Synthetic Tree Models from Iterated Discrete Graphs

Ling Xu* David Mould†

Carleton University, Canada

ABSTRACT

We present a method to generate models for trees in which we first create a weighted graph, then places endpoints and root point and plan least-cost paths from endpoints to the root point. The collection of resulting paths form a branching structure. We create a hierarchical tree structure by placing subgraphs around each endpoint and beginning again through some number of iterations. Powerful control over the global shape of the resulting tree is exerted by the shape of the initial graph, allowing users to create desired variations; more subtle variations can be accomplished by modifying parameters of the graph and subgraph creation processes and by changing the endpoint distribution mechanisms. The method is capable of matching a desired target structure with a little manual effort, and can easily generate a large group of slightly different models under the same parameter settings. The final trees are both intricate and convincingly realistic in appearance.

Keywords: Procedural modeling, tree modeling, natural phenomena, geometry synthesis.

Index Terms: I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling —Curve, surface, solid, and object representations

1 INTRODUCTION

Trees are commonplace in the natural world, and tree models often appear in the virtual worlds of computer games and films. To ease the burden on digital artists, procedural modeling methods have been devised, especially for complicated subjects such as trees. The key elements of tree modeling are the branching structures and the complex shape of individual branches.

This paper presents a procedural method to model trees, based on finding least-cost paths through a weighted graph, a modeling idea previously introduced by Xu and Mould [20, 21]. The essential idea is to create a graph with random edge weights, then plan least-cost paths from a single root node to destination nodes. The resulting paths form a tree. By varying the graph shape and edge weights, the method can create a wide range of tree models.

This paper builds on the basic idea to create sequences of graphs, using path planning to link endpoints in all graphs with a single root node. Our method can create realistic, highly intricate tree models with quite direct user control over the final tree shape through specifying the shapes of the graphs in which the tree is built.

The paper is organized as follows. Following the introduction, we review some previous work in tree modeling. In section 3, we describe the algorithm. Results and evaluation are given in section 4. Finally, we conclude and discuss future work.

2 PREVIOUS WORK

Tree modeling has a long history in computer graphics. The most notable modeling approach is the parallel rewriting grammar called L-systems, used for plant forms and even entire ecosystems [12, 7, 4]. General control over grammar-based methods is offered by Talton et al.'s work [15], although the sampling process can be very time-consuming. The space colonization method of Runions et al. [14] offers a biologically motivated alternative with control over global shape, exploited by Palubicki et al. [10] for self-organizing tree modeling; here, the tree growth process follows the competition of branches for resources (e.g., light and space) with internal signaling mechanisms based on L-systems. The resulting forms can be controlled interactively, e.g., by sketching.

Geometric methods (e.g., that of Weber et al. [17]) explicitly vary geometric quantities such as segment length and angles; generating models involves adjusting a huge number of parameters. An alternative is to use input images to drive tree creation [13, 8, 16]: such methods can attain extremely high quality, although the need to supply input images is a drawback.

The basic idea of path planning [3] for tree modeling is due to Xu and Mould [20, 21], who exploited path planning for general modeling of dendritic natural phenomena including trees, coral, and lightning. In their work, the graph containing the paths is a 2D lattice or 3D grid. The regular lattice imposes substantial penalties: the resolution of the model is limited by the spacing of the graph, and hence small-scale features (e.g., tiny twigs) need a high-resolution graph, incurring immense memory cost.

To enhance the control in tree modeling, sketch-based methods are used to provide clues of crown shape or main branches [5, 9, 2, 18]. Based on L-systems, Ijiri et al.'s system [5] controls the growth direction of a tree by user-drawn strokes. However, to model a complex tree would require a lot of user interaction. Okabe et al. [9] build tree models using freehand sketches with the assumption that branches are spreading to maximize the distance between each other. In Chen et al.'s method [2], Markov random fields are used to infer the branch shape from the drawn sketches. Both methods use examples from a library of tree templates for branch propagation, which release the burden on user sketching. With similar stochastic optimization, Wither et al. [18] use a priori botanical knowledge to infer the branch shapes from user sketched crown silhouettes at different scales, and can generate realistic tree models with good overall controls.

Compared to the above methods, scanning methods place more emphasis on creating models that conform to real trees, with point clouds of tree data by 3D scanning. Xu et al. [19] build a tree skeleton by connecting neighborhood points to form a graph, where a single-source shortest path algorithm is applied to reconstruct branches. Bucksch et al. [1] extract a the tree skeleton by subdivisiding the point cloud. Livny et al. [6] apply global optimizations to reconstruct multiple overlapping trees simultaneously. Scanning methods can achieve high quality of tree models, but are not intended to model novel trees.

3 ALGORITHM

We build on the method of Xu and Mould [20], who created least-cost paths through a regular lattice connecting multiple endpoints to a common root in order to build general tree-like structures. Since they used a regular lattice, they little investigated the task of building graphs; a significant portion of this paper is devoted to defining the graph shapes, which have an enormous impact on the shape of

*e-mail: lxuc@scs.carleton.ca

†e-mail: mould@scs.carleton.ca

the final tree. The earlier work also did not pay much attention to the details of endpoint placement. We propose an iterative method whereby successive stages of endpoints are distributed within subgraphs, resulting in a high degree of visible structure; we discuss the details of the method next, to be followed by examples of our synthetic tree images.

3.1 Basic Algorithm

We construct a graph and find the shortest paths from multiple endpoints to a common root point. The collection of the paths form the tree model. The basic algorithm can be decomposed into the following steps.

1. Build a graph and set edge weights randomly.
2. Choose a node to be the root point and some nodes as endpoints of the structure.
3. Find least-cost paths from the endpoints to the root.
4. Create geometry around the path segments and render the resulting model.

Xu and Mould used a graph consisting of a regular lattice, but the resulting paths suffered from lattice artifacts. We propose instead an irregular graph, obtained by creating a Poisson disc distribution of nodes within a designated volume; nodes are connected by edges whenever they are sufficiently close together, and a random weight is assigned to each edge. Figure 1 contrasts regular and irregular graphs: use of an irregular graph avoids lattice artifacts without necessitating higher resolution.

Using a single graph produces non-hierarchical structures, somewhat resembling trees but overly simple. In the next section, we describe how we create a succession of graphs attached to the first, whereby we can create more elaborate and convincing trees.

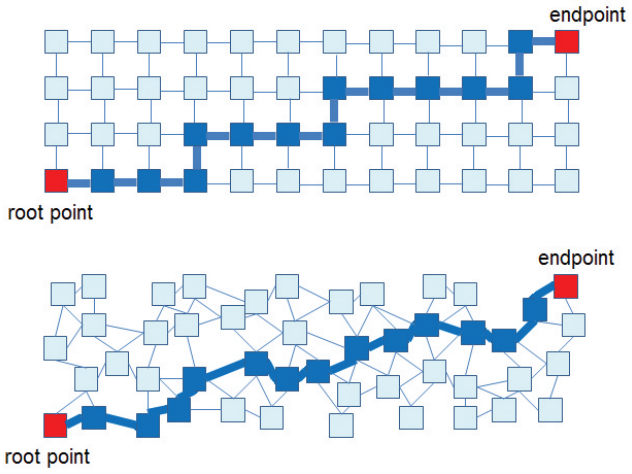

Figure 1: A regular and an irregular graph.

3.2 Iterated Structure Building

A tree has a recursive structure where large branches grow from the main trunk and smaller twigs develop from branches. Our algorithm creates the hierarchy explicitly: iteratively extending the original graph by adding subgraphs at the tips of earlier branches.

The overall process operates as follows. A base graph shape is defined by a composition of geometric primitives, with the graph itself constructed by distributing nodes throughout the volume and connecting nearby nodes with edges. Endpoints are then placed and a preliminary tree model is created by connecting the endpoints with the root using least-cost paths. Subsequently, for level $i > 1$, we create a subgraph around each endpoint from level $i-1$, again defining a volume and distributing nodes and endpoints within it. (Details of the subgraph creation are given in section 3.3.2.) The endpoints are connected to the structure obtained in level $i-1$, producing a new structure. The process repeats for some number of levels, say 4; depending on the desired complexity of the final structure, the number could be higher.

Maketree
Global parameters:
subgraph resolution k (number of nodes)
node linking distance d
subgraph shrinking parameter a ($a < 1$)
subgraph sizing angle α
Note, a node N has property $\vec{x}$, spatial position.

Arguments:
V (graph volume),
R (root node),
b (branching factor),
K (number of nodes in graph),
r (size),
L (remaining lifespan)

Output:
T, a list of all path segments making up the tree.
1. Create the graph for the current volume:
1A. $G \leftarrow \emptyset$.
1B. Find a random location $\vec{p}$.
1C. If $\vec{p}$ is outside V, reject $\vec{p}$.
1D. For all nodes $N \in G$, if $|\vec{p} - N.\vec{x}| < d$ reject $\vec{p}$.
1E. If $\vec{p}$ not rejected, create node m with $m.\vec{x} = \vec{p}$ and set $G \leftarrow \{G, m\}$.
1F. Repeat 1B to 1E while $|G| < K$.
1G. For all pairs of nodes $n, m \in G$, create a connecting edge if $|n.\vec{x} - m.\vec{x}| < d$.
1H. For all edges, set random weights.
2. Repeat for b endpoints e:
2A. Set $e.\vec{x} \leftarrow$ random position in V.
2B. For all nodes $n \in G$, create an edge from n to e if $|e.\vec{x} - n.\vec{x}| < d$.
3. Create paths for all endpoints e; before starting, initialize $T \leftarrow \emptyset$.
3A. Find the least-cost sequence of edges P from e to R.
3B. For all edges $E \in P \setminus T, T \leftarrow \{T, E\}$.
4. Recurse on all endpoints:
4A. For each endpoint e:
4B. $\vec{v_g} = (e.\vec{x} - R.\vec{x})/|e.\vec{x} - R.\vec{x}|$.
4C. define V as the portion of the sphere centred at $e.\vec{x}$ with radius r that lies within angle α of $\vec{v_g}$
4D. If $L > 0$, $T \leftarrow \{T,$ maketree$(V, e, b, k, a * r, L-1)\}$
5. Return T.

Figure 2: Pseudocode for tree construction.

Pseudocode describing the building process is given in Figure 2. While above we described the process in an iterative fashion, and our implementation is likewise iterative, we found it more convenient to present pseudocode for a recursive implementation, echoing the visual recursion of the the final structure of the tree.

The preceding gives the process to construct the schematic of the model; we then interpret the paths as geometry, placing a cylinder around each edge in the structure. The thickness w of each cylinder at level i is decided by the distance from the segment to the branch tip, written d': $w = (d')^{\zeta}/(i+1)$; larger values of ζ make the branches taper more quickly. Typically we use $\zeta = 0.3$. We can

render the structure in a schematic fashion by directly drawing the cylinders as black regions on a white background (used in numerous visualizations throughout the paper); we can also render the geometry photorealistically, and employed POV-Ray [11] for that purpose in this paper.

Figure 3 illustrates the method. The initial graph is a composition of a hemisphere and a cylinder. Endpoints are randomly positioned in the hemisphere, and paths are planned from the root to the endpoints. Then, a subgraph is created for each endpoint; we illustrate an example subgraph in the lower left. The lower right figure shows the structure once the second level has been completed.

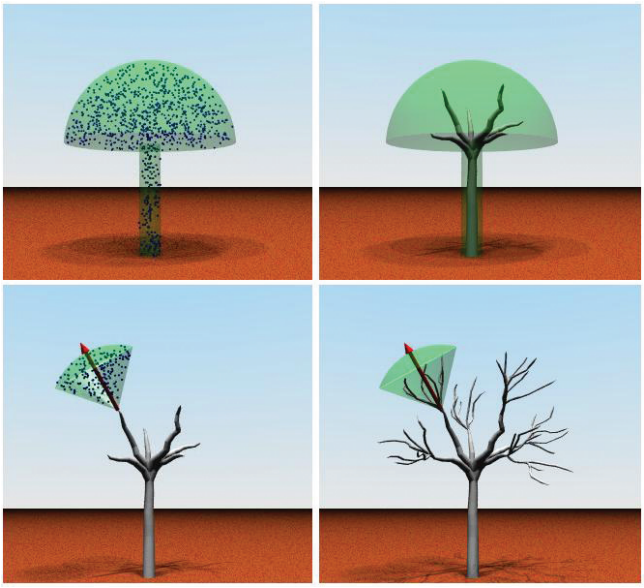

Figure 3: Illustration of the iterative tree building process.

To capture the hierarchical structure of real trees, we use the concept of *lifespan*. Each endpoint is assigned a lifespan value L; endpoints in a subgraph will have a lifespan strictly smaller than the parent endpoint's lifespan, usually by taking $L_{i+1} = L_i - 1$. No subgraph is created if an endpoint's lifespan reaches zero. The subgraph shape and size can depend on the lifespan of the subgraph root.

We used our method to generate the structures shown in Figures 4 and 5. Figure 4 shows an elaborate branching structure obtained by starting with six endpoints in a sphere and continuing for four levels. The resulting form is somewhat abstract, but its intricacy is compelling. By imposing more structure on the initial level, we can create structures more closely resembling trees, shown in Figure 5; here, the initial shape is a mushroom-shaped cylinder plus hemisphere, reflected in the overall shape of the final tree. The top view reveals the desired horizontal anisotropy of the tree, while the close view allows better appreciation of the detailed small-scale structure.

3.3 Variations from parameters

The three main mechanisms to modify the shapes of the synthetic trees are initial graph shape, subgraph shape, and lifespan. The initial graph shape has a profound effect on the overall shape of the tree. Logic governing subgraph shape controls how the tree develops at levels beyond the first, and affects the general appearance of the tree in a more subtle way. Finally, lifespan can introduce additional variety by altering individual branch development. We discuss each of these in more detail below.

Figure 4: A fractal dendrite in 3D.

3.3.1 Graph shape

The graph shape in the first level controls the overall shape of the resulting model. We describe two methods to obtain graph shape: combination of geometric primitives and user sketching. In the former, primitives such as cylinders, spheres, and ellipsoids are manually arranged into an approximation of the desired shape. In the latter case, we use a user sketch to infer a volume in which we distribute nodes. Many sketch-based modeling possibilities exist, which we have little explored in the present work; we demonstrate the feasibility by showing trees derived from the volume enclosed by the surface of revolution of a user-sketched stroke.

Figure 6 shows three different trees along with their corresponding graph shapes. The final model does not strictly match the original graph shape owing to the structures added in levels beyond the first, but the correspondence is clear. More specific results are also possible: Figure 7 shows an example of modeling an irregular tree, imitating a photograph. The graph was composed and endpoints selected manually to match the desired outcome.

Sketch-based modeling provides more flexible and direct user control than assembling geometric primitives. Here, we provide a glimpse of how sketching can be used with our method, allowing users to indicate a volume of revolution. A user draws a curve with reference to an axis, and the sketched curve will be rotated around the axis to achieve a 3D volume of revolution. Then the graph nodes are placed in the enclosed volume and connected with edges. Figure 8 shows some examples of results obtained from sketching interactions. We can see the ease with which more complex volumes can be defined; as before, the shape of the initial graph is the most significant contributor to the global shape of the final tree model.

3.3.2 Subgraph creation

We require users to specify the graph shape for the first level, providing control over the tree's large-scale appearance. While subgraph shapes for subsequent levels can in principle also be user-defined, in practice it is tedious to do so, so we compute the subgraph volumes procedurally, as follows.

We use a cone-shaped subset of the sphere as our subgraph, where the cone's tip is placed at the root of the graph. First, we compute an orientation $\vec{v}_g$ for the subgraph by taking the normalized vector from the root of this subgraph to the root of the previous subgraph. The volume is defined as all points whose vector from the root lie within an angle α of the vector $\vec{v}_g$. The volume is populated with nodes in the same way as the initial graph, and the subgraph is formed by linking nodes closer together than a minimum distance d.

The size of the subgraph sphere deceases as we progress to higher levels: the parameter a, where usually $a < 1$, is the ratio between the sizes of spheres at two successive iterations. Figure 9 shows two trees obtained with different a. The left tree, with $a = 0.7$, demonstrates a clear hierarchical relationship in branch

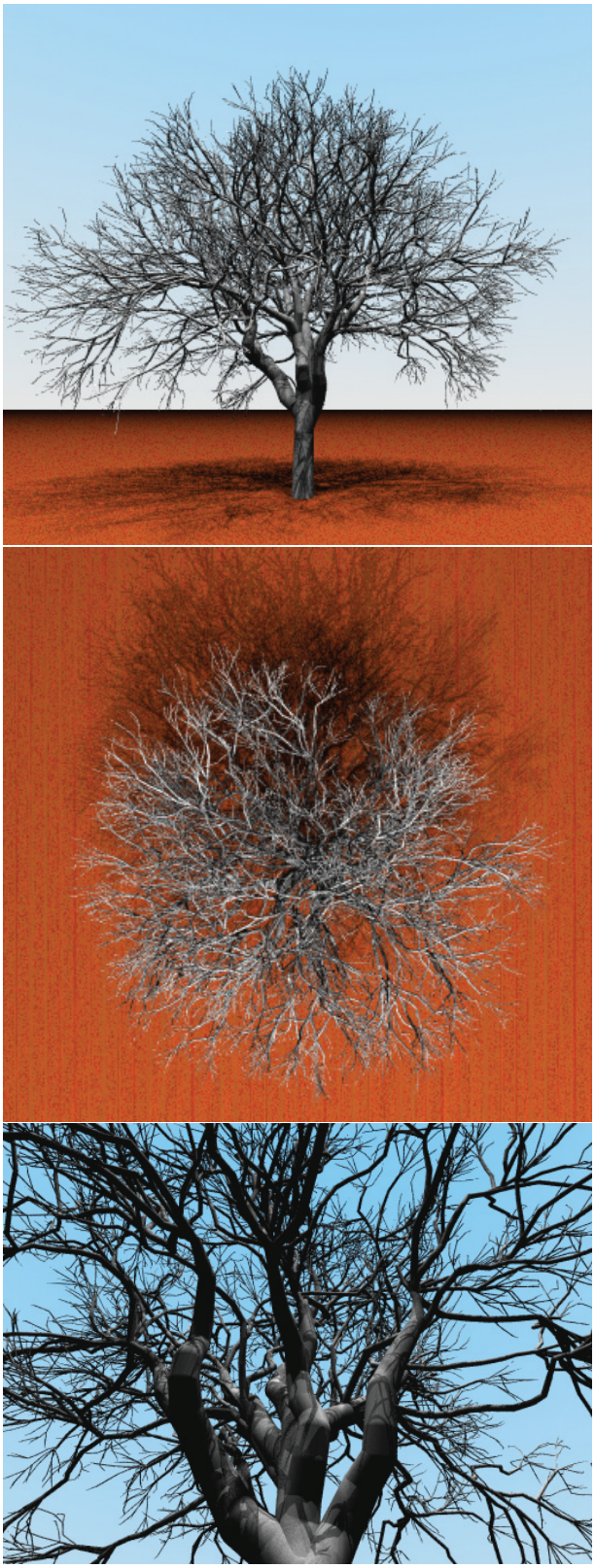

Figure 5: A synthetic tree created with four iterations.

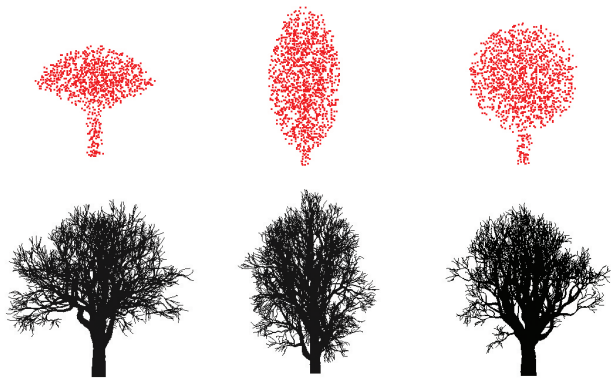

Figure 6: Structures obtained by different shapes of graph.

Figure 7: A tree with a tilted trunk. Left: initial graph; middle: inspirational photograph; right: our model.

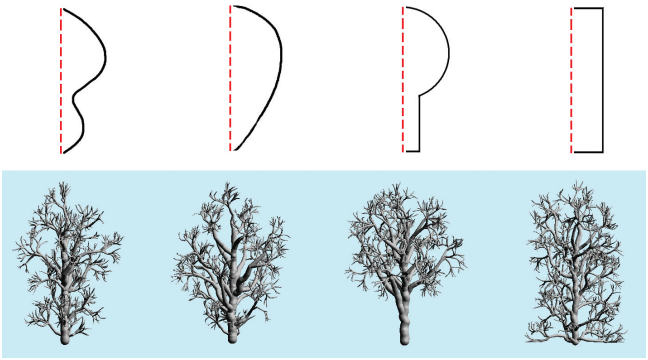

Figure 8: Structures obtained by user sketches. Above: user sketched curves with rotation axes (dashed lines); below: resulting structures.

lengths. The branch segments closer to the trunk are long and those near the tips are short. The tree in the right has a constant subgraph size (with $a = 1$) at each level.

To produce paths with long-term curvature (similar to willow branches, for example), we proceed segment by segment. We have previously described the subgraph orientation $\vec{v}_g$, obtained by finding the vector from the subgraph root to the preceding root; now, we apply a consistent transformation to $\vec{v}_g$ at each level. When the transformation is a rotation about the horizontal, the overall branch curves upward or downward. Figure 10 shows an example of structures obtained by the above method, with four iterations applied.

Figure 9: Right: tree with $a = 0.7$; left: tree with $a = 1$.

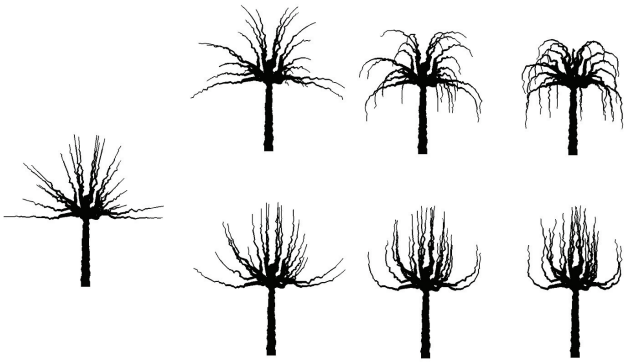

Figure 10: Curving branches from progressively rotating subgraphs.

Note that this particular procedural approach to subgraph shape is not the only possibility, although it is a convenient option that we rely heavily on in this paper. Other possibilities include the following: different subgraph shapes, e.g., inverted cones; different mechanisms for computing the orientation, e.g., using random directions or a fixed vertical orientation; or adjusting the direction, shape, or size of the subgraph based on environmental information. We will show examples of some of the possibilities in section 4.

3.3.3 Lifespan

Previously we had lifespan dropping at a constant rate, i.e., $L_{i+1} = L_i - 1$. However, this produces very uniform trees, where all branches are approximately the same length. If we allow the lifespan parameter to vary more widely, we can produce more irregular trees. One possibility is to use a distribution of possible decrements: for example, we can assign a 30% probability of terminating a branch and a 70% chance of instead decrementing its lifespan by a random number in the range $(0, L)$, where L is its current lifespan.

Figure 11 shows some examples obtained by using the above distribution for lifespan decrement. The resulting trees have a striking irregularity and seem more lively and natural than the trees generated with fixed lifespan decrement. However, the approach does not reliably generate models of this caliber. Further investigation of lifespan is a direction of future work.

Figure 11: Irregular bush and tree obtained by use of lifespan.

4 Results and Evaluation

The elements of our tree modeling algorithm, including edge weights, graph shape, and node placement, provide a wide range of results. This section shows a cross-section of results and provides comparisons to real photographs and to previous methods. In Figure 12, synthetic tree images are compared with photographs. Our trees have similar structures to the photographed trees: the same tree crown shape, a few thick main branches, and a large volume filled with twigs yet with natural-seeming irregularities and gaps. Overall, it is difficult to distinguish between the real and synthetic trees from these images, and we judge that our method is quite effective.

Figure 12: Left: our black silhouette tree model; right: real photograph.

Due to the involvement of random elements – particularly random edge weights and random endpoint placement – we can generate similar but distinct trees by keeping parameter settings fixed. Figure 13 shows three trees of the same type. In each case, the initial graph is composed of a cylinder and a hemisphere, but each tree

has a slightly different structure while keeping large-scale characteristics in common, such as the crown size and the branch density.

Figure 13: Three trees of the same type.

By varying graph shape and available parameters, we can create a wide variety of trees; examples are shown in Figure 14. We chiefly varied initial graph shape to generate these examples; some of them also used custom graph shapes for the subgraphs. A list of parameter settings for these trees can be found at the end of this paper.

In addition to the wide variety of trees shown, our algorithm can be used to model the root system. Figure 15 shows two trees with different shapes of root systems; the roots were created in a graph bounded by a hemisphere. The taproot is a path from one single endpoint to the root point. The lateral roots of both trees are obtained by placing endpoints randomly around the taproots. This is the same method used to create the structure shown in Figure 4.

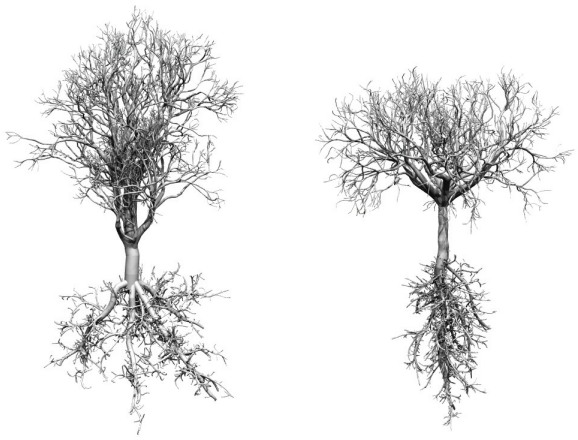

Figure 15: Two trees with different root systems

Compared to existing tree modeling methods, our method has its strengths. One key element we provide is the ability to model structures with irregular branches. In the case of geometric based methods, the more irregular the object is, the more parameters are needed. However, in our method, the irregular paths are the byproduct of path planning through a randomly weighted graph. Without extra effort, our method generates irregular yet globally controllable structures.

Specific comparisons to previous methods are shown in Figures 16, 17, and 18. Based on these comparisons, our method is competitive. To our knowledge, robust metrics for tree quality do not exist, so we discuss the visual comparison in a general way below.

Figure 16 shows our tree model compared with a tree model generated by Xu and Mould [21]. Compared to their model, our result is much more detailed, with many more branches and with branches of different sizes. The use of iterated graphs allowed us to create small features without an explosion in overall memory usage. Figure 17 compares our tree model to an example by Neubert et al. [8] created using a particle tracing method. Both have realistic visual effect. However, in general, particle tracing methods have difficulty enforcing large-scale coordinated movement of the particles so that the desired shape is formed; in this case, and that of Tan et al., extra information in the form of input images provides the needed large-scale coordination. Our method does not require real photos and hence allows the user to skip that step, potentially providing more control over the output tree shape. Figure 18 shows our tree model and a tree model by the self-organizing method of Palubicki et al. [10]. Our result has a similar global shape and branching structure as their model. However, we characterize it as less regular: its branches are more crooked and in the projection to 2D the gaps are more unevenly distributed. Whether this is an advantage or not depends on the application; users might sometimes prefer the irregularity in pursuit of certain effects (for example, in creating a haunted forest).

Our method has some limitations as well. While we are free from lattice artifacts and hence can create convincing tree models with lower graph resolution, the feature resolution is still tied to the node spacing and hence the approach is fairly memory-intensive. Detailed control over tree shape is provided by endpoint placement, but we care about the path rather than the tip position. Finally, while we have argued that the irregularity of the resulting models is a strength of our approach, the models are still overall more regular than we would prefer; our experiments with lifespan, though promising, are still in their infancy. Increasing the irregularity of the output models is an important direction for us. We close this section by providing parameter and timing information for some of our models. Timing information and statistics for selected tree models appears in Table 1. Smaller trees could be completed in a few seconds; our largest tree, with about 30k endpoints in graphs of over 300k nodes, required about 30 seconds.

In all cases, timing results are with respect to a 3.0 GHz CPU with 3.0 GB RAM. In general, the time required is linear in the total number of nodes in all graphs combined, given a suitable spatial subdivision scheme for proximity queries in graph construction. The parameter information for the trees in Figure 14 is given in Table 2.

Figure 16: Left: a model created by Xu and Mould; right: our tree model.

Tree	number of graph nodes	number of endpoints	timing
Figure 12(top left)	351810	34740	32.4s
Figure 16	84755	3732	4.5s
Figure 18	207458	9330	20.0s

Table 1: Timing and construction data for selected models.

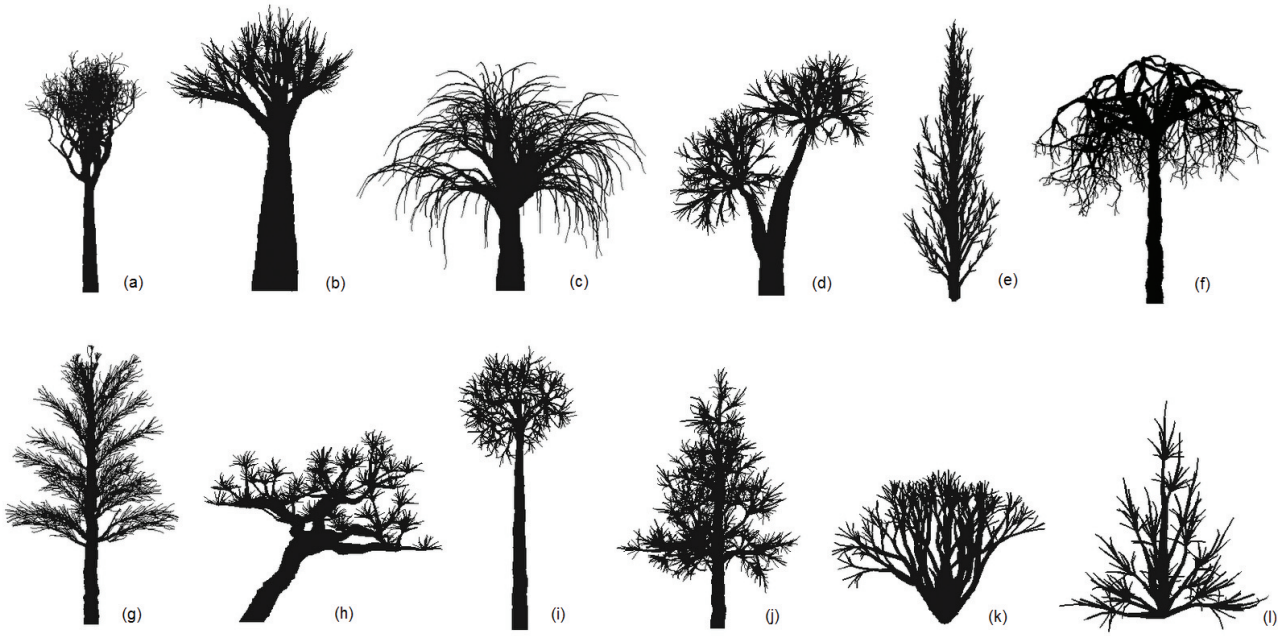

Figure 14: Different types of trees.

Figure 17: Left: a model from Neubert et al.; right: our tree model.

Figure 18: Left: a self-organizing tree model; right: our tree model.

5 Conclusions and Future Work

We demonstrated that procedural tree modeling based on path planning is capable of producing elaborate and realistic trees. The general shape is decided mainly by the initial graph shape and partially by the shapes of subgraphs added in later stages. Wide variations are possible, producing many different shapes of trees, some of which are chronicled in this paper. Since the algorithm involves random edge weight and node and endpoint placement, many different but similar models can be constructed from the same parameter settings. In most applications general shape control is considered important, so we provide large-scale control through specifying graph shape. Optionally, the finer structure can be guided by specifying graph shapes to use in later iterations; we provide defaults which provide generically appealing results.

We have two main directions for future work: increasing the automatic level of irregularity of our trees, and increasing the degree of user control over the output. Our concept of branch lifespan seems like a promising approach for increasing irregularity; we can explore different distributions of possible lifespans, perhaps in a coordinated way across multiple endpoints. Further exploration of endpoint distributions is another possible direction. For user control, we believe that sketching can feasibly be combined with the current approach; users might place endpoints or sequences of endpoints with an interactive tool, or paint maps of greater or lesser edge weights. We have begun to investigate sketching as a means of controlling overall graph shape, and this would seem to be rich ground for further exploration.

Acknowledgements

We thank Dr. Prusinkiewicz for his comments and advice. We also thank NSERC and the GRAND NCE for funding support.

References

[1] A. Bucksch, R. C. Lindenbergh, and M. Menenti. Skeltre - fast skeletonisation for imperfect point cloud data of botanic trees. In *3DOR'09*, pages 13–20, 2009.

tree	level i	graph shape	b	note	timing
a	1	cylinder and portion of sphere with $\alpha = 0.3\pi$	12		6.4s
	2,3	$\alpha = 0.3\pi$	3		
	4	$\alpha = 0.3\pi$	6		
b	1	cylinder and portion of sphere with $\alpha = 0.25\pi$	10		7.0s
	2	$\alpha = 0.25\pi$	5		
	3	$\alpha = 0.25\pi$	10		
c	1	cylinder and portion of sphere with $\alpha = 0.5\pi$	30		13.4s
	2, 5	partial sphere ($\alpha = 0.25\pi$)	2	subgraphs of level 2 - 8 use $\gamma = 0.12\pi$	
	3-8	partial sphere ($\alpha = 0.25\pi$)	1		
d	1	cylinder and portion of sphere with $\alpha = \pi$	2		5.6s
	2	$\alpha = 0.4\pi$	10		
	3, 4	$\alpha = 0.4\pi$	6		
e	1	cone	40		3.3s
	2 - 4	$\alpha = 0.25\pi$	1	subgraphs of levels 2 - 4 use $\gamma = -0.05\pi$	
	5	$\alpha = 0.25\pi$	3	At final level, build subgraphs around endpoints from all previous levels	
f	1	portion of sphere with $\alpha = 0.5\pi$	10		7.5s
	2, 3	$\alpha = 0.5\pi$	4	subgraphs of level 2 are oriented in the negative vertical direction	
	4	$\alpha = 0.3\pi$	6		
g	1	cylinder and cone	32		3.0s
	2 - 7	$\alpha = 0.25\pi$	1	subgraphs in levels 2 - 7 use $\gamma = 0.03\pi$	
	8	$\alpha = 0.25\pi$	8	At final level, build subgraphs around endpoints from all previous levels	
h	1	tilted cylinder and hemisphere	8		2.1s
	2	$\alpha = 0.5\pi$	6		
	3	$\alpha = 0.5\pi$	12		
i	1	cylinder and portion of sphere with $\alpha = 0.7\pi$	11		0.7s
	2,3	partial sphere ($\alpha = 0.45\pi$)	6		
j	1	cylinder and cone	16		3.2s
	2,3	cone	8		
k	1	portion of sphere with $\alpha = 0.3\pi$	13		4.2s
	2,3,4	$\alpha = 0.2\pi$	2		
	5	$\alpha = 0.3\pi$	4		
l	1	cone	16		1.3s
	2	$\alpha = 0.25\pi$	1	subgraphs of level 2 use $\gamma = -0.03\pi$	
	3	cone	10		

Table 2: Parameters for the models in Figure 14

[2] X. Chen, B. Neubert, Y.-Q. Xu, O. Deussen, and S. B. Kang. Sketch-based tree modeling using Markov random field. *ACM Trans. Graph.*, pages 109:1–109:9, 2008.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2009.

[4] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 275–286, 1998.

[5] T. Ijiri, S. Owada, and T. Igarashi. The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics*, pages 138–146, 2006.

[6] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Trans. Graph.*, 2010.

[7] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 397–410, 1996.

[8] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph.*, 2007.

[9] M. Okabe, S. Owada, and T. Igarashi. Interactive design of botanical trees using freehand sketches and example-based editing. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, 2006.

[10] W. Palubicki, K. Horel, S. Longay, A. Runions, B. Lane, R. Měch, and P. Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Trans. Graph.*, pages 58:1–58:10, 2009.

[11] Pov-Ray Org. http://www.povray.org/, 2011.

[12] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. *Computer Graphics*, 28:351–358, 1994.

[13] Y. Rodkaew, P. Chongstitvatana, S. Siripant, and C. Lursinsap. Particle systems for plant modeling. In *Plant Growth Modeling and Applications*, pages 210–217, 2003.

[14] A. Runions, B. Lane, and P. Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena*, 2007.

[15] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30:11:1–11:14, 2011.

[16] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. Image-based tree modeling. *ACM Trans. Graph.*, 2007.

[17] J. Weber and J. Penn. Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 119–128, New York, NY, USA, 1995. ACM.

[18] J. Wither, F. Boudon, M.-P. Cani, and C. Godin. Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Comput. Graph. Forum*, 28(2):541–550, 2009.

[19] H. Xu, N. Gossett, and B. Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 2007.

[20] L. Xu and D. Mould. Modeling dendritic shapes - using path planning. In *GRAPP (GM/R)*, pages 29–36, 2007.

[21] L. Xu and D. Mould. Constructive path planning for natural phenomena modeling. In *3IA 11th International Conference on Computer Graphics and Artificial Intelligence*, pages 59–69, 2008.