Procedural Multiresolution for Plant and Tree Rendering

Javier Lluch
jlluch@dsic.upv.es

Emilio Camahort
camahort@dsic.upv.es

Roberto Vivó
rvivo@dsic.upv.es

Computer Graphics Section
Department of Information Systems and Computation
Polytechnic University of Valencia
46020 Valencia, Spain

Abstract

Modeling and rendering of plants and trees requires generating and processing large numbers of polygons. Geometry simplification methods may be used to reduce the polygon count and obtain a multiresolution representation. However, those methods fail to preserve the visual structure of a tree. We propose a different approach: procedural multiresolution. We build procedural models that reflect a tree's visual structure at different resolution levels. The models are based on parametric L-systems. Our method takes a parametric chain representing a tree and generates a new chain with embedded multiresolution information. The algorithm is based on a metric that quantifies the relevance of the branches of a tree. The representation supports efficient geometry extraction and produces good visual results.

CR Categories: F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems: Parallel rewriting systems, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, I.6.3 [Simulation and Modeling]: Applications.

Keywords: image synthesis, modeling of plants, L-system, multiresolution, level-of-detail (LOD).

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 videogames. Their representation is commonly made of procedural models based on L-systems. Current graphics applications convert those models to polygon-based representations for hardware-accelerated rendering on graphics engines.

Accurate modeling of plants and trees requires large numbers of polygons. A typical outdoor scene contains many plants and 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. To reduce the amount of geometry rendered we need software-based strategies, like multiresolution modeling.

Multiresolution modeling reduces the polygon count by using geometry-based simplification methods. These methods, however, fail to capture the nature of plants and trees. They tend to prune the youngest branches, thus resulting in a smaller tree visually different from the original one. More natural simplification methods can be obtained by implementing multiresolution at the procedural level.

In this paper, we propose a new multiresolution representation based on parametric L-systems. We build it at the procedural level using a chain of parameterized symbols. Our algorithm relies on a metric that captures the nature of a tree. We interpret the chain to produce the geometry of a tree at different resolutions. The result is a multiresolution model that reflects its visual structure. We call the new representation *procedural multiresolution*.

In the following section, we survey related work in procedural and multiresolution modeling. Then we give brief introductions to parametric L-systems and geometric multiresolution. In the fourth section, we develop our model and show how to produce a multiresolution procedural representation. We explain how this representation is interpreted graphically while generating the levels of detail. In the last two sections, we present our results and conclusions.

2 Related Work

Our work is primarily related to two different areas of computer graphics: plant and tree modeling and multiresolution representations. The most commonly used modeling technique for plants and trees is parametric L-systems. Lindenmayer first introduced L-systems for modeling cellular division [Lind68]. Later, Prusinkiewicz et al. applied L-systems to the representation of plants and trees [Prus88]. Parametric L-systems were reported in the works by Prusinkiewicz and Lindenmayer [Prus90] and by Hanan [Hana92].

Recent developments in plant and tree modeling allow the simulation of complex phenomena, like the interaction between the development of a plant and its surroundings [Prus94], the information exchange between plants and their environment [Prus96], and the representation and rendering of plant ecosystems [Deus98]. L-systems have also been applied to the representation of artificial growth models, like city growth models [Pari01].

Multiresolution is a modeling technique that was first introduced to accelerate rendering of complex geometries [Heck94][Pupp97]. Perlin and Velho use procedural multiresolution in their Live Paint texture editing and composition system [Perl95]. Their system allows the representation and composition of procedural multi-scale textures with an arbitrary number of levels of detail. Additional, multiresolution techniques have been proposed for image representations [Rose84], curve and surface modeling [Eck95], and volumetric data sets [He95]. Some techniques such as degradation at range [Webe95], space partition [Mars97], LDI's [Max96][Max99], volumetric textures [Meye98] or bidirectional textures [Meye01] have been specifically proposed for tree representation.

3 Modeling Plants and Trees Using L-Systems

Parametric L-systems are the most commonly used method for modeling plants and trees. An L-system is given by an axiom and

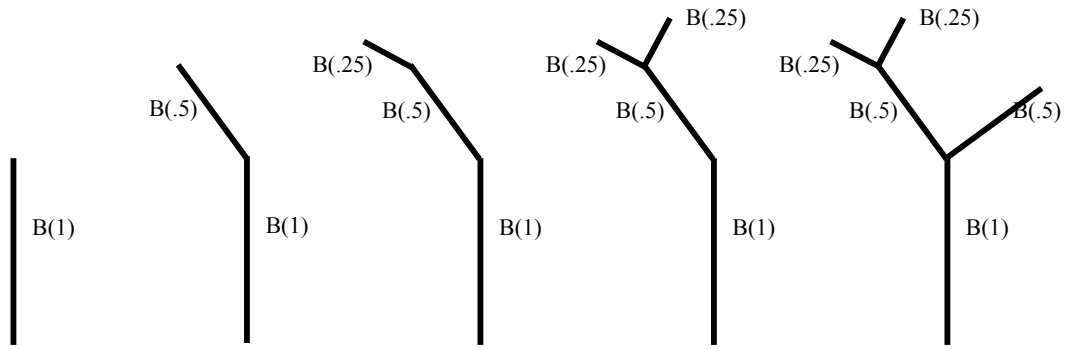

Figure 1: The sequence of trees obtained during the interpretation of chain (2).

a set of derivation rules. The modeling process starts with the derivation of the axiom and the generation of a derived chain. The derived chain is then interpreted to generate a geometric model suitable for rendering. Both steps, derivation and interpretation, are described in this section.

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 children. 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.

Axiom: A(length)
Rule1: A(l) : itNum < maxIt → B(l)[A(l/2)A(l/2)] (1)
Rule2: A(l) : itNum = maxIt → B(l)

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. The derivations can be applied in parallel to the modules. 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)] (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 [Prus90] and [Hana92] 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 [Abel82].

A turtle is characterized by a *current position,* a *current orientation* and a *stack structure*. The position and orientation indicate the current state of the turtle. The state can be changed by one of two types of modules: *advance* and *rotate*. An *advance* module moves the turtle from the current position along the current orientation. It typically generates a graphics primitive, like a cone or a cylinder, representing a branch. A *rotate* module changes the turtle orientation without generating a primitive.

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. The branch may have its own children, followed by a closing bracket].

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. Figure 1 illustrates the construction of the tree represented by chain (2).

4 Geometric Multiresolution

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 triangles.

A good multiresolution model is characterized by the following four properties.

- The size of the model must not increase with the number of LODs.
- Extraction of the LODs must be fast enough to support interactive rendering.
- There must be no loss of information: the finest LOD must reproduce the original model at its full resolution.
- The changes between LODs must be smooth.

A simplification method is used to generate a model's LODs. It starts with the full-resolution model, and decides what information is the least important for the model. It eliminates that

information, and obtains a coarser representation with a smaller number of polygons. Successive applications of the simplification method generate coarser LODs until a certain limit is reached.

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.

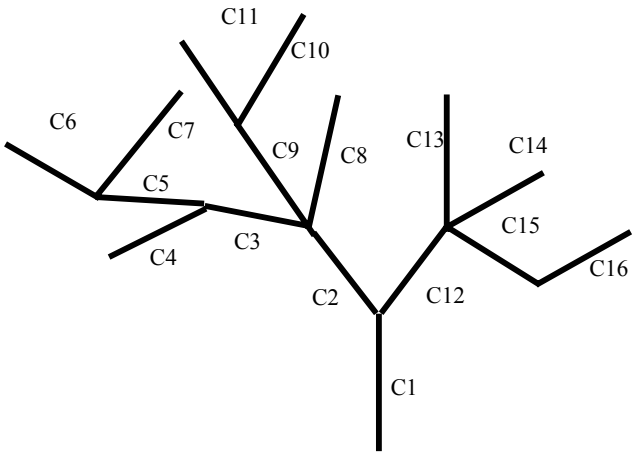

Figure 2: A possible interpretation of chain (3).

5 Procedural Multiresolution

We study alternative simplification methods for procedural tree models. Our goal is a procedural-based multiresolution representation that maintains the structure of the tree. The first question is: given an output chain, 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:

C1[C2[C3[C4][C5[C6][C7]]][C9[C11][C10]][C8]][C12 [C13][C14][C15[C16]]] (3)

where *Cn* represents the chain of modules that generates branch *n* (see Figure 2 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.

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 *weighted tree*. The weighted tree 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 weighted tree 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.

In this section, we propose a metric that preserves a tree's visual structure when generating its LODs. We show how to build the weighted tree and how we use that metric to establish an order relation on its branches. Finally, we present algorithms to generate and interpret the multi-chain.

5.1 Selecting a Metric

Consider an input chain *S*. We construct a weighted tree *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 metrics that associate larger values to those branches that better reflect the visual structure of the botanical tree. There are a number of possible metrics and all of them must fulfil the property to be an order relation between nodes. We have considered four of them: (a) number of children, (b) number of descendants, (c) longest path to a leaf, and (d) branching length. Formally, given the node $n \in T$, those metrics can be expressed as:

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

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

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

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

Where *{n}* is de 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. Eventually, the path-length *lX(n,m)* is the sum of every length in the path *X(n,m)* between nodes *n* and its descendant *m*. This path-length is useful to update the node weight as we will see below in (4).

5.2 Building the Weighted tree

The weighted tree 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 following algorithm parses the input chain and builds the weighted tree.

```
Create the root of the weighted tree and make it
the current node
  For each module in the input chain
    In case the module is
      [: add a child to the current node and make
         it the current node
      ]: add the branching length of the current
         node to its parent's
         set the current node to the current
         node's parent
```

```
otherwise: add the module's length to the
   current node's branching length
   add the module to the current node's
   chain
```

Figure 3 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.

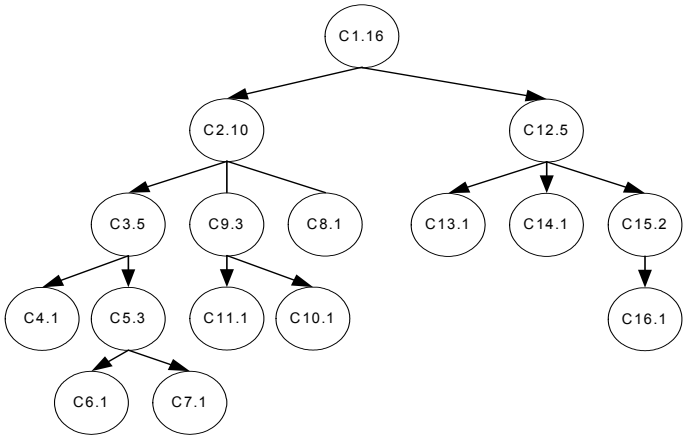

Figure 3. Weighted tree obtained from chain (3) assuming that all the branches are of unit length.

5.3 Generating the Multiresolution Chain

Given the weighted tree, 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 weighted tree. 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 *k.L* be the branching length of node *k*. We update the branching lengths as follows:

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

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 *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 weighted tree 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 weighted tree have been output.

For example, given the weighted tree of Figure 3, the algorithm generates the following multi-chain:

C1 SAVE(1) C2 SAVE(2) C3 SAVE(3) C5 SAVE(5) C6 RESTORE(2) C9 SAVE(9) C11 RESTORE(1) C12 SAVE(12) C15 C16 RESTORE(3) C4 RESTORE(5) C7 RESTORE(9) C10 RESTORE(12) C13 RESTORE(2) C8 RESTORE(12) C14 (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.

5.4 Interpreting the Multiresolution Chain

We describe how the multi-chain is interpreted. *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.

For example, the interpretation of multi-chain (5) generates the following nine LODs:

LOD 1: C1 SAVE(1) C2 SAVE(2) C3 SAVE(3) C5 SAVE(5) C6
LOD 2: LOD 1 + RESTORE(2) C9 SAVE(9) C11
LOD 3: LOD 2 + RESTORE(1) C12 SAVE(12) C15 C16
LOD 4: LOD 3 + RESTORE(3) C4
LOD 5: LOD 4 + RESTORE(5) C7
LOD 6: LOD 5 + RESTORE(9) C10
LOD 7: LOD 6 + RESTORE(12) C13
LOD 8: LOD 7 + RESTORE(2) C8
LOD 9: LOD 8 + RESTORE(12) C14

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.

6 Results

We took a tree modeling and rendering application based on L-systems, and modified it to support procedural multiresolution. Our implementation takes a tree's output chain, converts it to a multi-chain, and generates the geometry of all the LODs of the tree. The rendering algorithm selects a tree's LOD as a function of the distance from the tree to the observer.

	Generic Tree		Acacia	
LOD	#Branches	#Polys	#Branches	#Polys
Coarsest	27	172	57	255
Medium	233	872	414	1430
High	638	2103	805	2605
Finest	1023	3252	2048	6334

Table 1: Number of polygons and branches for the trees in Figures 4 and 6.

Figures 4 and 6 show close views of two trees, a generic tree and an acacia, at different LODs. Table 1 contains the number of branches and polygons of each model. Figures 5 and 7 show the models' LODs at the distance they would be rendered. The bottom row of each figure was rendered with leaves to increase realism. Note that the visual structure of the trees is preserved by all four LODs. Furthermore, our representation satisfies all the multiresolution properties outlined in Section 4, but the last one.

7 Conclusions

We propose a new procedural multiresolution representation that represents trees at different LODs using a chain of parametric 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 visualization. We have tested our method by generating trees at various resolutions with smaller numbers of polygons. Our algorithms and data structure produce LODs that preserve the visual structure of the tree.

As future work, we are studying a multiresolution model that includes the leaves of a tree. We are also considering implementing smooth transitions between LODs. Our procedural multiresolution method may also be applicable to other natural phenomena, particularly those modeled using L-systems.

Acknowledgements

This work was supported by TIC1999-0510-C02-01 research project.

References

[Abel82] H. Abelson and A. A. diSessa. Turtle geometry. M.I.T. Press, Cambridge, 1982.

[Deus98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomir Mech, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. Proceedings of SIGGRAPH 98 (Orlando, Florida, July 19-24, 1998). In Computer Graphics Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 275-286.

[Eck95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, August 1995, pages 173-182.

[He95] T. He, L. Hong, A. Kaufman, A. Varshney and S. Wang. Voxel-based object simplification. Proceedings of Visualization 95, IEEE Computer Society Press, 1995.

[Heck94] Paul Heckbert and Michael Garland. Multiresolution Modeling For Fast Rendering. In Proceedings of Graphics Interface '94, pages 43-50, Canadian Information Processing Society, 1994.

[Lind68] Aristid Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18:280-315, 1968.

[Pari01] Yoav I. H. Parish and Pascal Mueller. Procedural Modeling of Cities. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17, 2001). In Computer Graphics Proceedings, Annual Conference Series, 2001, ACM SIGGRAPH, pp 301-308.

[Perl95] Ken Perlin and Luiz Velho. Live paint: Painting with procedural multiscale textures. In Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pages 153-160, August 1995.

[Prus88] Przemyslaw Prusinkiewicz, Aristide Lindenmayer and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988), in Computer Graphics 22, 4 (August 1988), pages 141-150, ACM SIGGRAPH, New York, 1988.

[Prus90] Przemyslaw Prusinkiewicz and Aristide Lindenmayer. The algorithmic beauty of plants. Springer-Verlag, New York, 1990.

[Hana92] J. S. Hanan. Parametric L-systems and their application to the modeling and visualization of plants. PhD thesis, University of Regina, June 1992.

[Mars97] D. Marshall, D. S. Fussell, and A. T. Campbell. "Multiresolution rendering of complex botanical scenes". In Proceedings of Graphics Interface 97, 97-104, 1997

[Max96] N. Max, "Hierarchical rendering of trees from precomputed multi-layer Z-buffers", in X. Pueyo and P. Schröder, editors, Rendering Techniques 96, pages 165-174, Springer Wien, 1996

[Max99] N. Max , O. Deussen, B. Keating, "Hierarchical Image-Based Rendering using Texture Mapping Hardware", in EG Workshop on Rendering'99,57-62, Springer Wien, 1999

[Meye98] A. Meyer, F. Neyret, "Interactive volumetric textures", in EG Rendering Workshop'98,157-168, 1998

[Meye01] A. Meyer, F. Neyret, P. Poulin, "Interactive Rendering of Trees with Shading and Shadows", in EG Workshop on Rendering'01, 2001

[Prus94] Przemyslaw Prusinkiewicz, Mark James, and Radomir Mech. Synthetic topiary. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In Computer Graphics Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, pp. 351-358.

[Prus96] Radomir Mech and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. Proceedings of SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 397-410

[Pupp97] E. Puppo and R. Scopigno. Simplification, LOD and Multiresolution Principles and Applications. In Proceedings of EUROGRAPHICS 97. CG Forum, vol. 16, no. 3, 1997.

[Rose84] Azriel Rosenfeld. Multiresolution Image Processing and Analysis. Springer-Verlag, Berlin, 1984.

[Webe95] J. Weber and J. Penn, "Creation and Rendering of Realistic Trees", In Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pages 119-128, August 1995

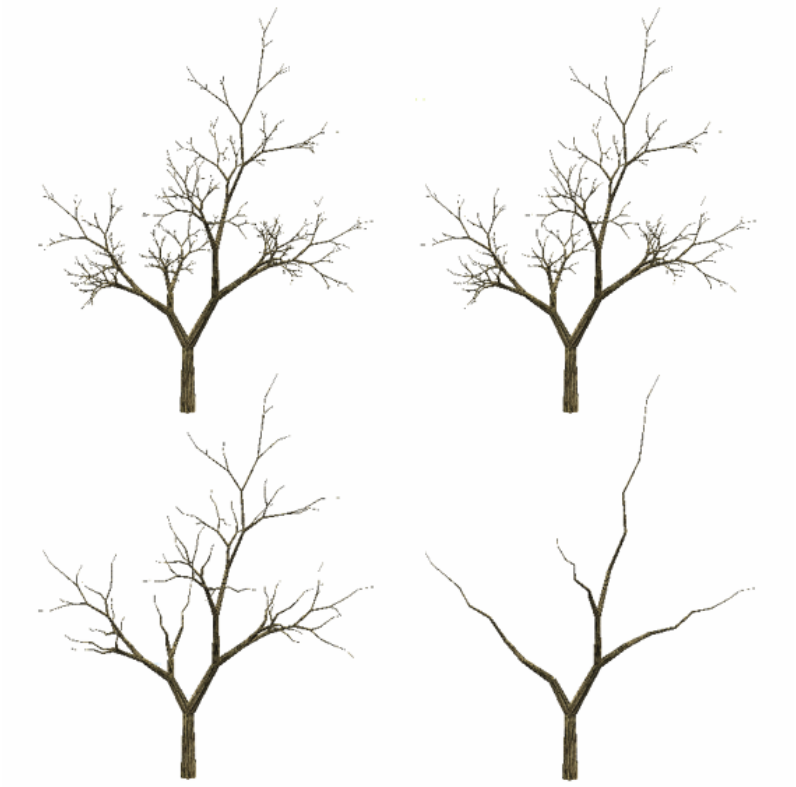

Figure 4: A tree and three finer LODs generated with our method. The images represent finest, high, medium and coarsest LODs, listed in Table 1, sorted from left to high and from top to down.

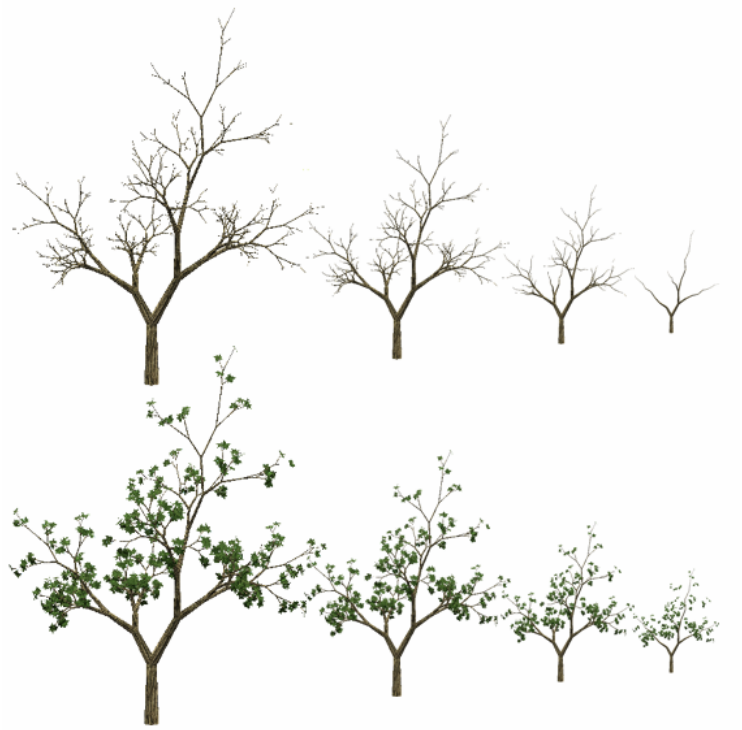

Figure 5: The trees of Figure 4 rendered at proper distances according to their resolution.

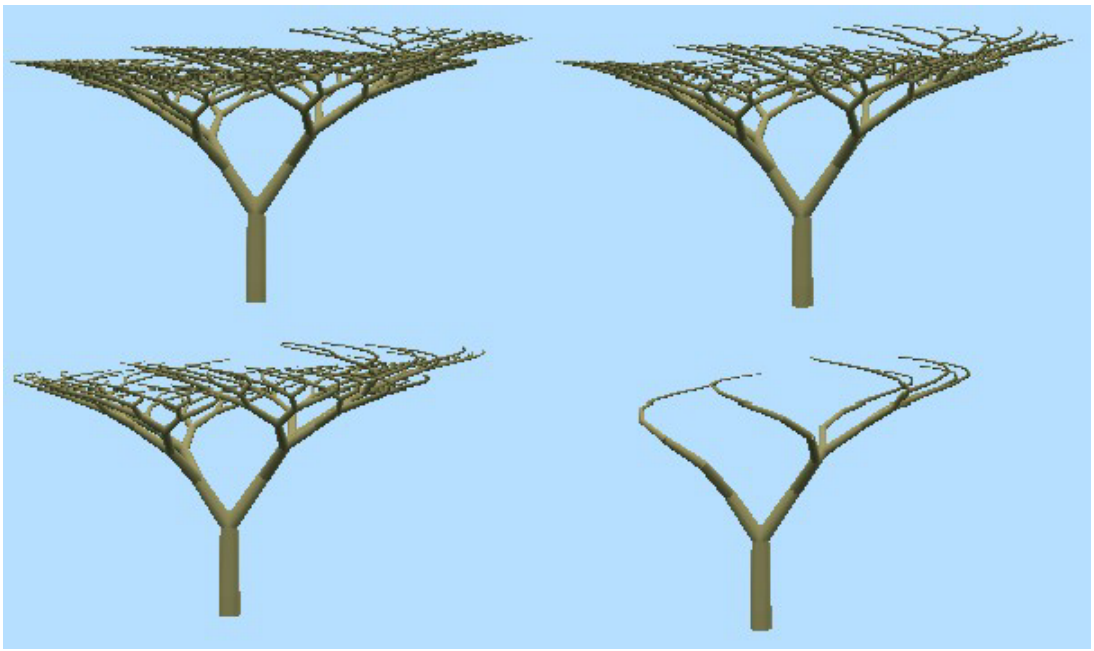

Figure 6: Four different LODs of an acacia model. The images represent finest, high, medium and coarsest LODs, listed in Table 1, sorted from left to high and from top to down.

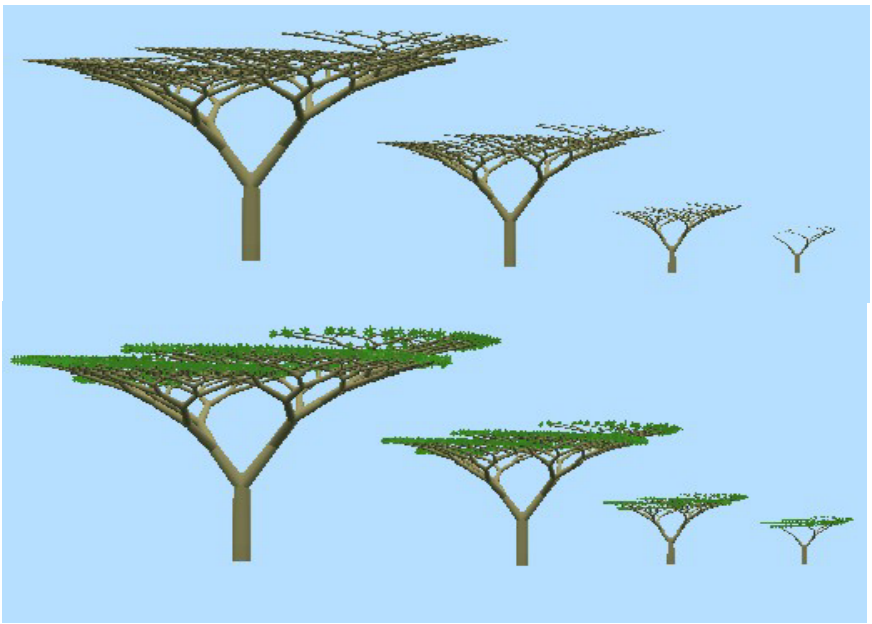

Figure 7: The acacia LODs rendered at proper distances according to their resolution.