Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2012) 485–494
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

A Hybrid Mutiresolution Representation for Fast Tree Modeling and Rendering

Javier Lluch[a], Emilio Camahort[a], Jose Luis Hidalgo[a], Roberto Vivo[a]

[a] *University Institute of Control Systems and Industrial Computing, Universidad Politecnica de Valencia, Spain*

Abstract

We introduce a new representation for modeling and rendering plants and trees. The representation is multiresolution and can be automatically generated and rendered in an efficient way. For the trunk and branches we procedurally build a multiresolution structure that preserves the visual appearance of a tree, when rendered at different levels of detail. For the leaves we build a hierarchy of images by pre-processing the botanical tree structure. Unlike other representations, the visual quality of our representation does not depend on viewing position and direction. Our models can be applied to any computer graphics area that requires modeling outdoor scenes like interactive walkthroughs and fly-by's, realistic rendering, simulation and computer games.

Keywords: L-Systems, Procedural Multiresolution, Image-Based Modeling and Rendering, Levels of Detail, Plant and Tree Modeling, 3D Rendering, Computer Graphics

1. introduction

Plants and trees are an integral part of outdoor scene modeling in real-time graphics applications like virtual reality, simulation, navigational aids and computer games. Their representation is commonly made of procedural models based on L-systems. Most graphics applications convert those models to polygon-based representations for hardware-accelerated rendering on graphics engines.

Accurate modeling of trees requires large numbers of polygons. A typical outdoor scene contains many trees, each of them made of hundreds of thousands of polygons. Even on modern graphics hardware, the number of polygons greatly exceeds the per-second polygon rate of the graphics engine, we aim to reduce the amount of geometry rendered.

Multiresolution modeling reduces the polygon count by using geometry-based simplification methods, but most of these methods fail to capture the nature of plants and trees. Image-based tree models replace geometry by images called impostors, but they require a lot of storage and introduce artifacts at close views. However, introducing geometry and image-based simplification methods at the procedural level leads to more natural simplification methods, what we call *procedural multiresolution*.

We introduce a new multiresolution representation for plant and tree modeling and rendering. Our representation is a hybrid procedural and image-based representation. Trunk and branches of a tree are modeled using procedural

Email addresses: jlluch@dsic.upv.es (Javier Lluch), camahort@dsic.upv.es (Emilio Camahort), jhidalgo@ai2.upv.es (Jose Luis Hidalgo), rvivo@dsic.upv.es (Roberto Vivo)

1877-0509
doi:10.1016/j.procs.2010.04.052

multiresolution which preserves the visual appearance of the tree. The leaves of a tree are modeled using a hierarchical image-based technique. The representation requires a small amount of storage and provides a realistic sense of depth and is able to generate different levels of detail (LOD) on demand.

We can render scenes with more than a thousand trees at 20 frames per second. Such a scene only takes 250Kbytes of storage in compressed form and the generation of geometry and images takes a few seconds at load time. Trees can be as leafy as desired because the increase in rendering time is negligible. Our method is also suited for transmitting a scene from a server to a client with very little bandwidth use.

This article is organized as follows. Section 2 surveys interactive tree rendering and multiresolution modeling. Section 3 explains how to model trees using L-systems. The following section describes in detail our representation and the algorithms required to build it. Section 5 presents some results obtained with our representation.

2. Background

Our work is primarily related to two areas of computer graphics: plant and tree modeling and multiresolution representations.

2.1. Plant and Tree Modeling

The simplest tree models are made of a few impostors, texture-mapped polygons that represent the tree. One approach uses two orthogonal rectangular impostors forming a cross. A different approach, called billboard, uses a single rectangle that always faces the viewer. Both techniques are inadequate for rendering trees from above and at close range.

A different representation used in computer games uses both geometry and impostors, modelling the trunk with tubes and sets of branches with billboards. SpeedTree [1] is a commercial application that uses this approach, it supports rendering of landscape and tree wind movement. However, relatively expert skill is necessary to build trees from scratch and it uses discrete LODs, lacking realism in short distances.

The most commonly used modeling technique for plants and trees is parametric L-systems. Lindenmayer first introduced L-systems for modeling cellular division [2]. Later, Prusinkiewicz et al. applied L-systems to the representation of plants and trees [3]. Parametric L-systems were reported in the works by Prusinkiewicz and Lindenmayer [4] and by Hanan [5].

A different modeling approach is modeling by components introduced by Lintermann and Deussen [8]. Models have also been built using images of real plants [9] and real trees [10] [11].

2.2. Level-of-Detail Techniques for Plants and Trees

New techniques have been proposed in the literature that introduce LOD modeling and advanced image-based representations. LOD modeling techniques include degradation at range and pixel-based LODs [15], BSP-tree space partitioning and multiresolution [16], cluster-based hierarchical polygon decimation and compression [17], and LOD foliage with continuous multiresolution and hardware-assisted rendering [18]. Image-based representations include layered depth images [19] [20], volumetric textures [21] and bidirectional textures [22].

2.3. Multiresolution Modeling

Multiresolution is a modeling technique that was first introduced to accelerate rendering of complex geometries [23] [24].

A multiresolution model represents an object using different levels of detail (LODs). The finest LOD represents the full-resolution model. Coarser LODs represent lower resolution versions of the model suitable for faster rendering. For example, when using a geometric representation, coarser LODs typically contain smaller numbers of vertices, edges and polygons.

A good multiresolution model is characterized by the following four properties: (i) The size of the model must not increase with the number of LODs.(ii) Extraction of the LODs must be fast enough to support interactive rendering. (iii) There must be no loss of information: the finest LOD must reproduce the original model at its full resolution.(iv) The changes between LODs must be smooth.

Geometry-based simplification methods have been successfully applied to many areas of computer graphics. However, they fail to maintain the general structure of a tree: (i) they may join vertices belonging to independent branches, (ii) they may insert geometry in otherwise empty space, and (iii) they eliminate smaller terminal branches first, resulting in smaller trees as the simplification proceeds. During the process, the tree's volume diminishes and it loses its visual appearance.

2.4. Discussion

Current tree representations typically need a large amount of storage to produce good quality visual results. Additionally, they require an expert designer to build them and several hours to generate a scene suitable for rendering. We propose an alternative representation that can be automatically built given a simple L-system. Our tree generation algorithm runs in a few seconds and produces tree models whose rendering quality does not depend on viewing position or direction. We divide the problem of modeling a tree into two sub-problems: modeling the trunk and branches and modeling the leaves.

For the trunk and branches we introduce a new multiresolution representation based on parametric L-systems. For the leaves we build an image-based representation that is both hierarchical and multiresolution. Once both representations are generated, we combine them into a hybrid model suitable for rendering at interactive rates.

3. Modeling Plants and Trees using L-Systems

Our hybrid representation is based on parametric L-systems. Parametric L-systems are the most commonly used method for modeling plants and trees. In this section we describe the process of derive and interpret an L-System.

3.1. L-Systems: Definition and Derivation

A parametric L-system is given by an *axiom* and a set of *derivation rules*, also called *productions*. The axiom is made of a *chain* of bracketed *modules*. Each module contains a *symbol* and a list of *parameters*. Productions can be guarded by Boolean conditions on the parameters of a module. When a production is applied to the axiom, a module is replaced by a new set of bracketed modules, thus resulting in a new *derived chain*. A set of bracketed modules represents a branch and its descendants. A choice of axiom and derivation rules determines the features of a given species.

For example, the following L-system generates a tree whose branches have each two children, and each child is half the length of its parent. All other branch-related information (thickness, position, orientation, etc.) has been omitted for simplicity.

$$\begin{aligned} Axiom &: A(length) \\ Rule1 &: A(l) : itNum < maxIt \rightarrow B(l)[A(l/2)A(l/2)] \\ Rule2 &: A(l) : itNum = maxIt \rightarrow B(l) \end{aligned} \tag{1}$$

Here *itNum* is the number of derivations that have been applied so far, and *maxIt* is the total number of derivations to be applied. Note that B is a terminal symbol, since modules containing B cannot be further derived.

The characteristics of a specific tree are given by the initial value of *length* and by the application of the derivations. After a certain number of derivations, the derived chain only contains terminal symbols. That chain represents a tree specimen. We call it output chain. For instance, the following *output chain* represents a specimen of the above species:

$$B(1)[B(.5)[B(.25)B(.25)]B(.5)] \tag{2}$$

We have briefly introduced L-systems. L-systems may also contain logical expressions, stochastic productions and context-sensitive derivation rules. The reader is referred to [4] and [5] for a detailed description of these extensions.

3.2. Chain Interpretation

We obtain the geometric representation of a tree by interpreting its output chain. The process is based on the turtle metaphor, commonly used in graphics applications based on the LOGO programming language [29].

Turtle's state is composed by *current position*, and *current orientation*. The state can be saved and and recovered from a stack. Position and orientation can be changed by one of two types of modules:*advance* and *rotate*. Advance modules also has a graphical interpretation, typically a cone or cylinder representing a branch.

The turtle stack allows storing the state of the turtle. It is used to traverse the branching structure of the tree, as given by the bracketed structure of the output chain. If the interpreter finds an opening bracket "[", it pushes the turtle state onto the stack. After parsing the next module, the interpreter generates the module's geometry as a child branch stemming from the current position. When the closing bracket "]" is reached, the turtle pops its state from the stack and continues from the same position where the branch was started, possibly generating a sibling branch for the next module.

4. A Hybrid Tree Representation

Given a parametric L-system designed with our authoring tool, we generate a hybrid tree representation that uses procedural multiresolution for the trunk and branches and image-based modeling for the leaves.

4.1. Trunk and Branch Modeling: Procedural Multiresolution

We want to model the trunk and branches using a multiresolution representation. Our goal is a procedural-based multiresolution representation that maintains the structure of the tree. The first question is: given the output chain of an L-system, how do we choose a simplification method?

The obvious choice is to undo the derivations in the opposite order that they were applied. However, the output chain is generated following growth rules and is not appropriate for extraction of visual LODs. The results produce images of the tree at successive growth stages.

An alternative is to construct the LODs using some metric on its branches that preserves the structure of the tree. For example, consider the following output chain:

We can construct the LODs using some metric on its branches that preserves the structure of the tree. For example, consider the following output chain:

$$C1[C2[C3[C4][C5[C6][C7]]][C9[C11][C10]][C8]][C12[C13][C14][C15[C16]]] \tag{3}$$

where Cn represents the chain of modules that generates branch n (see Figure 1(a) for a possible interpretation). Associated to each branch Cn we have a set of features, like length, texture or distance to the ground. We select the features that are quantifiable and use them to determine which branches of the tree best represent its visual structure. Once we have that information, we simplify the model by eliminating the least relevant branches.

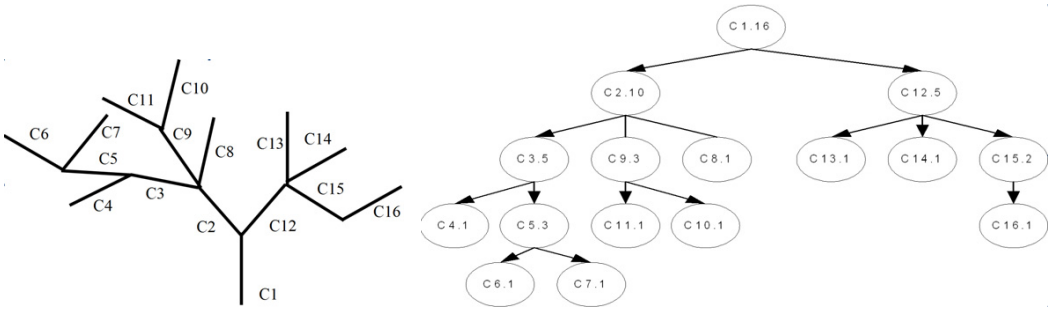

Figure 1: (a)A possible interpretation of chain (3). (b)T-ADT obtained from chain (3) assuming that all the branches are of unit length.

In practice, our LOD generation method works incrementally, adding a new set of branches in each iteration. First, we parse the output chain and build an intermediate data structure we call *tree abstract data type* or *t-ADT*. The t-ADT

```
Create the root of the t-ADT and make it the current node
for all module in the input chain do
  if module = [ then
    add a child to the current node and make it the current
    node
  else if module = ] then
    add the branching length of the current node to its parent's
    set the current node to the current node's parent
  else
    add the module's length to the current node's branching
    length
    add the module to the current node's chain
  end if
end for
```

(a) t-ADT

```
for all module of the chain do
  if module = PUSH then
    create a new box
    give initial values to min, max
    mark the box as last and open
    increment level
  else if module = POP then
    close last box
    save box
  else if module = GEOMETRIC SHAPE then
    update max, min of all open boxes
  else if module = FORWARD then
    modify turtle position
  end if
end for
```

(b) Bounding Box

Figure 2: Algorithm to compute the t-ADT (a) and the algorithm to compute the bounding box hierarchy(b).

represents the tree and contains the required metric information. With the metric, we establish an order relation on the set of branches. Traversal of the t-ADT in that order generates the procedural LODs. They are the re-ordered modules of the output chain with an enhanced bracketed structure. We call this chain the *multiresolution chain* or *multi-chain*.

At rendering time, we extract and interpret those modules of the multi-chain needed to obtain the desired visual quality. We can do it efficiently, because the modules are sorted from most relevant to least relevant. Progressive transmission and rendering are also possible with this representation. Given a limited time or bandwidth budget, we only transmit, interpret and render a limited-length prefix of the multi-chain.

4.1.1. Selecting a Metric

Consider an input chain S. We construct a t-ADT T containing a node for each branch resulting from the interpretation of S. We assign a weight to each node by applying a metric to each branch. We want a metric that associates larger values to those branches that better reflect the visual structure of the tree. There are a number of possible metrics, but we are only interested in metrics that induce an order relation on the nodes of the t-ADT. We consider four of metrics: (a) number of children, (b) number of descendants, (c) longest path to a leaf node, and (d) branching length. All these metrics quantify the relevance of the branches of a tree.

Given a node $n \in T$, the metrics are formally expressed as:

a) $B(n) = \text{car}\,\{n\}$

b) $R(n) = \text{car}\,d(n)$

c) $P(n) = l(n) + \max_{m\in\{n\}}(P(m))$

d) $L(n) = l(n) + \sum_{m\in\{n\}} L(m) = l(n) + \sum_{m\in\{n\}} l(m)$

where $\{n\}$ is the set of children of n, $d(n)$ is the set of descendants of n and $l(n)$ is the length of the branch that n represents. This length is calculated adding the distances associated to the advance symbols in the chain contained in the node. All these metrics quantify the relevance of the branches of a tree.

We prefer the branching length because it reflects the density of the descendants of a branch. The idea is to give higher precedence to branches with larger ramification. The branching-length satisfies this property and produces the best visual results.

4.1.2. Building the t-ADT

The t-ADT contains a node for each branch of the tree. Each node contains the chain of modules that generate the branch and its branching length as given by the above metric. The Figure 2(a) shows the algorithm that parses the input chain and builds the t-ADT.

Figure 1(b) shows the result of applying the algorithm to output chain (3). Each node contains its associated chain followed by a period and its branching length. For simplicity, we assume that all the branches are of unit length.

4.1.3. Generating the Multiresolution Chain

Given the t-ADT, we traverse it to generate the multi-chain. The multi-chain contains the modules of the output chain plus two new types of modules, *SAVE(id)* and *RESTORE(id)*. They define the LODs and control their extraction. *SAVE(id)* modules instruct the turtle to store its state with a unique identifier id. *RESTORE(id)* modules recover the state of the turtle stored with identifier *id*. We generate the multi-chain's LODs by extracting paths from the t-ADT. Each new path is chosen according to the branching length of its nodes.

When an LOD is extracted, we update the branching length of the nodes in the path. Let $X(n, m)$ be the extracted path, and let the path-length $lX(n, m)$ be the sum of every length in the path $X(n, m)$ (between node n and its descendant m). If $k.L$ is the branching length of node k. We update the branching lengths as follows:

$$\begin{aligned} \forall k \in X(root, m) - X(n, m) \qquad & k.L \leftarrow k.L - lX(n, m) \\ \forall k \in X(n, m) \qquad & k.L \leftarrow k.L - lX(k, m) \end{aligned} \tag{4}$$

For the nodes up to the parent of the first node n, we reduce the branching length by the path length $lX(n, m)$. For all the other nodes, we reduce the branching length by the length of the path from that node to the leaf node m.

We create the first LOD by traversing the tree from the root to one of its leaves. At each step, we choose the node with the largest branching length. When we visit a new node, we append its chain to the multi-chain and mark it as output. If the node is an intermediate node, we also append a *SAVE* module with the same id as the node's. Later, we use the saved state to generate a new finer LOD starting with one of its children. Once the LOD has been output, we update the branching lengths of all the nodes visited.

To generate the next LOD we start at the root and search for the first non-output node with the largest branching length. Once we find it, we append to the multi-chain a *RESTORE* module with its parent's id. Later, we use that module to restore the turtle's state, so that we can generate a new set of branches stemming from the parent branch. The LOD is generated by traversing the t-ADT from the node to a leaf in the same way that we generated the first LOD. Afterwards, we update the branching lengths of all the nodes between the root and the leaf. We continue this process until all the leaves of the t-ADT have been output.

For example, given the t-ADT of Figure 1(b), the algorithm generates the following multi-chain:

$$\begin{aligned} & C1 \text{ SAVE(1) } C2 \text{ SAVE(2) } C3 \text{ SAVE(3) } C5 \text{ SAVE(5) } C6 \\ & \text{RESTORE(2) } C9 \text{ SAVE(9) } C11 \text{ RESTORE(1) } C12 \\ & \text{SAVE(12) } C15\ C16 \text{ RESTORE(3) } C4 \text{ RESTORE(5) } C7 \\ & \text{RESTORE(9) } C10 \text{ RESTORE(12) } C13 \text{ RESTORE(2) } C8 \\ & \text{RESTORE(12) } C14 \end{aligned} \tag{5}$$

Note that it contains roughly the same number of modules as the input chain. However, the modules of the multi-chain are sorted depending on how relevant their branches are to the visual structure of the tree. Each LOD, except the coarsest one, starts with a *RESTORE* module and ends with a terminal module. The coarsest LOD is given by the prefix of the chain up to the first *RESTORE* module. Finer LODs follow. We can extract the entire tree by traversing the multi-chain up to the last module.

4.2. Hierarchical Textures for Foliage Rendering

We present now a hierarchical data structure to store the geometry and images of the leaves. The data structure can be easily generated from an L-system, and it can be used to represent any branched structure. It is organized as a directed acyclic graph where each node contains a geometric primitive, like a cylinder, a sphere or a polygon.

Associated to each node we also store a bounding box that encloses all the geometry of that node and its descendants. The orientation of the bounding box is the same as the turtle's orientation for that node. To build our representation for the leaves we render images of the leaves contained in each bounding box. Then we organize them in the same hierarchy as the bounding box hierarchy.

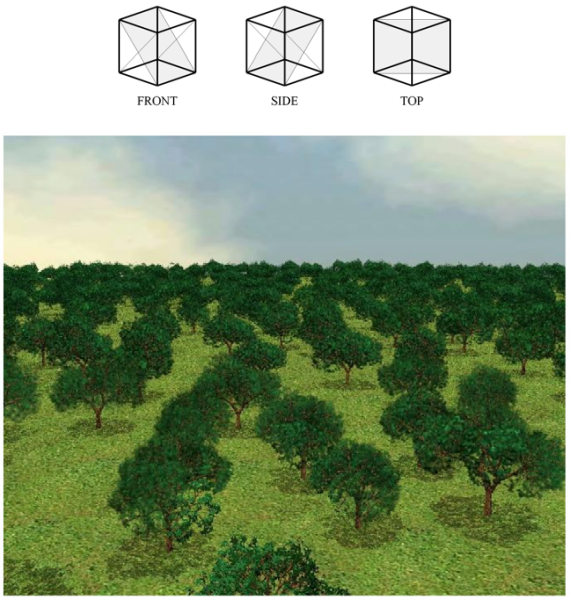

Figure 3: Top: Diagonal plane divisions used for leaves(a). Bottom: Multiresolution trees in a landscape(b).

4.2.1. Generating Bounding Boxes

The bounding box hierarchy is built at same time as the graph. *PUSH* and *POP* modules delimit the beginning and the end of a new branch and its new bounding box. The number of bounding boxes is controlled by a threshold called *top detail level*. During the interpretation of a chain, an *open box* is a bounding box whose dimensions are still being computed by traversing its sub-graph. The most recent box created is called *last box*. Finally, the *level* gives the current depth in the graph. In the Figure 2(b) is the algorithm that computes a bounding box hierarchy.

The algorithm runs as follows. When the interpreter finds a *PUSH* module, a new *open* box is created, initialized, and marked as the *last* box. As geometry nodes are encountered, the dimensions of all the *open* boxes are updated until a *POP* module appears. When the algorithm ends, each node has an associated bounding box containing some of the tree's geometry. Now we compute a set of images of the leaves located in each box. Later these images will be used as textures to draw impostors.

4.2.2. Generating Pre-computed Textures

We have to decide how to compute the impostor textures. First, we tried using six images. We computed six orthographic projections of the leaves of a box, one for each side of the box.

The problem of this solution is that the geometry does not look deep when the images are rendered as impostors. So we tried a modified version of the representation proposed in [21]. We divided the volume of the box into parallel slices and computed an image for each slice. With this approach, a better sense of depth was achieved, and we obtained smoother transitions between LODs. However, we had to use large amounts of storage for the textures and this substantially limited the number of different trees that could be rendered in a scene.

We finally decided to store six images obtained by projecting the leaves onto the six diagonal planes that cross a bounding box (see Figure 3(a)). This still requires a lot of storage, especially if the textures have an inadequate size or the number of bounding boxes generated is too large. An additional problem is texture memory in graphics cards, a limited resource. To reduce the amount of texture memory used by the representation, we limit the size of the texture images and the number of levels of the bounding box hierarchy.

4.2.3. Texture Size

Bounding boxes at the same level in the graph belong to the same LOD, and boxes in a level include all the boxes in higher levels. If the textures of a bounding box are rendered, then the boxes of its descendants are not rendered. So, if the observer is located far enough, the tree foliage is rendered using the images computed for the bounding box that includes the whole tree. Since the tree is far away its impostors only cover a few pixels. As the observer gets closer to the tree, smaller bounding box images are rendered. As the tree size grows on the screen, textures replacing the leaves are smaller; so they cover a number of pixels similar to the lower levels of detail at closer distances.

For this reason we decided to make the size of the textures constant for all levels. We use textures of 64x64 and 128x128 pixels. Texture size does not depend on the size of the box it represents. Suppose that we set the size to 128x128 pixels. If we want to store the images of a balanced binary tree with 10 levels, then we need 96Mbytes of memory. So we look for ways of reducing the number of levels.

4.2.4. Number of Levels

Consider the projected area of a set of leaves compared to the projected area of the texture replacing those leaves. If it covers more pixels than the texture covers, then the texture must be scaled up resulting in *blurring* or *aliasing* depending on whether texel colors are interpolated or not. To avoid this, we descend one more level in the hierarchy and generate textures for its children bounding boxes. To reduce the spatial cost of the model, we limit the number of texture levels taking into account the tree's topology. We compute the ratio of a node's bounding box volume to the root node's bounding box volume. If the result is smaller than a threshold, then images are not generated for that node and its descendants.

At rendering time we use blending to obtain smooth transition between the different impostors of a box. Otherwise, seams appear when rendering more than one impostor. To solve this problem we adjust the alpha channel of the polygons progressively, depending on the viewer's position.

4.3. Combining Both Representations

We describe how to obtain a tree model combining the two representations. We start designing an L-system using GREEN. The system is then derived and an output chain is obtained. This chain is used as input to two processes. The first process interprets the chain and generates the hierarchy of bounding boxes that represent the leaves. It then renders the images associated to each bounding box and stores them in compressed format. At rendering time the images will be used to replace the geometry of the leaves.

The second process takes the output chain and generates a multiresolution chain. The multiresolution chain is then interpreted as follows. *Advance* and *rotate* modules are interpreted in the same way they are interpreted in the output chain. *SAVE* modules store the turtle state in a vector indexed by *id*. *RESTORE* modules update the turtle state with the vector element given by id. To interpret the first n LODs of the multi-chain, we parse all the modules until the n-th is found. The next LOD can be progressively generated by interpreting the following sub-chain up to the next *RESTORE* module. The interpretation process generates a vector whose elements contain the geometry associated to each of the LODs of the multi-chain. The vector can be populated lazily depending on the requirements of the renderer. LOD k is extracted by reading the geometry stored in the first k elements of the vector. Therefore, the LOD extraction algorithm is very efficient.

5. Results

We tested our representation on a Pentium IV at 3.4 Ghz with 1 Gbyte of RAM and a GeForce Fx5700 graphics card. Building our representation for a scene with a more than a thousand trees takes less than 8 seconds. The size of the models is roughly 4 Mbytes. To transmit the model we compress the images using a standard format. As for the geometry, we can send the definition file of the L-system instead, since it only requires less than 1 Kbyte.

Figure 4(a) shows a tree model rendered at different distances using different LODs. Note that the visual structure of the tree is preserved by all four LODs. Furthermore, our representation satisfies all the multiresolution properties outlined in Section 2.3, but the last one. Still, this is not a major problem since changes between LODs are barely perceptible, especially if the tree is fairly leafy.

Figure 4: (a)A tree model rendered at different distances using different LODs: top, without leaves and bottom, with leaves. (b)Different LODs of a tree model in a landscape.

In Figure 4(b) we show different LODs of the same tree. In the upper left image the individual leaves near the viewer are represented by one polygon. The other leaves of the tree are rendered using the pre-calculated images. Figure 3(b) shows a scene we used to time a walkthrough. The resulting frame rates are summarized in Table 1. The tree model contains 2047 branches, 4085 leaves, and 10409 polygons. The reader is referred to [30] for different animations that show the performance of our system.

Number of Trees		100	250	500	750	1000	1500	2000
Multiresolution Model	fps (min)	69	57	34	12	11	8	6
	fps (max)	76	73	68	39	66	47	32
	fps (average)	73,0	64,8	52,9	24,9	20,6	18,6	14,9
Geometric Model	fps (min)	3	1	0,1	0,1	0,1	0,1	0,1
	fps (max)	30	18	10	8	5	2	1
	fps (average)	14,3	6,3	2,4	1,5	0,6	0,1	0,1

Table 1: For a scene made of different numbers of trees, we compare the frame rates of our multiresolution representation with the frame rates of a geometry-based representation: a) average fps, b) minimum fps and c) maximum fps.

6. Conclusions and Future Work

We have introduced a new representation for plant and tree modeling and rendering. Our representation separately handles the branches and the leaves of a tree. To model the trunk and branches, we propose a new procedural multiresolution representation that represents trees at different LODs using a chain of parameterized symbols.

We construct the chain by analyzing the tree and choosing those branches that best represent its visual structure. The processing algorithm is based on a metric that quantifies the ramification of the branches of the tree. The length of the resulting multiresolution chain is similar to the length of the original chain. It contains the LODs sorted from coarsest to finest. An LOD builds on the sub-chain of the previous LODs, and can be progressively transmitted and rendered. Interpretation of the multiresolution chain produces a geometry data structure that allows efficient LOD extraction for rendering.

To model the leaves of a tree we propose an image-based multiresolution representation. Images are pre-computed for sets of leaves enclosed in bounding boxes. The bounding boxes are generated following the structure of the tree. Depending on viewer distance, the leaves can be rendered using their geometry or using one or more pre-computed images. The amount of memory used for these images is independent of the leafiness of the tree.

Our hybrid representation can be generated automatically in a few seconds, given an L-system. Scenes made of more than thousand trees can be rendered at interactive rates. Our models can be viewed from any position and direction without noticeable changes in visual quality. The representation preserves the general structure of the tree and supports variable multiresolution and progressive transmission. Any graphics application requiring rendering of outdoor scenes can benefit from our representation. For example, virtual reality, simulation, navigational aids and computer games would be good candidates.

We plan on improving our representation in three ways. We want to implement smooth the transitions between LODs to remove the few popping artifacts that our representation produces. We also want to reduce the amount of memory needed to store the images of the model. Finally, we want our representation to efficiently support wind movement.

Acknowledgements

The work described in this paper was partially supported by grant TIN2009-14103-C03-03 and MICINN-P19/08 of the Spanish Ministry of Sciencie and Innovation.

References

[1] SpeedTree. [link].
URL http://www.idvinc.com/
[2] A. Lindenmayer, Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs, Journal of Theoretical Biology 18 (3) (1968) 300–315. doi:10.1016/0022-5193(68)90080-5.
[3] P. Prusinkiewicz, A. Lindenmayer, J. Hanan, Development models of herbaceous plants for computer imagery purposes, in: SIGGRAPH, 1988, pp. 141–150.
[4] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants, Springer, 1991.
[5] J. S. Hanan, Parametric l-systems and their application to the modeling and visualization of plants, Ph.D. thesis, University of Regina (1992).
[6] Y. I. H. Parish, P. Müller, Procedural modeling of cities, in: SIGGRAPH, 2001, pp. 301–308.
[7] D. Schmalstieg, M. Gervautz, Modeling and rendering of outdoor scenes for distributed virtual environments, in: VRST, 1997, pp. 209–215.
[8] B. Lintermann, O. Deussen, Interactive modeling of plants, Computer Graphics and Applications, IEEE 19 (1) (1999) 56–65.
[9] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, S. B. Kang, Image-based plant modeling, ACM Trans. Graph. 25 (3) (2006) 599–604.
[10] A. R. Martinez, I. Martín, G. Drettakis, Volumetric reconstruction and interactive rendering of trees from photographs, ACM Trans. Graph. 23 (3) (2004) 720–727.
[11] P. Tan, G. Zeng, J. Wang, S. B. Kang, L. Quan, Image-based tree modeling, ACM Trans. Graph. 26 (3) (2007) 87.
[12] P. Prusinkiewicz, M. James, R. Mech, Synthetic topiary, in: SIGGRAPH, 1994, pp. 351–358.
[13] R. Mech, P. Prusinkiewicz, Visual models of plants interacting with their environment, in: SIGGRAPH, 1996, pp. 397–410.
[14] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, P. Prusinkiewicz, Realistic modeling and rendering of plant ecosystems, in: SIGGRAPH, 1998, pp. 275–286.
[15] J. Weber, J. Penn, Creation and rendering of realistic trees, in: SIGGRAPH, 1995, pp. 119–128.
[16] D. Marshall, D. S. Fussell, A. T. C. III, Multiresolution rendering of complex botanical scenes, in: Graphics Interface, 1997, pp. 97–104.
[17] X. Zhang, F. Blaise, M. Jaeger, Multiresolution plant models with complex organs, in: VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, ACM, New York, NY, USA, 2006, pp. 331–334. doi:http://doi.acm.org/10.1145/1128923.1128980.
[18] C. Rebollo, I. Remolar, M. Chover, O. Ripolls, An efficient continuous level of detail model for foliage, in: U. of West Bohemia (Ed.), Journal of WSCG, Plzen (Czech Republic), 2006, iSBN 80-86943-05-4.
[19] N. L. Max, Hierarchical rendering of trees from precomputed multi-layer z-buffers, in: Rendering Techniques, 1996, pp. 165–174.
[20] N. May, O. Deussen, B. Keating, Hierarchical image-based rendering using texture mapping hardware, in: Rendering Techniques, 1999, pp. 57–62.
[21] A. Meyer, F. Neyret, Interactive volumetric textures, in: Rendering Techniques, 1998, pp. 157–168.
[22] A. Meyer, F. Neyret, P. Poulin, Interactive rendering of trees with shading and shadows, in: Rendering Techniques, 2001, pp. 183–196.
[23] P. S. Heckbert, M. Garland, Multiresolution modeling for fast rendering, in: Proc. Graphics Interface '94, Canadian Inf. Proc. Soc., Banff, Canada, 1994, pp. 43–50.
[24] E. Puppo, R..Scopigno, Simplification, lod and multiresolution principles and applications, in: EUROGRAPHICS 97, Vol. 16 of Computer Graphics Forum, 1997.
[25] K. Perlin, L. Velho, Live paint: painting with procedural multiscale textures, in: SIGGRAPH, 1995, pp. 153–160.
[26] A. Rosenfeld, Multiresolution Image Processing and Analysis, Springer-Verlag, 1984.
[27] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle, Multiresolution analysis of arbitrary meshes, in: SIGGRAPH, 1995, pp. 173–182.
[28] T. He, L. Hong, A. E. Kaufman, A. Varshney, S. W. Wang, Voxel based object simplification, in: IEEE Visualization, 1995, pp. 296–303.
[29] A. A. diSessa, H. Abelson, Turtle Geometry: the computer as a medium for exploring mathematics, MIT Press, Cambridge, MA, 1981.
[30] J. Lluch. [link].
URL http://www.sig.upv.es/evergreen