Pacific Graphics 2015
N. J. Mitra, J. Stam, and K. Xu
(Guest Editors)

Volume 34 (2015), Number 7

Procedural Tree Modeling with Guiding Vectors

Ling Xu† and David Mould‡

Carleton University, Ottawa, Canada

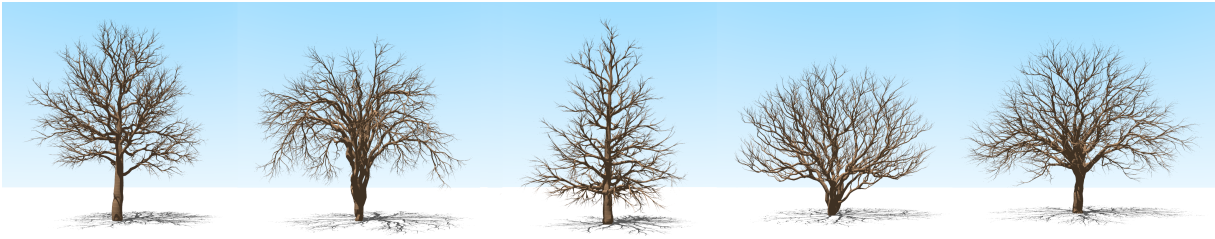

Figure 1: *Sample tree models synthesized using our method.*

Abstract

We propose guiding vectors *to augment graph-based tree synthesis, in which trees are collections of least-cost paths in a graph. Each node has an associated guiding vector; edges parallel to the guiding vector are cheap, but edges are more expensive when their orientation differs from the guiding vector. We further propose an incremental method for assigning guiding vectors over the graph, in which a node's guiding vector is an incremental rotation of that of its parent. We present a complete procedural system for tree modeling; our use of guiding vectors enables the graph-based method to produce high-quality tree models resembling a variety of real-world tree species.*

Categories and Subject Descriptors (according to ACM CCS): I.6.5 [Computer Graphics]: Model Development—Modeling methodologies

1. Introduction

Modeling natural phenomena such as plants, mountains, and oceans has long been a topic of great interest for computer graphics. Trees have been a particular source of fascination, with numerous automatic and semi-automatic methods for tree modeling proposed over more than thirty years. Synthetic tree models have become extremely realistic.

Despite the attention paid to the problem of tree modeling, some problems remain. For automated methods, the objectives of speed and controllability are elusive. Data-driven and manual approaches have their own drawbacks: it is inconvenient to obtain laser scans or photographs of a desired tree, and human intervention can be costly.

In this paper, we present an automated tree modeling algorithm. Our goal here is to model the tree's architecture – the spatial arrangement of trunk and branches – and not other details such as leaves and bark. Our algorithm possesses several desirable qualities. It is moderately fast, requiring only a few seconds to generate a full tree (27k branch segments). It is automatic, requiring no user guidance or external data such as point cloud or photographic representations of measured trees; it is controlled by a collection of parameters, many of which can be left at default settings. Finally, the method is versatile, capable of generating a wide variety of trees; some examples appear in Figure 1.

Our synthetic trees are collections of least-cost paths through a weighted graph, where an irregular graph provides

† lingxu@cmail.carleton.ca
‡ mould@scs.carleton.ca

a discrete representation of the space within which the tree will reside. Least-cost paths have been used for trees before, but without the ability to control the intermediate-scale shape of the trees; the synthetic branches do not curve in a way resembling real tree branches.

This paper introduces the concept of *guiding vectors* for weighted graphs: each node has an associated vector, and the weight of each outgoing edge will be set such that travelling along the vector is inexpensive, and cost increases as the direction of travel deviates from the vector's direction. Thus, least-cost paths will follow the vectors, allowing us to specify the medium-scale flow of the branches while retaining the advantages of single-source shortest paths, such as guarantees of no loops or self-intersections.

The paper makes two main contributions:

- It proposes guiding vectors for setting edge weights. Prior methods (particle tracing, self-organizing trees, graph-based trees) did not have an effective way to specify local directions; architectural control was available only coarsely, through attraction point placement, through endpoint placement, or through a density map. Guiding vectors provide local control over branch orientation.
- It suggests a mechanism for assigning guiding vectors. The guiding vectors can be set according to a global vector field. However, we have found it worthwhile to merge the vector field creation and the tree creation processes. We propose to determine the guiding vectors incrementally as the shortest-paths algorithm progresses, computing a guiding vector for a newly visited node as a small rotation of its parent's guiding vector. This approach allows us to specify intermediate-scale tree architecture with a few simple rules.

We also present a complete graph-based tree modeling system, with automatic graph construction and endpoint selection. The remainder of the paper is devoted to describing the algorithm and showing trees generated using the method.

2. Previous Work

Many methods have been proposed to synthesize tree structures. L-systems and later extensions are most notable: L-systems are a grammar that generates strings, subsequently interpreted as different branching structures and used for modeling of plant growth and plant ecosystems [PJM94, MP96, DHL*98]. One issue with L-systems is the difficulty of global control; it is a challenge to devise a system of rules to obtain a particular effect. Prusinkiewicz et.al [PJM94] proposed a *synthetic topiary*, where a user-specified bounding shape restricts the plant growth. Benes et al. [BSMM11] divide space into guide regions for branch growth, and obtained a high-level depiction by editing the guides. While spatial operations can provide top-down control over grammar-based methods, the need to write the production rules can be daunting.

The geometric method of Weber et al. [WP95] represents trees as collections of narrow near-conical tubes. A tree structure is generated from transformations of the segments; a user adjusts meaningful parameters, such as branching angles and segment lengths. Unfortunately, creating a new tree is cumbersome, requiring adjustment of up to 80 parameters.

A broad trend in computer graphics is data-driven synthesis, where models are created based on real-world measurements such as images [RCSL03, NFD07, TZW*07] or laser scans of geometry [XGC07, BLM09, LYO*10, AK14] and the development of geometry over time [LFM*13]. Such methods can be extremely effective. However, they are often intended for reconstruction, not synthesis; obtaining new types of trees is beyond their scope.

Laser scans create point clouds, and reconstruction methods can be based on least-cost paths through an inferred graph, as in the method of Xu et al. [XGC07]. Bucksch et al. [BLM09] approach the problem similarly, but use an octree to organize the point cloud data. Livny et al. [LYO*10] sought to reconstruct multiple overlapping trees simultaneously, applying a series of global optimizations based biologically-derived heuristics. Laser scanning trees is difficult because of trees' complex shapes and frequent occlusions; Aiteanu et al. [AK14] addressed this with a method that distinguishes between a densely sampled core and a sparsely sampled outer region.

Graph-based methods can be used for tree synthesis as well as tree reconstruction, procedurally generating a graph rather than measuring one. Xu and Mould [XM12] constructed trees within a hierarchy of procedurally-created graphs. Their method provided good global structure, but the details of the branch shapes were unconvincing.

In an effort to create natural tree shapes with a biologically plausible process, Runions et al. [RLP07] devised the space colonization algorithm. Attraction points signal the availability of growth space; the tree extends towards nearby attraction points, which are removed when the space is no longer empty. The tree structure is formed gradually, developing from the root towards the attraction points. The idea was exploited by Palubicki et al. [PHL*09] for self-organizing trees; the resulting shapes can be controlled by constraining the space for tree growth and by properly distributing the initial attraction points. Stava et al. [SPK*14] proposed a method based on plant biology, and automatically determine the parameters to reproduce an input tree model. Wang et al. [WYZB14] start from a structure defined by botanical meaningful parameters, and iteratively apply top-down optimizations to make the resulting tree structure match a given guide shape. This group of methods has proven highly effective for tree modeling.

Full manual creation of trees is exhausting, but sketch-based methods reduce the needed user effort. Ijiri et al.'s system [IOI06] allows users to control L-systems with sketched strokes. The TreeSketch system of Longay et al. [LRBP12]

is highly sophisticated: users can interactively control tree development by sketching, with branches generated using the space colonization algorithm.

Sketching and data-driven approaches can be combined, with sketches used to select tree elements from a database. Okabe et al. [OOI06] convert 2D freehand sketches to 3D branches under the assumption that branches maximize the space around them; they add new branches using existing branches as examples. Chen et al. [CNX*08] infer a tree template from user-supplied strokes, and the parameters of the template guide the subsequent tree shape. Using a similar idea, Wither et al. [WBCG09] combine user sketches of foliage with botanical heuristics to determine branch connections and distribution. These methods are effective, but depend heavily on user input; we aim at an automated method with minimal user involvement.

3. Algorithm

3.1. Algorithm Breakdown

Our algorithm is composed of four steps. First, we populate with nodes the volume where the tree will reside, and link nearby nodes with directed edges. Second, we apply Dijkstra's algorithm to find least-cost paths from the source to all other nodes. We compute edge weights under lazy evaluation: a node's guiding vector is set when it is first opened, as a rotation from its parent node's vector, and undefined edge weights are assigned based on the relationship between the edge vector and the guiding vector. The branches tend to follow the guiding vectors, since edges are cheapest along the guiding vector direction. Third, we select endpoints in the graph and determine the least-cost paths from the source to these endpoints; the union of all such paths is a branching structure that will eventually form the tree. Finally, the source is set to the set of paths so far and the second and third steps are repeated, creating a hierarchical structure. Details of each step are described in the following.

3.1.1. Graph initialization

We construct an irregular graph by distributing nodes in a region of space and connecting nearby nodes with edges. The region is typically a square (in 2D) or cube, but other shapes are possible. The nodes themselves take a Poisson disc distribution, where the size of the disc dictates the branch feature size and the minimum branch tip spacing. Nodes are linked in a Yao-8 graph [Yao77], where the space surrounding each node is divided into 8 sectors and the node connects to the nearest node in each sector. Having a fixed number of outgoing edges per node, but an irregular spatial distribution of nodes, lets us limit potential lattice artifacts while maintaining predictable memory usage.

Each node has an associated data structure to store needed per-node information: its ID; distance from the source; ID of parent node; guiding vector; and outgoing edges. Guiding vectors are unit vectors used to determine edge weights: edge weights will be less for edges oriented along the guiding vector, and more expensive as the directions diverge.

The root node has distance zero, and has a vertical guiding vector. It is typically placed at the bottom centre of the graph. All other nodes have distance infinity and undefined guiding vectors; their vectors will be set in the next step.

The use of a single, irregular graph both allows us to reason about the tree in a unified framework and prevents lattice artifacts. One might imagine that a regular cubic or tetrahedral lattice would be superior: graph connectivity would not need to be stored, reducing memory requirements. However, the resulting lattice artifacts are apparent even if the node positions are jittered, and we have opted for the improved visual quality from the irregular graph.

3.1.2. Guiding vector and edge weight definition

We next compute shortest paths from the source to every other node in the tree. The source has known cost and known guiding vectors; the costs and guiding vectors of all other nodes are unknown. Note that although the source is a single root node in the first iteration, later iterations use a source which is the full set of paths so far, thus causing branches to be hierarchically added to the emerging tree structure.

Lazy evaluation of guiding vectors and outgoing edge costs is done in concert with the path planning; a node's guiding vectors is found by rotating its parent node's guiding vector. The individual branch shapes thus follow a well-defined trajectory while maintaining an overall organization of the global tree structure.

Our implementation uses Dijkstra's algorithm, which maintains a frontier of nodes with provisional costs and a set of nodes with finalized costs. Each step moves a node from the frontier to the finalized set. In our method, the event of finalizing a node's cost also fixes its guiding vector: the guiding vector is determined by rotating its parent's guiding vector. Typically, the rotation axis is the cross product of the up vector and the parent's guiding vector; the rotation angle is determined algorithmically, using specific rules or user-assigned parameters. A constant rotation angle causes branches to curve upwards or downwards in a predictable trajectory. Other strategies are possible, as detailed below.

Once the newly opened node has a guiding vector, the weights of outgoing edges are set. The weight of an edge E is $d_E(1-\vec{e}_E \cdot \vec{v}_E)$, where d_E is the length of the edge, $\vec{e}_E$ is a unit vector along the edge, and $\vec{v}_E$ is the guiding vector. Note that both $\vec{e}$ and $\vec{v}$ are unit vectors, so that the weight has a minimum when the directions coincide and a maximum when the directions are opposite. Other functions are of course possible, but we have found this equation to be effective. Variations arise from altering the endpoint placement and the rules governing the guiding vectors.

In the absence of obstacles, shortest-path calculations tend to produce paths heading approximately along a straight line linking the origin and destination. Our use of guiding vectors, however, can produce natural-seeming curving paths. Figure 2 shows a comparison with and without guiding vectors. The crooked path on the left follows the shortest path through the graph. On the right, the guiding vectors push the shortest path upwards at first; only gradually does the path change course. Notice that the path is not a streamline through the vector field: it is still computed according to the shortest-path algorithm, using edge weights that were set based on guiding vector direction. Also note that the guiding vectors are not a field: they are stored on a per-node basis and never evaluated at intermediate locations.

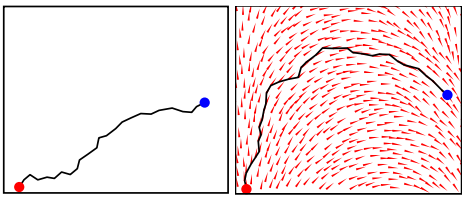

Figure 2: *Left: a path from graph without guiding vectors; right: a path from graph with guiding vectors.*

3.1.3. Branch construction

Next we extend the tree with a new set of branches. A subset of graph nodes are selected as endpoints; the least-cost paths from the source to these points will be new branches augmenting the emerging tree. Various strategies for endpoint selection are possible. Perhaps the simplest is to randomly select the endpoints from a user-specified volume; we usually employ this approach for the first level in our hierarchy. For later levels, a strategy that is aware of the current tree shape is called for; we suggest choosing endpoints from among the nodes that are k hops from the source in the previous iteration, thus causing the tree to extend outward. More details appear below.

3.1.4. Hierarchical structure

To complete the tree, we repeat the previous two steps a few times, typically 4-5 iterations in total. Nodes that are not part of the structure are reinitialized to have unknown cost and unknown guiding vectors; edge weights also become undefined. Then, the nodes in the structure are set as source nodes, i.e., added to the frontier with distance zero; optionally, the guiding vectors for the source nodes are changed; and the shortest-paths algorithm is used to propagate guiding vectors and costs into the remainder of the graph. A new set of endpoints is selected, the paths to these endpoints recorded, and these paths added to the structure, whereupon the process repeats again.

We illustrate the iterative tree construction in 2D in Figure 3 and in 3D in Figure 4. The background of Figure 3 shows the guiding vector directions; the coherence arising from the vector propagation method is apparent.

Typical trees have a central trunk and a collection of primary branches from which secondary, tertiary, and smaller branches spring. For this shape, we dedicate the first iteration to the trunk, with a single endpoint; more shrub-like trees can instead use more endpoints. Endpoint placement in the second iteration then becomes critical in defining the overall shape of the tree. Our practice has been to use a Poisson disc distribution within a bounding volume approximating the tree crown. Later iterations will fill out the tree from this basic shape. We can see the process in Figure 10, in which the bounding volumes are shown.

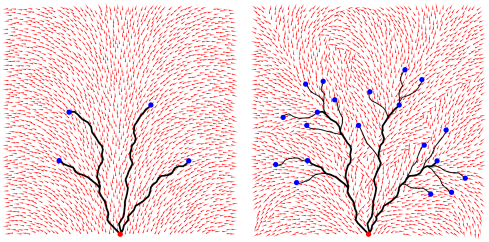

Figure 3: *Iterative shortest paths and guiding vector assignment. Left: first iteration; right: second iteration.*

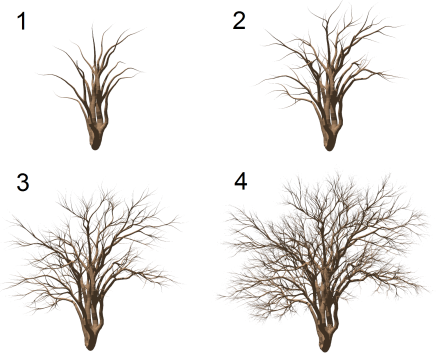

Figure 4: *Four iterations of tree construction.*

3.2. Elements of the Algorithm

3.2.1. Guiding vector transformation

The guiding vector for a newly opened node is an incremental rotation of its parent's: the vector is rotated about an axis perpendicular to both the up direction and the traversed edge. A typical example ($\alpha = 4^o$) is shown in Figure 5, upper left. With constant rotation angle α, we are limited in the types of tree we can create.

Making α a function of distance along the branch gives us more flexibility. Even a piecewise constant function can produce worthwhile results. Discretizing distance into the count of edges traversed (number of hops, h), we can write

$$\alpha(h) = \begin{cases} \alpha_1, & \text{if } h \leq h_0; \\ \alpha_2, & \text{otherwise.} \end{cases} \quad (1)$$

By varying the angles α_1 and α_2 and the switchover point h_0, we can obtain different types of trees. Figure 5 (upper right) shows an example with $\alpha_1 = 6^o$, $\alpha_2 = -6^o$, and $h_0 = 20$; the branches bend outwards and downwards for some distance, then begin to bend upwards. This is a simple heuristic used to create plausible tree shapes. Unless otherwise stated, we use Equation 1 for all remaining trees.

The rotation parameters can be changed for later iterations; an example is shown in the lower row of Figure 5. The upper row shows the initial tree structure, and for both trees, later iterations use $\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$. A large α_1 produces a noticeable fork between a branch and its children, while $\alpha_2 = 0$ ensures that the child branches are generally straight.

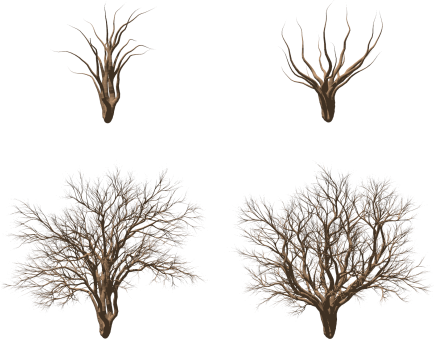

Figure 5: *Left: constant rotation angle; right: rotation angle reverses after some distance. Above: first iteration; below: after four iterations.*

Some additional variations on the rotation settings are shown in Figure 6. These examples differ solely on the rotation policy in the second iteration; the first iteration is used only to create a trunk (so $\alpha = 0$) and the later iterations produce straight branches ($\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$).

The partial trees after the second iteration are shown in the top row of Figure 6. Tree (a) has $\alpha = 10^o$, producing branches that strongly curve downward. Tree (b) uses $\alpha_1 = 60^o$, $\alpha_2 = 0$ and $h_0 = 1$: its branches emerge at a fixed upwards angle from the trunk and are fairly straight. Tree (c) uses $\alpha_1 = R$, $\alpha_2 = 0$ and $h_0 = 1$, where R denotes a random angle between 30 and 120 degrees; i.e., each node on the trunk has a different random direction. The branches point in different directions, but are fairly straight. Finally, tree (d) uses $\alpha_1 = 15^o$, $\alpha_2 = -5^o$ and $h_0 = 5$.

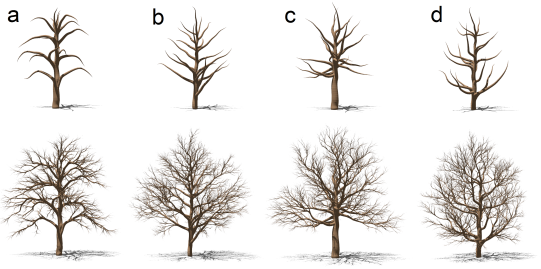

Figure 6: *Trees generated using different rotation settings. Above: results after two iterations; below: results after five iterations.*

Guiding vectors can also be set using a rule that incorporates global information. For example, we might set the rotation angle based on the angle to the central axis:

$$\alpha = \theta \times \sqrt{\beta/\pi} \quad (2)$$

where β is the angle, in radians, between the node's position vector and the up vector, with the root at the origin; θ is the parameter controlling the amount of rotation. In Equation 2, dividing by π ensures that the argument ranges from 0 to 1, and the square root then changes most quickly when close to the central axis, with more slowly changing angles when further away. A result obtained with $\theta = 25^o$ is shown in Figure 7, resembling an elm tree. Note that Equation 2 is used only for the first iteration; later iterations use $\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$.

Figure 7: *Left: the structure after one iteration; right: after four iterations.*

Instead of using incremental rotations to get guiding vectors, we can set vector fields directly. Existing tools for vector field construction can be deployed, such as field primitives [WH91]. The advantage is that the vector field can be directly manipulated. The disadvantage is that it may be difficult to produce the highly divergent fields that yield natural-looking trees. We have found it more straightforward to produce believable trees using incremental transformations. Nonetheless, direct generation of fields of guiding vectors is a possibility.

We directly generated the vector field to produce the trees in Figure 8: for examples (a), (b), and (c), we use a helical field, a rotational field about the z-axis, and a field tracing upwards along spherical surfaces, respectively. The branches approximately follow the field directions. Again, note that the branches are not streamlines; the guiding vectors alter edge weights, but the edges do not conform to vector field directions, and branches can deviate from field directions whenever doing so produces a shorter path. Accordingly, singularities in the vector field are not as grave a concern as they would be for streamline-based tree structures.

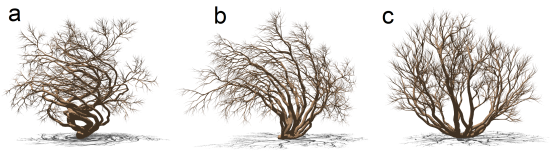

Figure 8: *Tree models from global guiding vector fields.*

3.2.2. Early endpoint placement

The placement of the earliest endpoints has a profound effect on the overall shape of the tree. We recommend placing endpoints in a user-specified volume, akin to the process of Xu and Mould [XM12]. Figure 9 shows an example: all three trees use the same parameters, but their first-generation endpoints are placed within different volumes. The final trees, after four iterations, show the lasting influence of the initial volume. We have found half-ellipsoids to be useful shapes, both because they approximate some commonly seen trees and because they offer a convenient parameterization. Note the difference between the endpoint selection mechanism we use and the clipping mechanism of the synthetic topiary [PJM94]; our final trees do not conform very closely to the specified volume, but produce a less structured and more organic shape.

Because each endpoint creates a separate path, using multiple endpoints in the first iteration leads to shrub-like trees. For a more conventional tree with a central trunk, a single endpoint should be used. A bounding volume in the second iteration can then guide the shape of the crown. Figure 10 shows two examples; the first iteration places the trunk, the second defines a crown shape with an ellipsoid, and later iterations add further definition to the tree shape. We discuss endpoint placement in later iterations next.

3.2.3. Late endpoint placement

While the earliest endpoints govern the general tree shape, endpoint placement in later iterations determines the further development of the tree. We must strike a balance between preserving the initial shape and fleshing out the preliminary structure of the early iterations. We place later generations of endpoints on a surface extrapolated from the tree so far. The

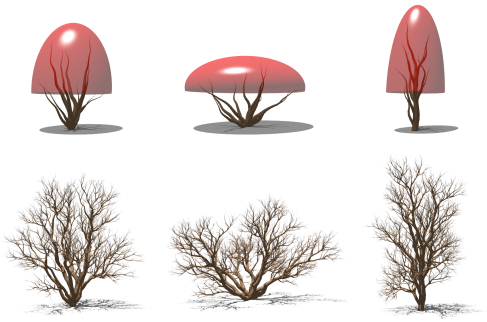

Figure 9: *Tree models from different bounding volumes.*

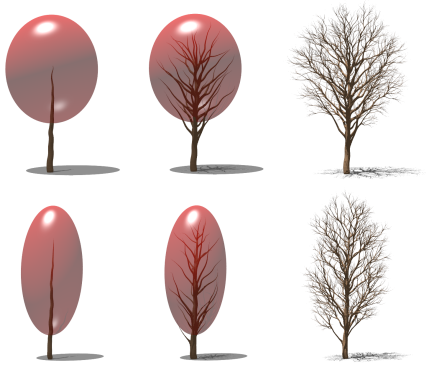

Figure 10: *Trees with trunks: bounding volumes control endpoint placement in the second iteration.*

surface is defined by the set of nodes exactly k hops from the existing tree, for some parameter k. We use the heuristic that trees grow outwards from the root, and hence order the surface nodes by the number of edges from the root; we then take the subset of the surface consisting of its outermost fraction, for some proportion t. Endpoints are selected from among the nodes on this surface subset.

By using hop count rather than Euclidean distance, we exploit the irregular graph structure to obtain an irregular surface shape. The exclusion zone for the initial inflated tree approximates a sphere but has some variation. Further variation is also possible, for example by using graph distance with random weights rather than strictly counting edges. We opted to parameterize the size of the exclusion zone as a proportion of nodes, making it scale-free.

The general approach is illustrated in Figure 11. The inflated tree appears on the left; the exclusion zone and remaining surface are shown on the right. Endpoints for the next iteration are chosen from the portion of the surface lying outside the exclusion zone. Figure 12 shows trees created with different exclusion zone sizes. With $t = 0.3$, endpoints are concentrated in the outer regions. At $t = 0.6$, more of the

tree shape is available for endpoint placement; we consider this to strike a good balance and have used it for most of our examples. At $t = 1$ there is no exclusion zone – all nodes on the initial surface are potential endpoints – and the entire tree is covered with small twigs.

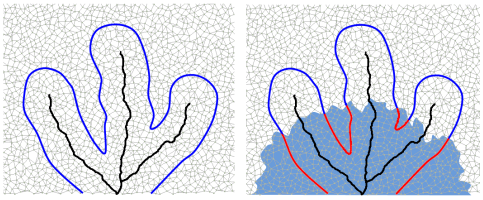

Figure 11: *Left: the surface of nodes k hops to the source. Right: the exclusion zone.*

Figure 12: *From left to right:* $t = 0.3$, $t = 0.6$, $t = 1$.

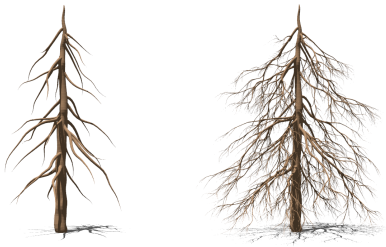

Figure 13: *A tree with branches growing downward. Left: result from two iterations; right: five iterations.*

Additional policies can augment the above method to synthesize particular tree shapes. One possibility is to allow only endpoints that are lower than their source nodes; the resulting branches appear to hang down, as in an example shown in Figure 13.

3.2.4. Endpoint density

Our synthetic trees can be made more sparse or full by adjusting the density of endpoints. Figure 14 shows three trees; from left to right, the trees have approximately 1000, 2000 and 4000 endpoints. Since there is a one-to-one correspondence between endpoints and branches, the models with more endpoints are more filled out.

Figure 14: *Tree models with approximately 1000, 2000, and 4000 endpoints from left to right.*

4. Results and Evaluation

This section shows several trees created with our method and provides comparisons with previous tree synthesis algorithms. We begin by showing a set of trees from fixed parameter settings: the random placement of graph nodes and endpoints means that similar but distinct trees can be generated by rerunning the algorithm with the same parameters. We used the parameters used to generate Figure 6(c) to make additional ash trees, shown in Figure 15. The trees are recognizably similar, such that an observer could identify them as belonging to the same virtual species. The sequence demonstrates the robustness of our method; we display the first six results we generated, none of which could be deemed a failure case.

Figure 15: *A series of trees generated with fixed parameter settings but different random choices.*

The parameter space for our method is smooth: intermediate settings produce intermediate trees. Figure 16 shows trees obtained by varying bounding volumes and guiding vector rotations. These trees were created with half-ellipsoids as bounding volumes and used Equation 2 to govern guiding vectors; from left to right, the half-ellipsoid bounding volumes have aspect ratios of 1.5, 0.9, 0.5, and 0.3, and θ takes on values of 14^o, 20^o, 26^o, and 32^o. The distributions of twigs and the branch densities are similar among all trees, but the overall shapes differ; there is a clear and smooth progression from one to the next.

Varying the parameters more widely can create more distinct trees, some of which are shown in Figure 17. Although

Figure 16: *A series of trees with different crowns and branch shapes.*

evaluating the quality of synthetic trees is difficult, in our judgement the trees have a desirable combination of detail and plausibility, evoking a sense of looking at real species of trees. Tree (a) has distinct primary branches, obtained by using a small number of endpoints in the first iteration. Tree (b) uses $\alpha_1 = 30^o$, $\alpha_2 = -10^o$ and $h_0 = 3$ in the upper portion, and $\alpha = 8^o$ for the drooping branches of the lower part. The spindly tree in (c) has short horizontal branches that give the impression of a dead tree. Tree (d) is intended to resemble a cottonwood, with broadly spreading branches and substantial empty space. Trees (e) and (i) have initial guiding vector directions set according to height, pointing more steeply upward as the source point elevation increases. Trees (f) and (h) are shrub-like, lacking a discernible central trunk, with many endpoints in the first iteration. Tree (g) shows a birch with divided trunk. Finally, tree (j) is intended to mimic a zelkova; it does not have small twigs and was generated over only two iterations. Some simple additional rules were sometimes employed. For example, tree (d) uses the rule that all endpoints must be higher than their source point, leading to a pronounced upward trend in the perceived growth pattern. Tree (a) combines a large number of endpoints in later iterations with few endpoints in the earliest iterations, leading to visible branch clusters.

A typical result was created in a graph of 150,000 nodes and has between 4000 and 8000 endpoints. The trees shown were each generated in under ten seconds once parameters had been selected. The trees of Figure 15, for example, required around 9.5 s each: about 6 s to construct the Yao graph, and another 2 s for path planning, with endpoint selection and other minor tasks making up the balance. Note that the graph does not necessarily need to be regenerated for each tree: in an interactive setting, the same graph could be reused for each experiment. Our timing figures are with respect to a desktop computer with 2.8 GHz CPU and 3 GB RAM.

Our approach offers certain benefits. Based on path planning, the method guarantees the absence of self-intersections in the final tree. Path planning is widely used, and we can take advantage of speedups in the base algorithm. The method is completely automated, with no need for a database of measurements or exemplars, and no user intervention beyond specifying parameters. Tree shape is governed by a few parameters and by direct spatial and geometric considerations, such as the shape of the bounding geometry for early endpoint placement. Unlike the unfamiliar parameters sometimes seen in biologically-inspired algorithms, the concrete parameters we use simplify parameter selection for generating novel tree types. Once parameter settings have been chosen, the same settings can be used to create numerous trees of the same general type, as seen in Figure 15.

The iterative construction process does not constrain the branch connections, creating a recognizable but loose hierarchy. Path planning controlled by guiding vectors produces irregular, natural-looking curves. The process is versatile, capable of generating a variety of tree shapes; examples appear in Figures 1 and 17. One key strength of our model is its ability to create irregular, gnarled trees, an area where particle tracing struggles. In short, we present a fast, well-rounded, expressive procedural method for tree modeling.

However, the method has some limitations as well. We have found our approach to incremental control over guiding vectors to be useful and powerful, but recognize that not all users will be comfortable creating vector fields in this way; we would like to explore alternative mechanisms for setting guiding vectors in future work. Endpoint placement is controlled indirectly, a consequence of our decision to automate the tree synthesis process. Likely the largest limitation of the graph-based approach is its high memory cost, $O(n^3)$ in the graph resolution. With 150,000 graph nodes, the path from top to bottom is barely 50 edges long. While the irregular graph structure does a great deal to conceal the low graph resolution, multiscale or at least variable-resolution graphs seem warranted.

Lastly, we give a visual comparison of our results with those of selected earlier methods. Figure 18 compares Neubert et al.'s image-based particle tracing method [NFD07] with ours. Their tree is highly realistic, corresponding closely to the photograph on which it is based. We are able to obtain a very similar tree by placing second-iteration endpoints in a cubic volume and setting α_1 to a random value in the range (30^o,120^o); we have $h_0 = 1$ and $\alpha_2 = 0$ for straight branches. In the remaining iterations (3-5), we use $\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$. Although we used their tree as a visual reference when setting parameters, our algorithm did not actually require any image data. We compare with Palubicki et al.'s self-organizing trees [PHL*09] in Figure 19. We used a half-ellipsoid as the bounding volume to get a similar shape, and created branches that slightly bend up with positive α, akin to Figure 6(d). The overall effect is comparable; one difference in our method is the visible crookedness of the branches as they follow the irregular graph. Finally, we compare with the method of Xu and Mould [XM12], which shares our graph-based tree modeling philosophy. We created a version of their oak tree using a half-ellipsoid shell for the second-iteration endpoints, and a small rotation angle for the guiding vector field. The resulting tree has a pleasing

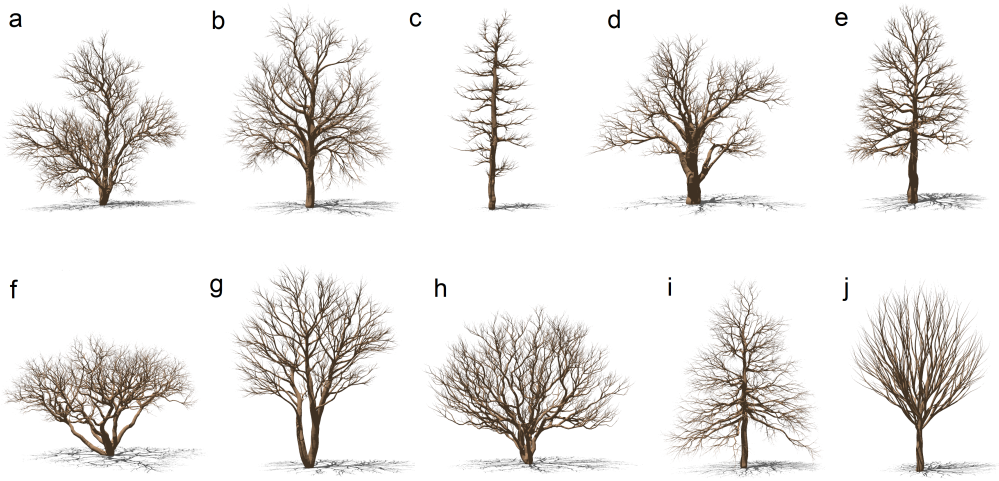

Figure 17: *A variety of different types of trees.*

unity of shape, believable sweep of branches, and overall a cleaner and more attractive appearance.

Our method provides comparable results to existing methods, and its algorithmic underpinnings are simpler. The guiding vectors offer a subtle yet powerful variation on the general path planning technique used previously for tree synthesis and reconstruction, allowing branch tropism to be included in this framework. Our suggested parametric assignment of guiding vector direction is flexible and controllable, while the path planning element enforces a unity over the resulting tree; although particle tracing techniques can be equally flexible, they do not naturally create a tree, and when their flexibility is exploited, they demand extra attention to ensure that the result is a tree without self-intersections and other defects. By improving on the results of earlier graph-based procedural tree modeling to the point where it is competitive with other procedural techniques, we have enlarged the space of available algorithms for tree modeling.

Figure 18: *Left: a model from particle tracing [NFD07]. Right: our tree model.*

Figure 19: *Left: a self-organizing tree model [PHL*09]. Right: our tree model.*

Figure 20: *Left: a model from iterated graphs [XM12]. Right: our tree model.*

5. Conclusions and Future Work

We proposed guiding vectors to augment graph-based tree synthesis. Each node in the graph has a guiding vector; outgoing edges have weights set according to whether the edge aligns with the guiding vector direction, thus inducing least-cost paths in the graph to conform to the guiding vector field. In the graph-based approach, synthetic trees are made up of collections of least-cost paths.

We incorporate the guiding vector field creation into the path planning process: a node's guiding vector is an incremental rotation of its parent's. In this way we can not only specify a full vector field with a rule about the transformation, we can also ensure that individual branches will tend to respect the vector field. Although the fields may be highly divergent, the trees still look natural, as in the case of Figure 18 with its branches going in multiple directions. Incremental path endpoint placement over multiple iterations creates a perceptible but non-strict hierarchy. Overall, with judicious placement of path endpoints, we can create elaborate, believable, high-resolution trees.

Several directions for future work are possible. We have concentrated on creating individual trees; it is worth investigating constructing multiple trees simultaneously in a single graph. Adding sketch-based and example-based control over the endpoint distribution and guiding vector field creation would be useful. Alternatively, simulation could be used to create vector fields.

Graph-based methods have been used primarily for reconstruction, and enlisting guiding vectors to help in that domain would be productive. Also, using guiding vectors to help with other modeling tasks, such as creating rivers or cracks, is an obvious direction.

Finally, we are also interested in enhancing the existing graph-based approach. At present, guiding vectors alter the edge weights unimodally; multimodal edge weight adjustments are possible, so that there can be multiple favored directions. Also, we would like to reduce the memory requirements of our approach. The fixed resolution of our graph simplifies the implementation, but a hierarchical or variable-resolution graph would reduce the memory usage.

References

[AK14] AITEANU F., KLEIN R.: Hybrid tree reconstruction from inhomogeneous point clouds. *Vis. Comput. 30*, 6-8 (2014), 763–771. 2

[BLM09] BUCKSCH A., LINDENBERGH R. C., MENENTI M.: Skeltre - fast skeletonisation for imperfect point cloud data of botanic trees. In *3DOR'09* (2009), pp. 13–20. 2

[BSMM11] BENES B., STAVA O., MĚCH R., MILLER G.: Guided procedural modeling. *Computer Graphics Forum 30*, 2 (2011), 325–334. 2

[CNX*08] CHEN X., NEUBERT B., XU Y.-Q., DEUSSEN O., KANG S. B.: Sketch-based tree modeling using Markov random field. *ACM Trans. Graph.* (2008), 109:1–109:9. 3

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), SIGGRAPH '98, pp. 275–286. 2

[IOI06] IJIRI T., OWADA S., IGARASHI T.: The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics* (2006), pp. 138–146. 2

[LFM*13] LI Y., FAN X., MITRA N. J., CHAMOVITZ D., COHEN-OR D., CHEN B.: Analyzing growing plants from 4d point cloud data. *ACM Trans. Graph. 32*, 6 (2013), 157:1–157:10. 2

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (2012), pp. 107–120. 2

[LYO*10] LIVNY Y., YAN F., OLSON M., CHEN B., ZHANG H., EL-SANA J.: Automatic reconstruction of tree skeletal structures from point clouds. *ACM Trans. Graph. 29*, 6 (2010), 151:1–151:8. 2

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), SIGGRAPH '96, pp. 397–410. 2

[NFD07] NEUBERT B., FRANKEN T., DEUSSEN O.: Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph. 26*, 3 (2007), 88–95. 2, 8, 9

[OOI06] OKABE M., OWADA S., IGARASHI T.: Interactive design of botanical trees using freehand sketches and example-based editing. In *ACM SIGGRAPH 2006 Courses* (2006), SIGGRAPH '06. 3

[PHL*09] PALUBICKI W., HOREL K., LONGAY S., RUNIONS A., LANE B., MĚCH R., PRUSINKIEWICZ P.: Self-organizing tree models for image synthesis. *ACM Trans. Graph.* (2009), 58:1–58:10. 2, 8, 9

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. *Computer Graphics 28* (1994), 351–358. 2, 6

[RCSL03] RODKAEW Y., CHONGSTITVATANA P., SIRIPANT S., LURSINSAP C.: Particle systems for plant modeling. In *Plant Growth Modeling and Applications* (2003), pp. 210–217. 2

[RLP07] RUNIONS A., LANE B., PRUSINKIEWICZ P.: Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena* (2007), pp. 63–70. 2

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse procedural modelling of trees. *Computer Graphics Forum 33*, 6 (2014), 118–131. 2

[TZW*07] TAN P., ZENG G., WANG J., KANG S. B., QUAN L.: Image-based tree modeling. *ACM Trans. Graph. 26*, 3 (2007), 1–7. 2

[WBCG09] WITHER J., BOUDON F., CANI M.-P., GODIN C.: Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Comput. Graph. Forum 28*, 2 (2009), 541–550. 3

[WH91] WEJCHERT J., HAUMANN D.: Animation aerodynamics. *SIGGRAPH Comput. Graph. 25*, 4 (July 1991), 19–22. 5

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 119–128. 2

[WYZB14] WANG R., YANG Y., ZHANG H., BAO H.: Variational tree synthesis. *Computer Graphics Forum 33*, 8 (2014), 82–94. 2

[XGC07] XU H., GOSSETT N., CHEN B.: Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph. 26*, 4 (2007), 19:1–13. 2

[XM12] XU L., MOULD D.: A procedural method for irregular tree models. *Computers and Graphics 36*, 8 (2012), 1034–1047. 2, 6, 8, 9

[Yao77] YAO A.: *On constructing minimum spanning trees in k-dimensional spaces and related problems*. Tech. rep., Stanford University, 1977. 3