1-2-tree: Semantic Modeling and Editing of Trees

Björn Ganster, Reinhard Klein

Universität Bonn
Email: {ganster,rk}@uni-bonn.de

Abstract

In computer graphics, procedural methods and L-systems are common approaches to model complex botanical trees. In contrast to previous tree modeling systems, we propose linking rules, parameters and geometry to semantic entities. This has the avantage that when an entity is clicked in the viewport, its parameters can be displayed immediately, and viewport editing operations can be reflected in the parameter set. Furthermore, we store the entities in a hierarchical data structure and allow the user to activate recursive traversal via selection options for all editing operations. Therefore, viewport or parameter changes can be applied to a single entity or many entities at once, and only the geometry for the affected entities needs to be updated. The proposed user interface aims at simplifying the modeling process.

1 Introduction

Modeling trees has been a topic for decades in computer graphics, with applications in areas such as films, computer games, advertising and architecture visualization, because outdoor scenes usually include trees. Current tree modeling tools are often powerful, but difficult to handle, or they are easy to use, but unable to capture the uniqueness of particular trees. Research in this area has reached a point where we can focus on making the modeling process easy for users without a background in computer graphics. For that reason, we aim to balance very powerful, but difficult-to-use systems with simpler, less powerful approaches.

Procedural systems for modeling trees use a principle for which Alvy Ray Smith coined the term data amplification paradigm [29]: It describes how complex models can be constructed from a small set of rules and some input parameters. Unfortunately, previous tree modeling systems create the geometry as an unstructured list of polygons, without associating rules with their geometry instances.

Linking the geometry closely with the parameters used for geometry construction is a strength of conventional modeling systems like 3DS Max, Maya, or Cinema4D, because local changes can be made without affecting the whole model or entailing a complete reconstruction. However, due to the sheer number of primitives, exclusively modeling a complex tree in this manner is impractical.

In the sketch-based modeling approach, the user creates a rough sketch of an object, and algorithms infer a 3D model from this input. This shifts the responsibility for managing the primitives to the computer, but altering the model requires a new sketch or different tools.

By integrating the procedural, conventional modeling and sketch-based approaches into a single system, we combine their strengths and eliminate their weaknesses: We use a procedural model to create instances, but we link the parameters with their geometry instances to form semantic entities in a hierarchical data structure. This permits us to change single instances or to perform mass updates after geometry creation without reconstructing the entire model. Sketch-based modeling yields additional methods for input, and the user can model on a semantic level after sketching. We call this semantic modeling because we are dealing with entities rather than abstract rules or raw geometry.

We use a single hierarchical data structure to store both the geometry and parameters for all branches of a tree. This paper's main contributions are exposing the hierarchy traversals during updates as selection options and retrieving branch parame-

ters from interactive changes in the viewport. This paper proposes a user interface and algorithms to ease modeling trees, but it does not aim at creating more realistic models.

2 Related Work

2.1 Procedural modeling of trees

Aristid Lindenmayer [12] introduced a notation for modeling growth processes in 1968. Named after their inventor, L-systems apply rewriting rules to an axiom to produce a linear symbolic description of a plant, the L-string. An L-system requires that all matches for a rule are applied in parallel, and an input parameter states the number of parallel rewriting steps to perform before the system terminates. The next stage in the algorithm produces geometry from the L-string. Lindenmayer and Prusinkiewicz describe this process with several extensions in their book [21]. Prunsinkiewicz et al published a series of extensions: interaction with the environment [15], an application to topiary [22], and the use of positional information [23]. Positional information is taken from a 2D function editor to change parameter values along the parent branch.

Honda describes a procedural approach that created the first 3D plant skeletons [9, 2]. Reeves invented particle systems, and describes an application to trees together with Blau [26]. Oppenheimer proposed a recursive algorithm inspired by Mandelbrot's fractals [19]. He needs few parameters to describe trees because he uses recursion factors. Bloomenthal demonstrated how to produce realistic branches [3]. de Reffye et al animate tree growth using procedural models [5]. Greene showed how to use voxel grids to speed up queries that prevent self-intersection and allow to simulate heliotropism [7]. Holton proposed the strand model to calculate realistic branching angles, radii and branch lengths [8].

Weber and Penn propose a parameterization for trees designed for use by non-experts [31]. In particular, they replace the recursion factors used by Oppenheimer with 13 parameters that apply to all branches on the same level of recursion. Therefore, parameter changes always affect all branches on the same level. There are special parameters with a 'V' suffix which define a maximum random deviation from the base parameter. The paper provides many examples demonstrating that the parameters suffice for modeling a wide range of trees.

Lintermann and Deussen designed a graphical programming language to describe trees, flowers and other plants [13]. Parameters can be varied by assigning mathematical functions which evaluate iteration counts or use pseudo random number generators.

Power et al apply an inverse kinematics optimization technique to create a natural branch form while a user drags a branch, and branches can be bent, rotated or pruned in the viewport [20].

Since Weber and Penn published their system, computers have evolved to a point where it is possible to store parameters for individual branches, as demonstrated by Boudon et al [4] for L-systems. Their system collects parameters in a hierarchical data structure called decomposition graph. The parameters can be inherited to child branches, similar to the inheritance mechanism used in object-oriented programming languages. The user may edit the silhouette of a branch including sub-levels, and an L-system fills it with the next level of branches and leaves.

2.2 Image-based modeling of trees

More recently, research has focused on reproducing existing plants from images [25, 27, 28]. Quan et al's system allows the user to draw and move curves, to edit the radius of a branch, or to place leaves in the 3D viewport [24]. The system by Neubert et al [16] requires only few input images and permits sketching. The user provides only approximate positions for the images, and no registration is needed.

2.3 Sketch-based modeling of trees

Sketch interfaces allow a user to create a 2D sketch from which the computer derives a 3D model. This has been demonstrated for trees, flowers [11] and phyllotactic arrangement of plant organs [1]. Okabe et al presented a system that deduces a 3D model of

a b c d e

Figure 1: Modeling a plane tree in 1-2-tree. a) The trunk and the main branches are modeled individually. b+c+d) Further branches are added using mass updates. e) Result after adding leaves.

a tree from a sketch, by assuming that trees spread branches so that the distances between the branches are maximized [18]. The system integrates three example-based interaction techniques that can be used to alter the model after the geometry has been created. The Sketch L-system by Ijiri et al allows the user to edit production rules in the 3D viewport [10]. The system applies the production rules interactively while the user sketches the trunk.

3 Overview

Our system treats tree organs such as the trunk, branches, twigs, roots, leaves, blossoms and fruits as semantic entities. The entities have simple parameters such as their size and color. Further parameters control placement and orientation of the tree organs. In this paper, we focus on the branches and leaves as the most important entities that define the tree structure and visual appearance, but future implementations could include other entities. As in other publications, the term “branches” includes the trunk, any twigs, and even roots.

The entities are stored in a simplified scene graph [30], where all nodes represent branches. Algorithms that alter parameters or update geometry have recursive scene graph traversal as a built-in feature. We expose the recursive data structure updates to the user as selection options. Thus, the user may choose to apply parameter changes to a single branch or several branches at once. A notation for positional information ensures variety.

Every entity has two representations: one is its parameterization, the other is its geometry. We store both representations in the scene graph node. This has a number of benefits: When the user selects an entity in the viewport, our system displays its parameters. When the user edits the geometry in the viewport, our system adjusts the entities’ parameters. Any changes to the parameters or the geometry are interactively applied to the other representation and optionally to other instances. Only the geometry for the affected nodes needs to be rebuilt.

The new system is designed to allow for fast and easy creation of trees, thus its name 1-2-tree (“easy as counting one-two-t(h)ree”).

4 Using 1-2-tree

We propose the following user interface: When the user starts the system, the trunk for a new tree is displayed. Branches are added by setting the number of child branches to nonzero or by sketching them in the viewport. All branches can be edited directly in the viewport or by adjusting their parameters. For capturing the uniqueness of specific trees, it often makes sense to model the trunk and some of the main branches individually, but as modeling a tree progresses, minor branches and twigs are usually best edited using mass updates. Leaves are usually added as a last step, as they hide the tree’s inner structure. See Figure 1 for an example.

4.1 Selection

The user may select branches in the viewport. In order to perform mass updates, further branches can be added to the selection using the following selection modes:

- Same Level: Selects all branches on the same branch level,
- Recursive: Adds the selected branches' children to the selection.

If both options are active, changes affect all branches on the same level and below the currently selected branch. If the trunk is selected and recursive selection is active, editing operations affect the entire tree.

There may be several kinds of child branches on a branch. For the trunk, there may be a few main branches, many small branches near its top, and the roots can be modeled as branches, too. The user may define groups of branches by assigning tags, and may limit updates to branches that carry the selected branch's tag. With properly assigned tags, it is therefore possible to edit only the main branches, for example.

Since all editing operations are applied to all selected branches, the selection modes permit the user to perform powerful editing operations in an intuitive manner.

4.2 Viewport tools

The branches are defined by control points and radii, which are stored in the scene graph node. The following tools may be used to manipulate the control points in the viewport:

The "Move" tool allows moving branch control points. All following control points on the same branch, its child branches and all branches in the selection are translated by the same vector. Another viewport tool interactively rotates the currently selected branches against their parents in the viewport. The parameters "Length" and "Angle against Parent" are updated for the entire selection.

The "Sketch" tool allows for sketching branches directly in the viewport. The user moves the mouse over the parent branch to select a starting point and then sketches the new branch. After sketching, the new branch lies in a plane, but the user can move the control points from a different perspective.

The currently selected branch can be deleted by pressing the DEL key. The user may delete the trunk in this manner and sketch a new one. The "Saw" tool can be used to cut off branch parts at the mouse position. It was implemented as a replacement for pruning.

4.3 Parameters

While sketching and moving control points works well for editing individual branches, editing many branches at a time can often be accomplished more easily by manipulating parameters. Our parameters are very similar to those of Weber and Penn [31], except that we store individual parameters for each branch. In order to increase variety during mass updates, we created a simple notation that allows the user to combine parameter increases or decreases along the parent branch with a noise function. This is a simplified form of positional information [23]. Please refer to Appendix A for details on the notation and the parameter list.

For many trees, smaller branches and leaves are concentrated near the hull of the tree. We model this observation by introducing the "anchor values" parameter: Every branch's anchor value selects the control points to use for placing the branch. The starting point $\vec{q}$ for a child branch with anchor value a and n parent branch control points $p_0, ..., p_{n-1}$ is computed by interpolation

$$\vec{q} = \begin{cases} \vec{p}_k + (j-k)(\vec{p}_{k+1} - \vec{p}_k) & \text{if } 0 < a < 1 \\ \vec{p}_0 & \text{if } a \leq 0 \\ \vec{p}_{n-1} & \text{if } a \geq 1 \end{cases}$$

where $j = a(n-1)$ and $k = \lfloor j \rfloor$. The user may restrict branches and leaves to the outer areas of a tree by specifying an interval for the anchor values. For example, assigning "0.5-1" as an anchor value will place branches only on the outer half of a branch.

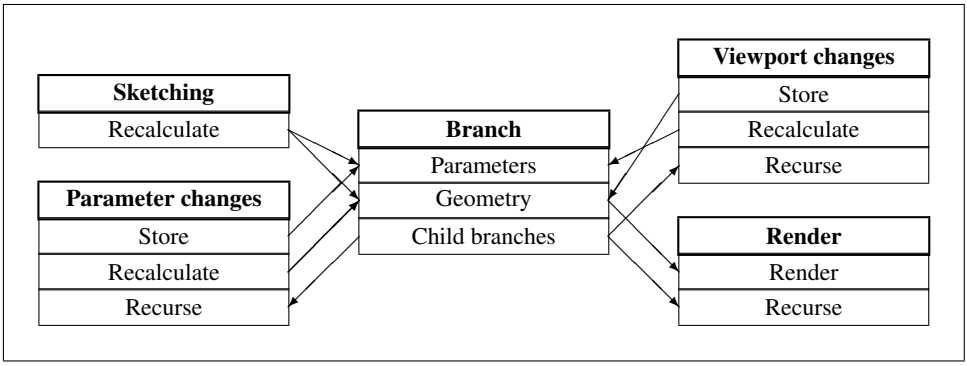

Figure 2: Processes affecting the tree's central data structure: scene graph node type Branch

5 Algorithms

Figure 2 summarizes how moving control points, sketching, changing parameters, and rendering affect scene graph nodes for branches. For any viewport changes, the geometry is updated, and new parameters for all currently selected branches are deduced. While the user is sketching, 1-2-tree recalculates the parameters and vertices interactively. When the user changes a parameter, the geometry and parameters are updated recursively. Rendering requires a recursive traversal of all scene graph nodes.

5.1 Selection in the viewport

Clicking small branches that occupy only few pixels in the viewport may be difficult for the user, because the mouse driver does not notify the system of every pixel the cursor runs through. Moreover, the user may not want to zoom onto a single branch, because he wants to view changes to a number of branches. In these cases, it is easier for the user to move the cursor to cross the branch instead of clicking it. This requires building a polygon from the last and current mouse positions and intersecting that polygon with the geometry.

5.2 Viewport tools

Whenever any branch's control points are moved, its child branches and the following control points on the same branch must be moved as well. If the branch's first control point is moved, its anchor value is recalculated. The child branches' anchor values a_j allow to decide quickly which branches need to be moved when control point p_i is moved. These are the branches b_j with anchor value a_j where $a_j n > i - 1$ for a branch with n child branches. All selected branches' control points are moved by an equal distance, and their "Length" parameter is recalculated.

For sketching, the start point for the new branch is selected with the same algorithm as for viewport selection. After that, all mouse positions are stored in an array and branch control points are selected equally spaced from the array in every frame. This allows to render the new branch interactively during sketching.

5.3 Geometry Creation and Rendering

We use the 4-point scheme [6] to produce smooth branch curves from the control points. It guarantees that the branch curve contains the control points. Then we compute generalized cylinders for the branches from the vertices and radii. The polygons are stored directly in the scene graph node.

Leaf polygons are managed and stored in their parent branch. Leaf texels are interpreted as fully transparent or completely opaque. This allows us to use alpha-testing rather than the more expensive depth sorting for rendering the leaves.

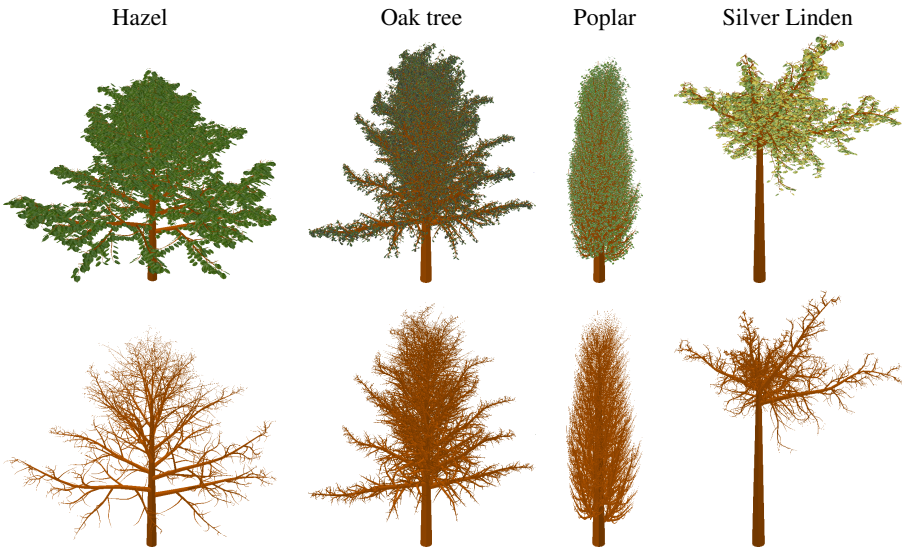

Figure 3: Example trees

Species	Modeling time	#Branches	#Leaves	#Branch levels	Geometry	Parameters
Hazel	4 min 58 s	2,221	17,939	4	21.9 MB	1.1 MB
Oak	5 min 03 s	9,931	57,228	4	22.3 MB	4.7 MB
Silver Linden	7 min 33 s	2,211	7,048	4	21.7 MB	1.1 MB
Poplar	3 min 29 s	6,631	27,483	4	32.4 MB	3.1 MB
Ash Tree	4 min 37 s	221	680	4	689.0 kB	106.5 kB
Beech Tree	5 min 57 s	7,517	187,000	5	23.8 MB	3.5 MB
Ahorn	ca 25min	34,001	393,822	5	75.5 MB	15.7 MB
Birch Tree	8 min 37 s	23,011	99,190	5	114.0 MB	10.9 MB
Plane Tree	ca. 9min	14,373	204,138	5	18.8 MB	7.0 MB

Table 1: Statistics for creating the trees in Figures 1, 3, 4

5.4 Persistence

In procedural modeling, the persistence problem refers to the difficulty of retaining user edits after parts of the model were changed [4, 14]. The naive approach to change the number of branches on a parent branch uses one of the old branches as a template for creating the new branches. However, if one of the deleted branches was edited with individual parameters, these parameters are lost. Therefore, instead of deleting branches, the existing branches should be moved to their positions, and then their positional information should be re-evaluated. Assigning tags further reduces the impact of changing the number of branches.

6 Results

1-2-tree recreates geometry only for scene graph nodes that were changed, and these changes are applied fast enough that parallel execution is not needed. Table 1 lists memory usage and time needed to model the trees shown in this paper. The amount of memory needed to store the geometry usually exceeds the memory needed for the parameters by a factor of at least 2. Given today's memory sizes, we think the added cost of individual parameters for each branch is affordable for editing single trees.

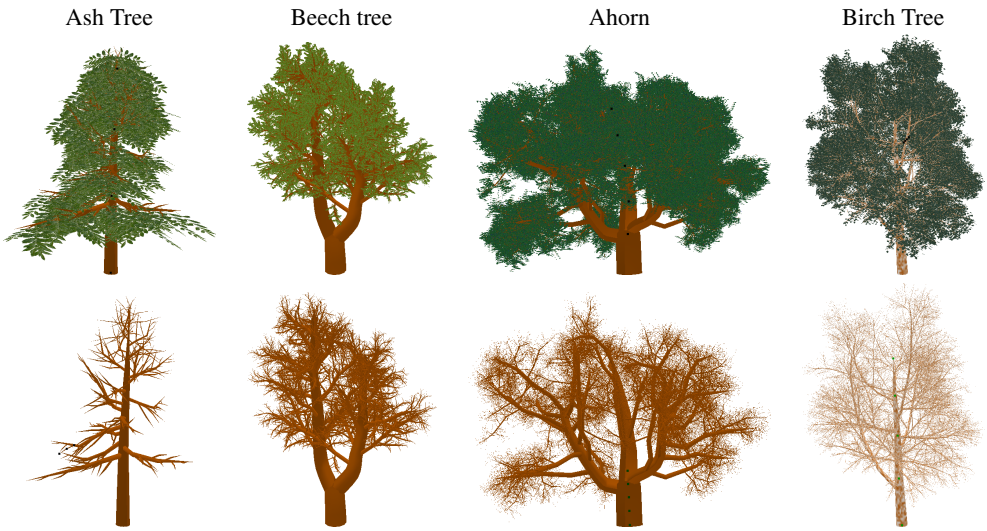

Figure 4: More examples. For the ash tree, the final level of branches was modeled as a texture.

Each branch takes 440 bytes of fixed storage and has a number of external string buffers used to store positional information. While memory use for parameters varies only slightly, memory cost for geometry depends on the level of detail generated. We chose a level of detail appropriate for display.

6.1 Discussion

We compare 1-2-tree to other tree modeling systems. L-systems are a long success story in modeling plants, but we preferred to operate directly on a scene graph for a number of reasons. First, the works of Weber and Penn, Lintermann and Deussen demonstrate that most trees do not require the flexibility offered by L-strings. Secondly, 1-2-tree's editing functions build on hierarchy traversal, which can be implemented more efficiently on hierarchical data structures than on an L-string. Thirdly, while any alteration requires completely parsing and copying the L-string, updates are very efficient with a hierarchical data structure because only the geometry of the affected scene graph nodes needs to be updated.

While Weber and Penn's system stores parameters for every branch level, 1-2-tree stores similar parameters for every single branch, and 1-2-tree can be configured to apply changes to all branches on a single level. We frequently use this selection mode. Whereas Weber and Penn's system allows varying parameters only randomly, our notation for positional information allows mixing random influences with parameter increases or decreases along the branch. Therefore, 1-2-tree is a superset of Weber and Penn's system.

Lintermann and Deussen's visual system, called Xfrog, provides node types with numerous parameters. The variety of plants that can be modeled with Xfrog may be comparable to the power of L-systems. Xfrog stores the rule system describing a plant in the so-called p-graph. Xfrog's algorithm for geometry creation takes the p-graph and a separate exception list as input to create the model's geometry. These three data structures are integrated into a single data structure in 1-2-tree, which reduces computation time and algorithmic complexity. While Xfrog and L-systems are more versatile systems for modeling plants, 1-2-tree specializes in modeling trees rapidly and individually. In contrast to Xfrog, users of 1-2-tree do not require knowledge about loops and mathematical formulae. 1-2-tree's selection modes and viewport editing allow for more intuitive modeling, because 1-2-tree relieves the user of finding the p-graph node that produces a certain effect.

Like 1-2-tree, Boudon et al's system stores individual parameters for each branch. In 1-2-tree, copying changes to other nodes is governed by intuitive selection settings rather than inheritance settings. 1-2-tree integrates the functionality of Boudon et al's decomposition graph and the branching structure browser into the 3D viewport to further ease modeling.

The system by Okabe et al [17] is orthogonal to ours. It focuses on sketch- and example-based modelling, whereas 1-2-tree focuses on procedural modeling and direct editing methods. Both approaches have their advantages, and future users might want to have a system that integrates the strengths of both systems.

Boudon et al report an average modeling time of 3 hours for each model [4], and Okabe et al's models took less than 10 minutes on average [17]. In 1-2-tree most models take between 4 and 10 minutes to create.

7 Conclusion

We tried to balance the various approaches and use sketching where it is strongest, which is modeling individual branches. We feel that deriving parameters from viewport input and applying the derived parameters to other instances nicely combines procedural modeling and sketch-based interfaces. We focused on creating a system that gives the user maximum control over the process of modeling trees without requiring him to learn concepts like loops, grammars and mathematical formulae. This reduces the flexibility of 1-2-tree, but greatly enhances the ease of use. Simple tree models can usually be created within a few minutes, and arbitrary precision can be obtained by modeling individual branches.

We have demonstrated the benefits of storing branches as semantic entities in a hierarchical data structure for modeling trees. Among these benefits are the powerful selection options for changing either single entities or masses of branches. Moreover, we have shown that this data structure can be used to propagate viewport changes back to the parameters.

7.1 Future Work

The system proposed here could be extended to edit trees that were reconstructed from images and to determine parameter distributions of real trees. The system would take a number of images as input, compute tree skeletons, conclude parameter distributions and finally use these parameters to create other trees of the same species. The same algorithm should compute branch textures from the input images.

1-2-tree could be improved by more options and better algorithms for automatic radius calculation, branch and leaf placement.

7.2 Acknowledgments

We would like to thank Martin Schneider, Roland Wahl, Lisa Schützendorf, and Klaus Greulich for discussions and advice.

A Parameters

We use a simplified notation to mimic the effects of positional information. The notation a-b increases or decreases the value along the parent branch from a to b. In order to introduce random variations, a parameter can be stated as cud, meaning a uniform distribution with an expected value of c and a maximum deviation of d. Both notations can be mixed: aub-cud. This notation allows to interpolate between randomly distributed values, therefore we call it an interpolated distribution for short.

Parameters are stored individually for each branch, and are sorted into categories. Category "Branch Parameters" defines basic branch properties, such as the branch's length and its number of control points. Further parameters include "Gnarl": maximum random deviation from a straight branch, "Gravitropism": a gravitational effect bending a branch down or up, "Subdivision Steps": number of subdivision steps used to produce smooth branches, "Complexity": number of polygons to create for each generalized cylinder, "CPS": allows assigning the currently selected branches to a different child parameter set. This is useful to protect branches from changes.

The category “Radius Parameters” contains parameters for editing the branch radius and buttons for calculating radii automatically. There are parameters for the radius at the start (“Radius”) and at its end (“Min. Radius”). The button “Calculate Radius” calculates radii from “Min. Radius” using da Vinci’s law for branch radii [8].

The category “Child branches” governs child branches. Its parameters are the “Number of Branches”, “Anchor values”. It is possible to configure these parameters for several separate child parameter sets (CPS).

The category “Angles” pools parameters that model a branch’s angle against sibling and parent branch. The category “Leaf” parameters contains similar parameters for leaves to the parameters explained before.

Category “Bark” allows creating a simple noise texture to use as bark. Small green branches can be used to produce the needles for a conifer. In that case, “Control Point Count”, “Complexity” and “Subdivision Steps” should be set to minimum values to reduce the number of polygons created.

References

[1] Fabricio Anastacio, Mario Costa Sousa, Faramarz Samavati, and Joaquim A. Jorge. Modeling plant structures using concept sketches. In *NPAR ’06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 105–113, 2006.

[2] M. Aono and T. Kunii. Botanical tree image generation. *Computer Graphics and Applications, IEEE*, 4(5):10–34, May 1984.

[3] Jules Bloomenthal. A representation for botanical trees using density distributions. In *1st Int’l Conf. on Engineering and Computer Graphics*, 1984.

[4] Frederic Boudon, Przemyslaw Prusinkiewicz, Pavol Federl, Christophe Godin, and Radoslaw Karwowski. Interactive design of bonsai tree models. In *Proceedings of Eurographics 2003: Computer Graphics Forum 22 (3)*, pages 591–599, 2003.

[5] Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger, and Claude Puech. Plant models faithful to botanical structure and development. In *SIGGRAPH ’88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 151–158, 1988.

[6] N. Dyn, J. Gregory, and D. Levin. A 4-point interpolatory scheme for curve design. In *CAGD 4 (1987)*, pages 257–268, 1987.

[7] N. Greene. Voxel space automata: modeling with stochastic growth processes in voxel space. In *SIGGRAPH ’89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 175–184, 1989.

[8] M. Holton. Strands, gravity and botanical tree imagery. *Computer Graphics Forum 13 (1)*, pages 57–67, 1994.

[9] Hisao Honda. Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. *Journal of Theoretical Biology*, pages 331–338, 1971.

[10] Takashi Ijiri, Shigeru Owada, and Takeo Igarashi. The sketch L-system: Global control of tree modeling using free-form strokes. In *6th International Symposium on Smart Graphics 2006, LNCS 4073*, pages 138–146. Springer Verlag, 2006.

[11] Takashi Ijiri, Shigeru Owada, Makoto Okabe, and Takeo Igarashi. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *SIGGRAPH ’05: ACM SIGGRAPH 2005 Papers*, pages 720–726, 2005.

[12] Aristid Lindenmayer. Mathematical models for cellular interactions in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[13] Bernd Lintermann and Oliver Deussen. A modelling method and user interface for creating plants. *Computer Graphics Forum*, 17(1):73–82, 1998.

[14] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. In *Proceedings of ACM SIGGRAPH 2008*, 2008.

[15] Radomír Měch and Przemyslaw

Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 397–410, New York, 1996. ACM.

[16] Boris Neubert, Thomas Franken, and Oliver Deussen. Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics*, 2007.

[17] Makoto Okabe and Takeo Igarashi. 3d modeling of trees from freehand sketches. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1, 2003.

[18] Makoto Okabe, Shigeru Owada, and Takeo Igarashi. Interactive design of botanical trees using freehand sketches and example-based editing. In *Proceedings of Eurographics 2005: Computer Graphics Forum 24 (3)*, pages 487–496, 2005.

[19] Peter E. Oppenheimer. Real time design and animation of fractal plants and trees. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, pages 55–64, 1986.

[20] Joanna L. Power, A. J. Bernheim Brush, Przemyslaw Prusinkiewicz, and David H. Salesin. Interactive arrangement of botanical L-system models. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 175–182, 1999.

[21] P. Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990.

[22] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, 1994. ACM.

[23] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 289–300, New York, 2001. ACM.

[24] Long Quan, Ping Tan, Gang Zeng, Lu Yuan, Jingdong Wang, and Sing Bing Kang. Image-based plant modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 599–604, 2006.

[25] Alex Reche-Martinez, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 720–727, 2004.

[26] William T. Reeves and Ricki 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*, pages 313–322, 1985.

[27] Tatsumi Sakaguchi and Jun Ohya. Modeling and animation of botanical trees for interactive virtual environments. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–146, 1999.

[28] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61, 2001.

[29] Alvy Ray Smith. Plants, fractals, and formal languages. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, pages 1–10, New York, 1984. ACM Press.

[30] Paul S. Strauss and Rikk Carey. An object-oriented 3D graphics toolkit. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, pages 341–349, New York, 1992. ACM Press.

[31] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, 1995.