster stores all the leaves that form a binary tree of the data organisation F^r, *Leaves*, and some additional information.

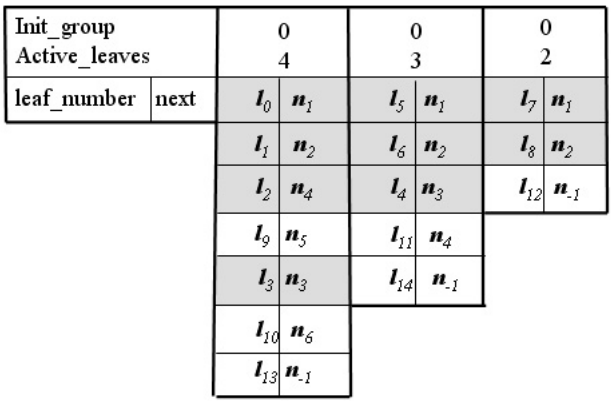

Figure 5. Example of data stored in clusters 0, 1 and 2 for the most detailed representation.

Regarding the stored leaves, each l_i is stored in the clusters following the order of simplification established by the FSA, i.e. by levels of depth in the binary tree. This storing order makes it possible to perform the leaf collapse or split operations quickly. Furthermore, leaves in each cluster are linked following the visualisation order: first of all the leaves that form the original representation are linked and then the ones obtained in the simplification process. These links also follow the order of creation determined by the FSA. This step makes it possible to extract the leaves that form the current LoD in a very short time. In Figure 5 the link of each leaf in the cluster is represented by n_j, j being the position of the leaf in the array.

In addition, some necessary data for leaves visualisation are also stored in each cluster. These data are:

- *Init_group*: the position in the array of leaves where the first leaf of the current LoD is stored.
- *Active_leaves*: number of leaves to be visualised in the current approximation.

In order to extract the sequence of leaves that form the required LoD, we must start traversing the group from the position indicated in the *Init_group*. The number of links that the algorithm must consider is the value stored in *Active_leaves*. Apart from this structure, the multiresolution model also needs to store the numbers of the clusters where every simplification operation that is processed to obtain F_{n-1} from F_0 is performed. This data structure is called *Changed_groups*. Following the example of the data shown in Figure 5, the data stored in this structure are represented in Figure 6.

The LoDF data organisation is well adapted to graphics hardware. First of all, the geometric information of the vertices is stored in the graphics card. Besides, the structure *Visualised_leaves* is also stored in the GPU. This structure stores the vertices that form the visualised leaves of each cluster in a determined level of detail. It is divided

l_{old1}	l_{old2}	l_{new}
l_0	l_1	l_9
l_2	l_9	l_{10}
l_5	l_6	l_{11}
l_7	l_8	l_{12}
l_3	l_{10}	l_{13}
l_4	l_{11}	l_{14}

gr
0
0
1
2
0
1

Figure 6. Left: Data obtained from the FSA. Right: Data stored in *Changed_groups*.

in the groups that represent the foliage organization. Each cluster sequentially contains the four indices that represent the leaves in the visualised level of detail. When a change in the LoD is required by the application, the affected clusters are updated in the CPU. The only data that will be sent to the hardware are the leaves of these updated clusters. The rest of the groups will remain in the graphics card without undergoing any kind of modification, that is to say, just as they were before the change of LoD. Visualisation time is considerably reduced due to the fact that only the information about the updated clusters is sent to the graphics card instead of sending all the geometry of the foliage.

A comparison of the visualisation process between no hardware-oriented and hardware-oriented implementations is shown in Figure 7.

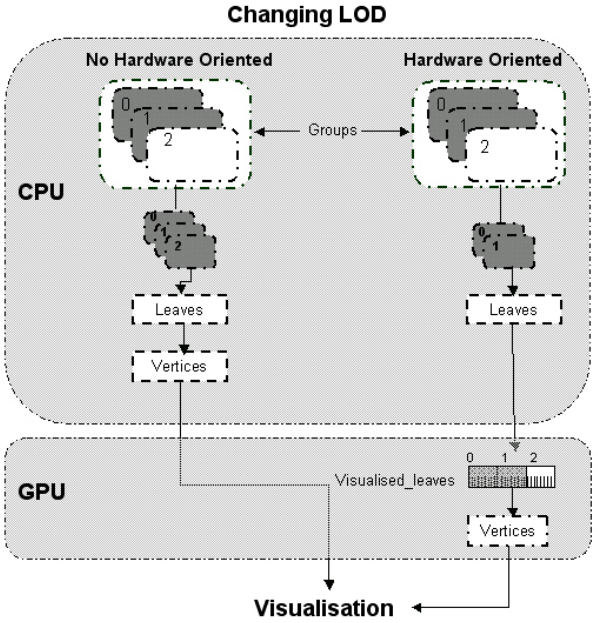

Figure 7. Example of the data transfer comparison between no Hardware-Oriented and Hardware-Oriented representation.

B. Rendering Algorithms for a Uniform LoD

The level of detail of a tree situated far from the observer does not generally require zones with more resolution than others. In this case, the level of detail has to increase in a uniform manner as it increases its importance within the scene.

In the presented multiresolution model, the defined criterion that determines the interest of the tree in the scene is its distance to the camera. This criterion has been implemented in the function *UniformLoD*. It decides the number of leaves to visualise in the foliage. In order to obtain a good management of the scene, two distances have been established:

- **Minimum distance** D_{MIN} . If the tree is closer to the viewer than this distance, the foliage is full detailed represented.
- **Maximum distance** D_{MAX}**.** The foliage of the tree is represented its coarsest approximation when it is further than this distance.

In the case of a tree situated between D_{MIN} and D_{MAX}, the number of the visualised leaves increases or diminishes in a lineal way at the same time as the distance does. The proposed pseudo-code of the function *UniformLoD* is shown in Algorithm 1.

```
Function UniformLoD (foliage,camera) : return (n_leaves)
  dist = Distance (foliage,camera);
  if (dist > D_MAX) then      // Too far
    n_leaves = N_LEAVES_MIN;
  else
    if (dist < D_MIN) then      // Very close
      n_leaves = N_LEAVES_MAX;
    else      // It is situated in the intermediate zone
      int_dist = D_MAX - D_MIN;
      int_leaves = N_LEAVES_MAX - N_LEAVES_MIN;
      n_leaves = Calculate_Leaves(dist,int_dist,int_leaves);
    end if
  end if
  end Function
```

Algorithm 1. Algorithm for the function *UniformLOD*.

Once the application where the foliage is included establishes the number of leaves for the appropriate LoD, the extraction algorithm determines the number of leaf split or collapse operations that must be performed in the foliage. In order to obtain a uniform resolution in the foliage, the leaf collapse operations have to be performed in the order established by the FSA. This sequence has been stored in the array *Changed_groups*. Then, a pointer, *index*, crosses the *Changed_groups* array and updates the *Init_group* and the *Active_leaves* fields in each cluster where an operation has to be performed.

Due to the storing order of the leaves in the clusters, a leaf-collapse is simply performed by increasing the *Init_group* by two units, and a leaf-split by decreasing this field by two units. These operations allow us to automatically update the position where the extraction algorithm starts traversing the leaves of the group. The pseudo-code of the algorithm designed in this case to

extract the data to visualize one is presented in Algorithm 2.

Init_group Active_leaves		4 2		2 2		0 2	
leaf_number	next	l_0	n_1	l_5	n_1	l_7	n_1
		l_1	n_2	l_6	n_2	l_8	n_2
		l_2	n_4	l_4	n_3	l_{12}	n_{-1}
		l_9	n_5	l_{11}	n_4		
		l_3	n_3	l_{14}	n_{-1}		
		l_{10}	n_6				
		l_{13}	n_{-1}				

Figure 8. Example of data stored after performing three leaf collapse operations.

```
n_collapse = 0;      n_split = 0;
// Obtaining the number of leaves to visualise
n_leaves = UniformLoD (foliage, camera);

// Obtaining the number of operations to perform
if ( n_leaves < n_actual_leaves)
    // New approximation requires less leaves than old one
    n_collapse = (n_actual_leaves−n_leaves);
else
    // New approximation requires more leaves than old one
    n_split = (n_leaves−n_actual_leaves);

// Performing collapse operations
while ( n_collapse > 0) do
    // Obtaining the cluster affected number
    gr = Cluster_Number(index, Changed_groups);
    // Updating the cluster
    Group[gr].Init_group = Group[gr].Init_group + 2;
    Group[gr].Active_leaves –;
    index ++;     n_collapse –;
end while

// Performing split operations
while ( n_split > 0) do
    // Obtaining the cluster affected number
    gr = Cluster_Number(index, Changed_groups);
    // Updating the cluster
    Group[gr].Init_group = Group[gr].Init_group - 2;
    Group[gr].Active_leaves ++;
    index - -;     n_split - -;
end while
```

Algorithm 2. Algorithm for uniform LOD extraction.

Figure 8 shows how three leaf collapses affect the initial data, situated on the left of the figure. According to the information stored in *Changed_groups*, two collapses are performed in the 0 and one in the 1 clusters. The leaves to be visualised are shaded. When the necessary operations have been processed, the visualisation step begins. This algorithm starts traversing the group from the position indicated in the *Init_group*. It follows the links stored in the leaves to achieve the leaf visualisation sequence. The number of links that the algorithm has to consider is stored in *Active_leaves*. The proposed pseudo-code for the visualisation process is shown in Algorithm 3.

C. Rendering Algorithms for a Variable LoD

The data organisation presented before has an important advantage: different clusters can be visualised at different resolutions. This fact makes it possible to represent the foliage with different detailed zones coexisting in the same representation. The application assigns a certain number of leaves to represent the foliage and the leaves that are visualised can be distributed following some specific criteria.

```
gr=0;
// For all the groups
while (gr <= NUMBER_GROUPS) do
   // Obtaining the data of the affected group
   first_leaf = Group[gr].Init_group;
   number_of_leaves = Group[gr].Active_leaves;
   // Starting the visualisation process
   while (number_of_leaves > 0) do
        Drawleaf(Group[gr].List[visualise_hoja].leaf_number);
      first_leaf = Group[gr].List[first_leaf].next;
      number_of_leaves - -;
   end while
   gr++;
end while
```

Algorithm 3. Algorithm for uniform LOD rendering.

A variable resolution LoD requires some specific criteria. These criteria decide which part of the object is to be simplified and which part is to be refined. They depend on the final application where the multiresolution object is to be included. In the case of LoDF, the *distance to the viewer* criterion has been implemented in order to determine the most detailed zone. Clusters situated near the camera require more detail than those farther away from it. The function *MostDetailedZone* calculates the appropriate number of leaves in the cluster depending on this criterion. This function evaluates each cluster and returns the appropriate number of leaves in each one following a linear function. Algotihm 4 offers a pseudo-code version of the extraction process.

The number of leaves to visualise allows us to know the number of leaf collapses or splits operations that have to be performed in the clusters. If the number of leaves to be visualised in the required LoD is higher than the number of leaves currently being visualised, then some leaf split operations have to be performed. In the other case, if the new LoD requires less leaves than the one currently being visualised, the algorithm has to perform the appropriate number of leaf collapses. This information is used to obtain the *Init_position* in the leaf array of the cluster. The number of links to be traversed is the number of leaves that we want to visualise, which is returned by the function *MostDetailedZone* and stored in the *Active_leaves* field of the cluster. Nevertheless, more

criteria can easily be added to LoDF if the application makes it necessary to do so. Figure 9 shows two trees where part of their foliage is less detailed than the rest.

```
For each_affected_group gr
// Obtaining the number of leaves to visualise
n_leaves = MostDetailedZone (gr, camera);

// Obtaining the number of operations to perform
if ( n_leaves[gr] < Group[gr].Active_leaves)
   // New approximation requires less leaves than old one
   n_collapse = (Group[gr].Active_leaves−n_leaves[gr]);
   // Updating the cluster
   Group[gr].Init_group = Group[gr].Init_group + 2 ∗ n_col;
else
   // New approximation requires more leaves than old one
   n_split= (n_leaves+Group[gr].Active_leaves);
   // Updating the cluster
   Group[gr].Init_group = Group[gr].Init_group - 2 ∗ n_split;
end if

// Updating the number of leaves in the cluster
Group[gr].Active_leaves = n_leaves[[gr];
end for
```

Algorithm 4. Algorithm for variable LOD extraction.

Figure 9. Example of variable resolution according to the implemented criterion. Some parts of the tree is more detailed than others.

V. Forest visualisation

It is usual to find in natural environments forests where a specific species of tree predominates. Using *instantiation* as a technique to visualise forests in real-time takes advantage of this fact. Then, each tree in the forest is represented as an instance of one species.

Each instance stores in the GPU a vector *Visualised_leaves* with the leaves that represent the current approximation. Besides, the only information that has to be stored for each cluster are the fields *Init_group* and *Active_leaves*, as well as a pointer that indicates the number of collapse operations performed in the data structure *Changed_groups*. This information is necessary to efficiently extract the geometry that visualises each instance. Figure 10 graphically shows the data organization to represent three instances of the same foliage.

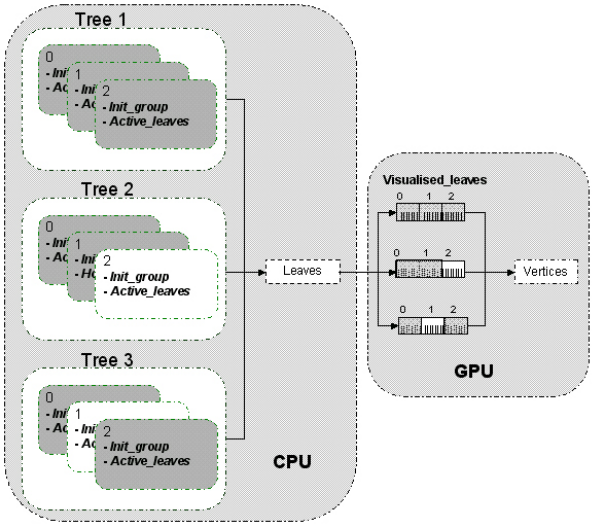

Figure 10. Data organization for visualising three instances of the same foliage.

Figure 12 shows a fully detailed forest. It is formed by 100 instances of 8 different species. The scene is represented using more than 15 millions of polygons. Figure 13 represents the same forest reducing the number of polygons in 40%. The visual appearance is similar in both Figures. Finally, Figure 14 makes a comparison between Figure 12 and Figure 13. The pixels that are different in both figures are marked, showing how the quality obtained with our solution is very high.

VI. Results

All the tests presented in this section were implemented in C++ with OpenGL as the graphics library and were conducted on a computer with an NVIDIA GeForce 6800 Series GPU.

Figure 11. Tree *Taxus Baccata* modelled using Xfrog. The trunk is made up of 34.202 polygons and the foliage consist of 96.320 (48.160 leaves).

The tests we carried out consist in traversing different LoDs of the tree shown in Figure 11. Throughout the tests, the LoD varies between 0 and 1, 0 being the most detailed approximation and 1 the least. The tests were designed following the criterion defined in the multiresolution

models. Two graphs are offered in every figure to show the results:

- **Total time.** Time that the model spends on extracting and also visualising each level of detail.
- **Extraction time.** Time that the model uses to extract the geometry required to change from one approximation to another one.

The method is compared with the one presented by Remolar et al. [11]. This method combined VDF for the foliage and MOM [25] for the trunks. In order to achieve a better comparison, the tests were performed on a computer with the same characteristics as the one used in that work. As the multiresolution methods *VDF+MOM* are not hardware-oriented, first of all they were compared with the methods presented in this paper, *LoDF+LodStrip*, but without taking advantage of the graphics hardware. In other words, all the geometric data about the *LoDF+LodStrip* models are stored in the main memory in the same conditions as the data in *VDF+MOM*. Two tests were designed in this case: one for uniform changes in the LoD and other one for variable changes. Results are shown in Figure 15. In the figures it can be seen that our method considerably reduces the total time. Using *LoDF+LodStrip*, trees can be visualised quite a lot faster than when using *VDF+MOM*.

Finally, as the main advantage of representing a tree using our method is that multiresolution models are graphics hardware oriented, other tests have been carried out. In this case, Figure 16 shows how the total and extraction time for visualising a tree can be reduced by using this property.

Moreover, the storage cost has also been considered. Table I summarizes different characteristic of the trees used in the experiments: the number of vertices, polygons and leaves of the original model, and their original storage cost (in megabytes). Tables II and III show the characteristics and storage costs of the *VDF* and *LoDF (no Hardware-Oriented)* representations, respectively. In these tables, it can be seen the number of the stored approximations, n, the number of leaves (the number of vertices is the same as the original representation) and their storage costs. In the case of the *LoDF (no Hardware-Oriented)* representation, it is indicated the number of clusters that form the foliage. A *VDF* representation is 1.62 times the original model, while a *LoDF (no Hardware-Oriented)* representation is only 1.24 times, which is a reduction of nearly a 25%.

Tables IV and V respectively show the characteristics and storage costs of the *LoDF (no Hardware-Oriented)* and *LoDF (Hardware-Oriented)* representations. It can be appreciated how the *LoDF (Hardware-Oriented)* representation is 1.49 times the original model and a *LoDF (no Hardware-Oriented)* representation is 1.24 times. This increase is justified due to the reduction of visualization time of the approximation.

VII. Conclusions and Future Work

This article presents a new multiresolution approach for foliage that exploits the characteristics of current graphics hardware. The model *Level of Detail Foliage* LoDF, combined with *LodStrip*, allow us to change the level of detail of a tree representation in a continuous way. The main advantage is that they reduce the traffic of data through the AGP/PCIe bus by diminishing the amount of information that is sent when a change in level of detail is produced. With respect to LoDF, this is obtained by grouping leaves in independent clusters and only modifying a small set of data. Furthermore, LoDF is also capable of offering variable resolutions of the crown of the trees.

These two methods are combined in the multiresolution scheme that is presented in order to make it feasible to represent detailed scenes of forest in interactive applications. It allows us to render every detail of the tree even when the viewer is extremely close to it. Because it is adapted to the graphics hardware, it produces better results than current models based on images or points.

One line of research we are currently working on is to obtain advanced illumination effects and animation of the foliage. One of our aims in these studies is to continue to take advantage of graphics hardware programming.

Acknowledgments

This work was supported by the Spanish Ministry of Science and Technology (TIN2004-07451-C03-03 and FIT-350101-2004-15), the European Union (IST-2-004363) and FEDER funds.

References

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

[2] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis, "Interactive visualization of complex plant ecosystems," in *VIS '02: Proceedings of the 13rd Conference on Visualization*. IEEE Computer Society, 2002, pp. 219–226.

[3] J. F. Ramos and M. Chover, "Lodstrips: Level of detail strips," in *International Conference on Computational Science*, 2004, pp. 107–114.

[4] C. Rebollo, I. Remolar, M. Chover, and O. Ripolls, "An efficient continuous level of detail model for foliage," *WSCG '06: Proceedings of 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 335–342, 2006.

[5] I. Remolar, C. Rebollo, M. Chover, and J. Ribelles, "Real time tree rendering," in *Lecture Notes in Computational Science 3039*, 2004, pp. 173–180.

[6] P. Maciel and P. Shirley, "Visual navigation of large environments using textured clusters," in *SI3D '95: Proceedings of the Symposium on Interactive 3D graphics*, 1995.

[7] N. Max, "Hierarchical rendering of trees from precomputed multi-layer z-buffers," in *Rendering Techniques '96: Proceedings of the 7th Eurographics Workshop*, X. Pueyo and P. Schrder, Eds. Springer-Verlag, 1996, pp. 165–174.

[8] P. Decaudin and F. Neyret, "Rendering forest scenes in real-time," in *Rendering Techniques '04: Proceedings of the 15th Eurographics Workshop*, 2004, pp. 93–102.

[9] A. Reche, I. Martin, and G. Drettakis, "Volumetric reconstruction and interactive rendering of trees from photographs," *ACM Transactions on Graphics, SIGGRAPH Conference Proceedings*, vol. 23, no. 3, pp. 720–727, 2004.

[10] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder, "Hierarchical image caching for accelerated walkthroughs of complex environments," in *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, 1996, pp. 75–82.

[11] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte, "View-dependent multiresolution model for foliage," *WSCG '03: Proceedings of 11st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, vol. 11, no. 2, pp. 370–378, 2003.

[12] I. Garcia, M. Sbert, and L. Szirmay-Kalos, "Leaf cluster impostors for tree rendering with parallax," in *Rendering Techniques '05: Proceedings of the 16th Eurographics, Short Presentations*, 2005.

[13] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen, "Realistic real-time rendering of landscapes using billboard clouds," *Computer Graphics Forum*, vol. 24, no. 3, pp. 507–516, 2005.

[14] A. Fuhrmann, E. Umlauf, and S. Mantler, "Extreme model simplification for forest rendering," in *Natural Phenomena '05. Eurographics Workshop on Natural Phenomena*, 2005, pp. 57–66.

[15] J. Lacewell, D. Edwards, P. Shirley, and W. Thompson, "Stochastic billboard clouds for interactive foliage rendering," *Journal of Graphics Tools*, vol. 11, no. 1, pp. 1–12, 2006.

[16] W. Reeves and R. Blau, "Approximate and probabilistic algorithms for shading and rendering structured particle systems," in *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, 1985, pp. 313–322.

[17] J. Weber and J. Penn, "Creation and rendering of realistic trees," in *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, R. Cook, Ed. ACM Press, 1995, pp. 119–128.

[18] M. Stamminger and G. Drettakis, "Interactive sampling and rendering for complex and procedural geometry," in *Rendering Techniques '01: Proceedings of the 12nd Eurographics Workshop*, S.Gortler and C.Myszkowski, Eds. Springer-Verlag, 2001, pp. 151–162.

[19] G. Gilet, A. Meyer, and F. Neyret, "Point-based rendering of trees," in *Natural Phenomena '05. Eurographics Workshop on Natural Phenomena*, P. P. E. Galin, Ed., 2005, pp. 67–72.

[20] A. Meyer, F. Neyret, and P. Poulin, "Interactive rendering of trees with shading and shadows," in *Rendering Techniques '01: Proceedings of the 12nd Eurographics Workshop*. Springer-Verlag, 2001, p. 88.

[21] J. Lluch, E. Camahort, and R. Vivo, "An image based multiresolution model for interactive foliage rendering," *WSCG '04: Proc. of 12nd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, vol. 12, no. 3, pp. 507–514, 2004.

[22] C. Rebollo, I. Remolar, M. Chover, and J. Gumbau, "Hardware-oriented visualisation of trees," in *ISCIS '06: Lecture Notes in Computational Science 4263, Proceedings of the 21st International Symposium on Computer and Information Sciences*, 2006, pp. 374–383.

[23] M. Garland and P. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, 1997, pp. 209–216.

[24] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo, "Geometric simplification of foliage," in *Proceedings of Eurographics 2002, Short Presentations*, I. Navazo and P. Slusallek, Eds. Eurographics, 2002.

[25] J. Ribelles, A. Lpez, O. Belmonte, I. Remolar, and M. Chover, "Variable resolution level-of-detail of multiresolution ordered meshes," *Proc. of 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2001)*, vol. 2, pp. 299–306, 2001.

Cristina Rebollo received her MS degree in Computer Science from the Universidad Católica de Deusto, Vizcaya, Spain, in 1988, and her PhD in Computer Science from the Universitat Jaume I, Castellón, Spain, in 2006.

Since 2004, she has been an Assistant Professor of Computer Science at the department of Computer Languages and Systems at the Universitat Jaume I, Spain. Her research interests include multiresolution modelling, hardware-graphics programming and real time visualisation of outdoor scenes.

Inmaculada Remolar received her MS degree in Computer Science from the Universidad Politécnica de Valencia, Valencia, Spain, in 1995, and her PhD in Computer Science from the Universitat Jaume I, Castellón, Spain, in 2005.

She has been an Assistant Professor of Computer Science at the department of Computer Languages and Systems at the Universitat Jaume I, Spain, since 1998. Her research interests include multiresolution modelling, real time visualisation of trees oriented to computer games.

Dr. Remolar is a member of Eurographics.

Miguel Chover received his MS degree in Computer Science in 1992, and his PhD in Computer Science in 1996, from the Universidad Politécnica de Valencia, Valencia, Spain.

Since 1992, he has been an Assistant Professor of Computer Science at the department of Computer Languages and Systems at the Universitat Jaume I, Spain. His research areas include multiresolution modelling, real time visualisation and collaborative virtual worlds.

Dr. Chover is a member of Eurographics.

Jesús Gumbau received his MS degree in Computer Science in 2004 from the Universitat Jaume I, Castellón, Spain and is currently pursuing his PhD in Computer Science from the same University.

He is currently a Software Engineer working in an European Project developed in the Universitat Jaume I.

His research areas include multiresolution modelling, real time visualisation and collaborative virtual worlds.

Oscar Ripollés is a full time PhD student at the Universitat Jaume I in Castellón, Spain, in the Computer Graphics group. He received his MS degree in Computer Science in 2004 at the same University.

His research interests include multiresolution modelling, geometry optimization, hardware programming and virtual communities.

Figure 12. Forest visualised full detailed.

Figure 13. Forest visualised reducing the detail in 40%.

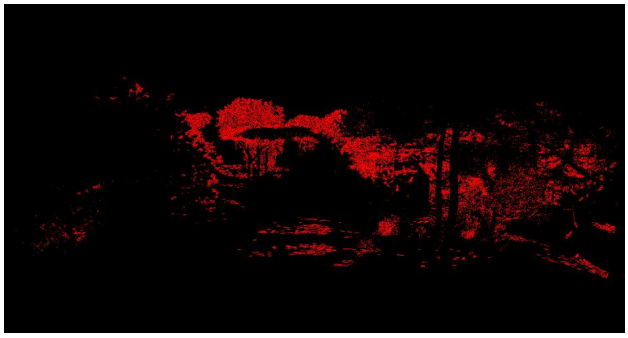

Figure 14. Visual comparison of the scenes.

	Original			
	Vertices	Polygons	Leaves	MB.
Betula Populifolia	32,560	16,280	8,140	0.50
Betula Lenta	81,504	40,752	20,376	1.24
Pistacia Lentiscus	22,376	11,188	5,594	0.34
Spartium Junceum	42,960	21,480	10,740	0.66
Olea Europaea	64,428	32,214	16,107	0.98
C.Sempervirens	79,804	39,902	19,951	1.22
Quercus Suber	81,124	40,562	20,281	1.24
Sorbus Aucuparia	99,360	49,680	24,840	1.52
J. Oxycedrus	129,156	64,578	32,289	1.97
Quercus Cerris	164,620	82,310	41,155	2.51
Taxus Baccata	192,640	96,320	48,160	2.94

TABLE I.
TREES WITH THEIR CHARACTERISTICS AND ORIGINAL STORAGE COST.

	VDF			Ratio
	n	Leaves	MB.	
Betula Populifolia	8,019	16,158	0.80	1.62
Betula Lenta	20,090	40,465	2.01	1.62
Sorbus Aucuparia	24,498	49,337	2.45	1.62
Taxus Baccata	47,520	95,679	4.76	1.62

TABLE II.
TREES WITH THEIR CHARACTERISTICS AND VDF STORAGE COST.

	LoDF (no Hardware-Oriented)				Ratio
	n	Leaves	Groups	MB.	
Betula Populifolia	8,019	16,158	119	0.61	1.24
Betula Lenta	20,090	40,465	285	1.54	1.24
Sorbus Aucuparia	24,498	49,337	339	1.87	1.24
Taxus Baccata	47,520	95,679	641	3.63	1.24

TABLE III.
TREES WITH THEIR CHARACTERISTICS AND LODF (NO HARDWARE-ORIENTED) STORAGE COST.

	LoDF (no Hardware-Oriented)				Ratio
	n	Leaves	Groups	MB.	
Pistacia Lentiscus	5,504	11,098	89	0.42	1.24
Spartium Junceum	10,585	21,325	155	0.81	1.24
Olea Europaea	15,880	31,987	288	1.21	1.24
C.Sempervirens	19,675	39,626	277	1.50	1.24
Quercus Suber	19,999	40,280	282	1.53	1.24
Sorbus Aucuparia	24,503	49,343	227	1.50	1.24
J. Oxycedrus	31,880	64,169	440	2.46	1.24
Quercus Cerris	40,598	81,753	558	3.10	1.24
Taxus Baccata	47,520	95,680	641	3.63	1.24

TABLE IV.
TREES WITH THEIR CHARACTERISTICS AND LODF (NO HARDWARE-ORIENTED) STORAGE COST.

	LoDF (Hardware-Oriented)				Ratio
	n	Leaves	Groups	MB.	
Pistacia Lentiscus	5,504	11,098	89	0.51	1.49
Spartium Junceum	10,585	21,325	155	0.97	1.49
Olea Europaea	15,880	31,987	288	1.46	1.49
C.Sempervirens	19,675	39,626	277	1.81	1.49
Quercus Suber	19,999	40,280	282	1.84	1.49
Sorbus Aucuparia	24,503	49,343	2'27	1.50	1.49
J. Oxycedrus	31,880	64,169	440	2.93	1.49
Quercus Cerris	40,598	81,753	558	3.73	1.49
Taxus Baccata	47,520	95,680	641	4.37	1.49

TABLE V.
TREES WITH THEIR CHARACTERISTICS AND LODF (HARDWARE-ORIENTED) STORAGE COST.

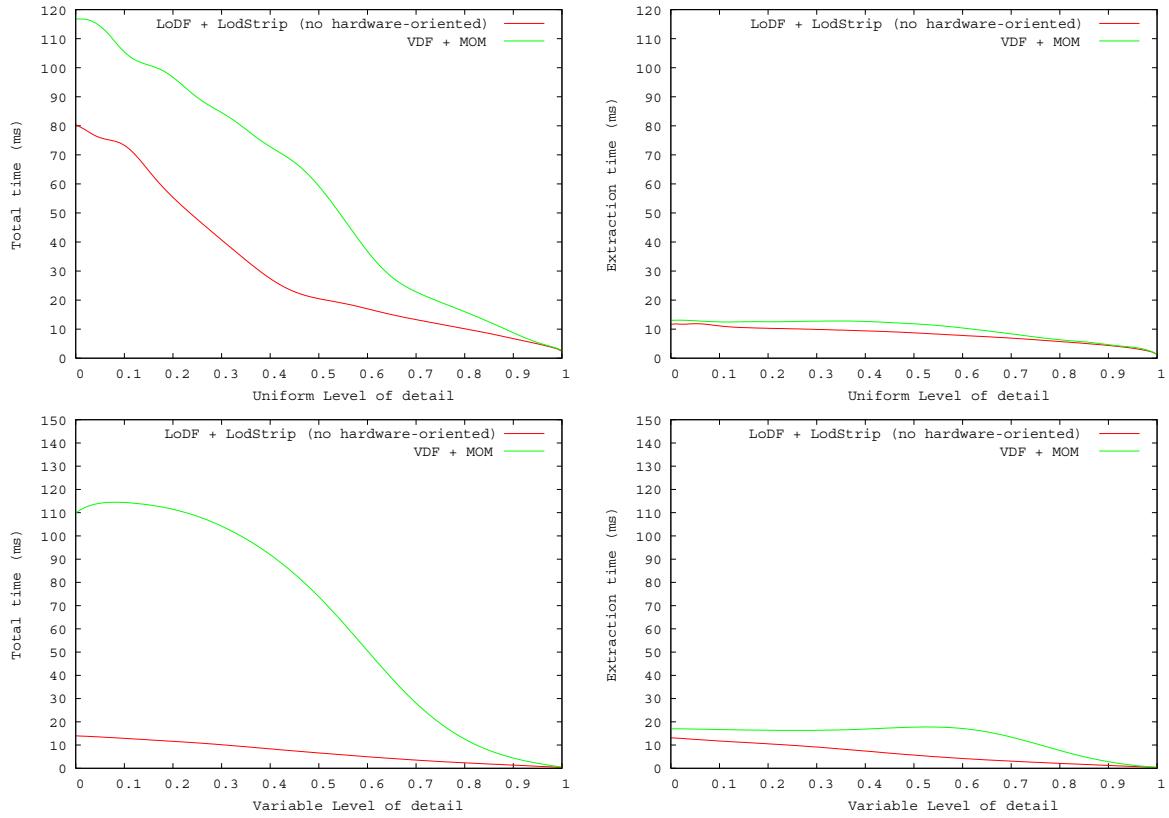

Figure 15. Tree *Taxus Baccata*. Results obtained when comparing *VDF+MOM* with *LoDF+Lodstrip*.

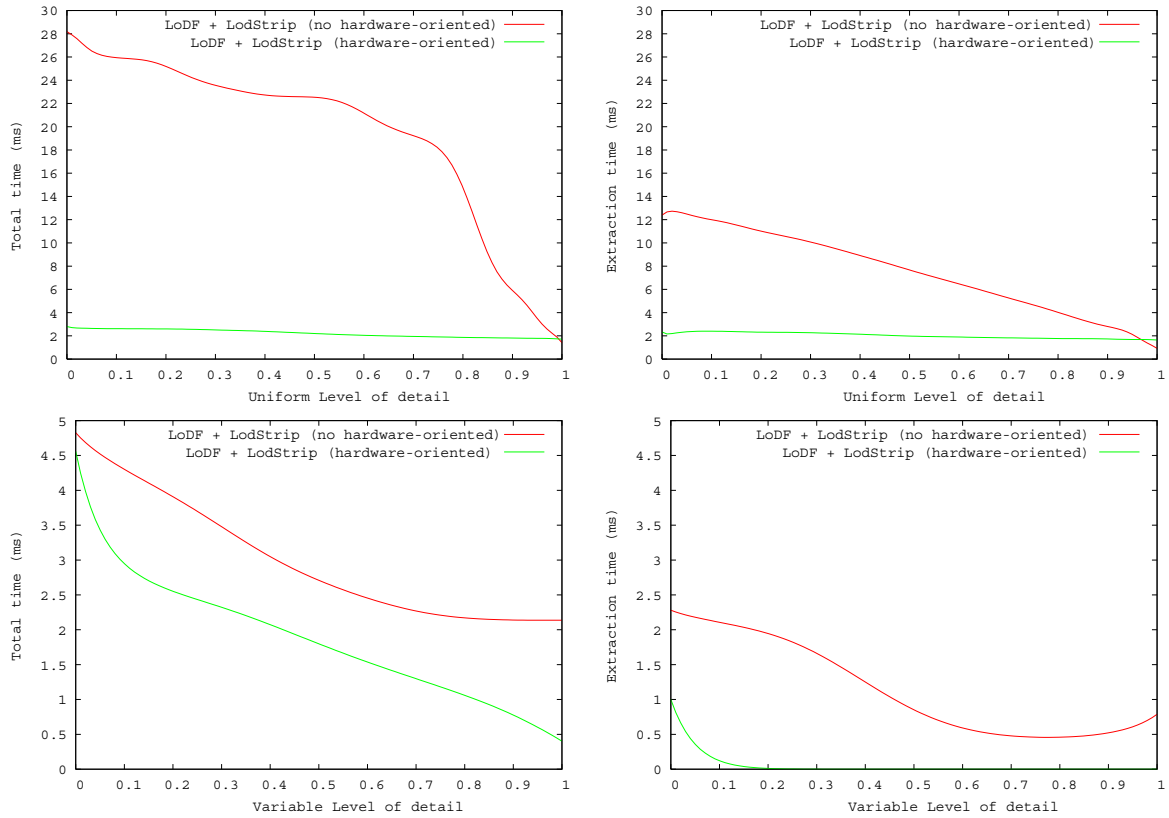

Figure 16. Tree *Taxus Baccata*. Results obtained when comparing *LoDF+LodStrip* with and without taking advantage of the graphics hardware.