Computers & Graphics 36 (2012) 1036–1047

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Graphics Interaction

A procedural method for irregular tree models

Ling Xu *, David Mould

Carleton University, Canada

ARTICLE INFO

Article history:
Received 2 July 2012
Received in revised form
18 August 2012
Accepted 22 August 2012
Available online 31 August 2012

Keywords:
Procedural modeling
Tree modeling
Natural phenomena
Geometry synthesis

ABSTRACT

We present a method to generate models for trees in which we first create a weighted graph, organized based on the Yao graph, then place endpoints and root point and plan least-cost paths from endpoints to the root point. The collection of resulting paths forms 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, composed of simple geometric primitives arranged in part manually and in part procedurally. Users can create desired variations by adjusting the initial graph shape; 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. Environmental effects can also be incorporated into the models by automatic parameter adjustment. The final trees are both intricate and convincingly realistic in appearance.

1. Introduction

Trees are commonplace in the natural world, and tree models often appear in the virtual worlds of computer games and films. Fig. 1 shows some tree structures. In the top image, we see some typical characteristics of trees: irregular individual branches and multiple levels of detail. A typical global tree shape, seen in the lower image, includes a single trunk, some main branches, and smaller branches that form the tree crown in a hierarchical structure. The overall tree shape possesses a complex beauty, which unfortunately is tedious to create with human labor. To ease the burden on digital artists, procedural modeling methods have been devised, able to create many types of models but especially useful for complicated subjects such as trees.

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 [1–3]. 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.

An earlier version of this paper [3] described a modeling method involving sequences of graphs, using path planning to link endpoints in all graphs with a single root node. In this extended version, we build our initial graph using the Yao graph to reduce the number of edges without compromising quality; we propose refining the shapes of graphs to get more natural details in the resulting tree structures; and we suggest an approach to incorporate environmental factors into our tree growth process. 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 [4–6]. General control over grammar-based methods is offered by Talton et al. [7], although their sampling process can be very time-consuming. The space colonization method of Runions et al. [8] offers a biologically motivated alternative with control over global shape, exploited by Palubicki et al. [9] for self-organizing tree modeling; here, the tree growth process follows the competition of branches for resources (e.g., light and space) with internal

* Corresponding author.
E-mail addresses: lxuc@scs.carleton.ca (L. Xu), mould@scs.carleton.ca (D. Mould).

0097-8493/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cag.2012.08.005

Fig. 1. Examples of tree structures.

signaling mechanisms based on L-systems. The resulting forms can be controlled interactively, e.g., by sketching [10].

Geometric methods (e.g., that of Weber et al. [11]) explicitly vary geometric quantities such as segment length and angles; generating models involve adjusting a huge number of parameters. An alternative is to use input images to drive tree creation [12–14]: such methods can attain extremely high quality, although the need to supply input images is a drawback.

The basic idea of path planning [15] for tree modeling is due to Xu and Mould [1,2], 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 [16–19]. Based on L-systems, Ijiri et al.'s system [16] 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. [17] 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 [18], 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 reduces the burden on user sketching. With similar stochastic optimization, Wither et al. [19] use a priori botanical knowledge to infer branch shapes from user sketched crown silhouettes at different scales, and can generate realistic tree models with good overall control.

Compared to the above methods, scanning methods focus on creating models of real trees, using point clouds of tree data obtained by 3D scanning. Xu et al. [20] 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. [21] extract a tree skeleton by subdividing the point cloud. Livny et al. [22] 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 [1], 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 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 by a policy, described below, and a random weight is assigned to each edge. Fig. 2 contrasts regular and irregular graphs: use of an irregular graph avoids lattice artifacts without necessitating higher resolution.

One option for the edge connection policy is simply to link two nodes whenever their distance is below some threshold. This policy is generally effective and was used for some examples in this paper. However, choosing the threshold is problematic: too large, and the number of edges per node is excessive; too small, and the graph can become disconnected. An alternative which

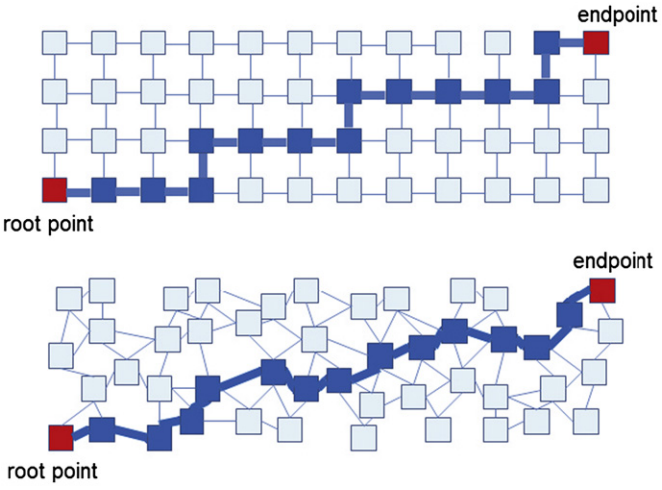

Fig. 2. A regular and an irregular graph.

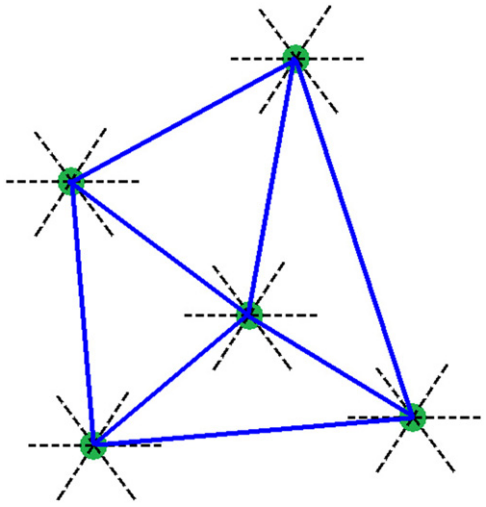

Fig. 3. Yao graph with six sections.

addresses this issue is to use the Yao graph [23], rediscovered as the Orthant Neighborhood Graph [24]. Edges in the Yao graph are determined by dividing the space surrounding a given node into sections with rays (in 2D) or planes (in 3D): an edge is added between the node and its nearest neighbor in each sector. Fig. 3 illustrates a Yao graph in 2D. Edges (solid lines) are connected as described above. The Yao graph automatically limits the number of edges while ensuring edges in every direction. Notice that there may be more than one edge per sector for a given node: in the example, each node originates at most six edges, but may receive additional edges originating from other nodes.

So far, we have described the process for tree generation using a single graph, thus producing 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.

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.

Pseudocode describing the building process is given in Fig. 4. While above we described the process in an iterative fashion, and our implementation is iterative as well, we found it more convenient to present pseudocode for a recursive implementation, echoing the visual recursion of 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 (truncated cone) around each edge in the structure. The radii of the

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 (subgraph size),
L (remaining lifespan)

Output:
T, a list of all path segments making up the tree.

1. Create the graph G 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. Create edges between nodes in G such that G is a Yao graph.
1H. For all edges, set random weights.

2. Create a set of endpoints S:
2A. $S \leftarrow \emptyset$.
2B. Choose a random node $e \in G$.
2C. Set $S \leftarrow \{S,e\}$.
2D. Repeat 2B to 2C while $|S|<b$.

3. Create paths for all endpoints $e \in S$; 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 centered at $e.\vec{x}$ with radius r that lies within angle α of $\vec{v_g}$
4D. If $L>0$, $T \leftarrow \{T, \text{maketree}(V,e,b,k,a*r,L-1)\}$

5. Return T.

Fig. 4. Pseudocode for tree construction.

cylinders are computed as follows. Working backwards from each endpoint of the final level, we compute the sum of segment lengths to a given node; call this distance s. We then compute a thickness w for the node using a tapering parameter ζ and the distance: $w=s^{\zeta}$. Larger values of ζ make the branches taper more quickly. A typical choice for ζ is 0.3.

A given node may lie on many paths. For a given path computation, if the thickness it proposes for an edge is larger than the thickness currently stored for this edge, replace the stored value with the larger one. In practice, we often found it worthwhile to trace

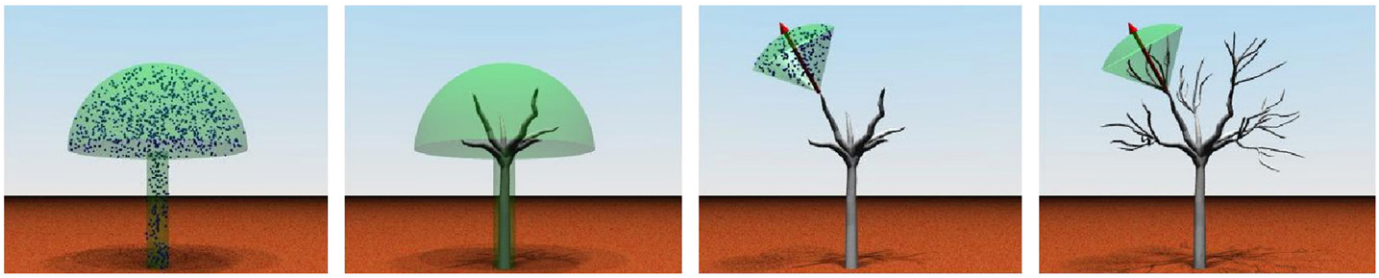

Fig. 5. Illustration of the iterative tree building process.

Fig. 6. A fractal dendrite in 3D.

endpoints from every level, not only those of the final level, and to use different ζ for different levels; this allows us to create a thicker trunk, if desired. The four-level tree in Fig. 7 used $\zeta = \{0.6, 0.5, 0.4, 0.3\}$.

Once thicknesses are known, we can place the truncated cone about the edges, using the thickness values stored in the nodes to provide the cone's radius at its ends. We also add a sphere to each node to conceal the join. We can render the structure in a schematic fashion by directly drawing the cones 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 [25] for that purpose in this paper.

Fig. 5 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, as illustrated in the third image. The rightmost figure shows the structure once the second level has been completed.

To capture the heterogeneous 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 Figs. 6 and 7. Fig. 6 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 Fig. 7; 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.

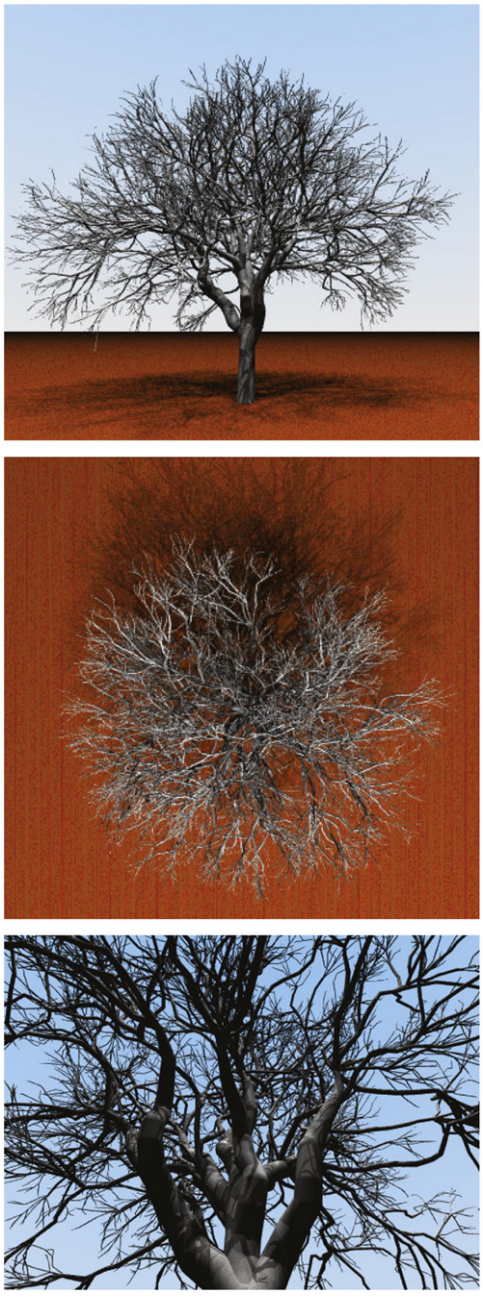

Fig. 7. A synthetic tree created with four iterations.

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.

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 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.

Fig. 8 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: Fig. 9 shows an example of modeling an irregular tree, imitating a photograph. The graph was composed and endpoints selected manually to match the desired outcome.

Using simple geometric primitives to build the graph in the first level can generate a uniform-looking tree structure. The overall tree silhouette follows the primitive shape. However, in general, natural trees have more irregular silhouettes (e.g. clusters of branches diverse in length, separated by gaps). We can control the level of irregularity by restricting endpoint placement to a shell near the surface of the volume, as described next. However, for a better balance between organization and chaos in the tree model, we propose refining the graph in the first level, described below.

We can control diversity in branch length by restricting endpoints to lie in a shell of thickness d_e near the surface of the graph. A greater value of d_e, yielding a large interval, produces branches with greater diversity of length; smaller d_e, and a small interval, forces endpoints into a thin shell so that the resulting paths have little diversity of length. Fig. 10 illustrates how d_e affects branch length variability. The graph is a composition of a hemisphere and a cylinder. Endpoints are placed in the shell between the outer and inner hemispheres. The thickness of the shell is indicated by the arrow. The resulting tree skeleton has not only long branches but also short branches. In the right tree, endpoints are placed close to the surface of the hemisphere; almost all branches have the same length. Fig. 11 shows the tree models after four growth levels. The left tree obtained with greater d_e has more diversity of branches and irregularity of silhouette, while the right one has a more homogeneous appearance.

For yet higher degrees of irregularity in the tree shape, we can refine the first-level graph, populating the large-scale structure with smaller elements. We chose to use spheres in the examples

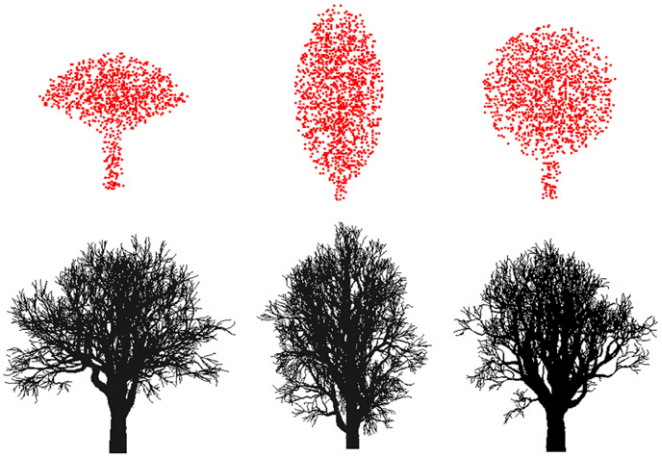

Fig. 8. Structures obtained using different shapes of graph.

Fig. 9. A tree with a tilted trunk. Top: initial graph; lower left: inspirational photograph; lower right: our model.

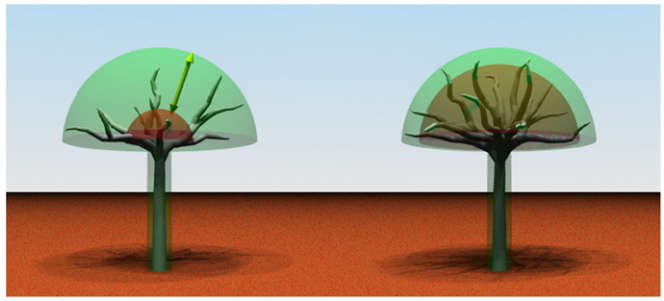

Fig. 10. Two structures with different shells to place endpoints. Left: a thicker shell can generate more branch variations in length; right: a thin shell provides more uniform looking branches.

Fig. 11. Two structures with different values of d_e.

for this paper, although other shapes could be substituted instead. We fill the coarse geometry with a Poisson disc distribution of points, where the disc spacing is quite large, allowing (say) only 4–10 points to be placed. For each such point, we place a sphere; we also place a conical volume connecting the sphere to the trunk or tree root. The union of all such volumes is filled with graph nodes, forming the first level in the iterated graph building. Subsequent levels are added as before. The process is visualized in Fig. 12.

The purpose of adding a conical volume attaching the sphere to the trunk is to ensure that endpoints within each sphere will have a path to the root. Further, it allows us to enforce a constraint on edge placement: no edges link nodes within different subvolumes at the first level. This constraint ensures that the individual subvolumes will remain distinct: their structure will not be blurred by shared edges. Also, when we place endpoints in the first level, we can allocate a fixed number to each subvolume, ensuring balanced coverage of the global graph shape.

Fig. 13 compares trees from the single-volume process to the subvolume process. Enforcing distinct subvolumes has provided noticeably more irregularity in the global shape, with greater complexity of branch distribution and higher diversity of silhouette; further, the branches are visibly organized into clusters, which enhances the natural appearance of the trees.

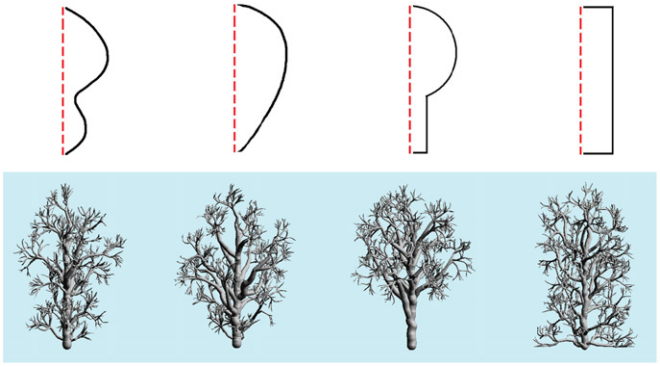

Fig. 14. Structures obtained by user sketches. Above: user sketched curves with rotation axes (dashed lines); below: resulting structures.

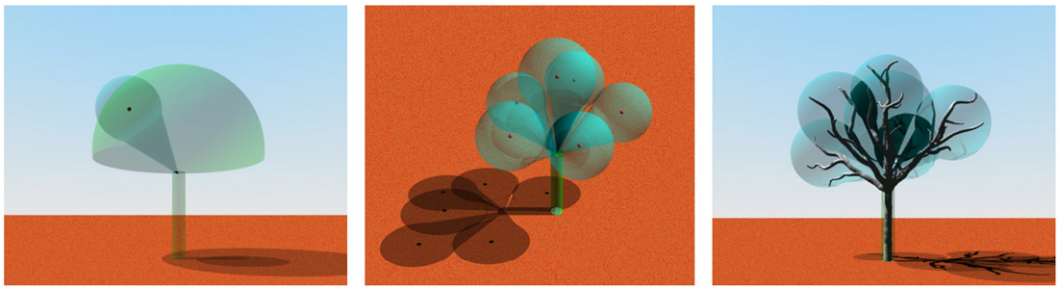

Fig. 12. The process to refine the basic graph volume with spheres and cones.

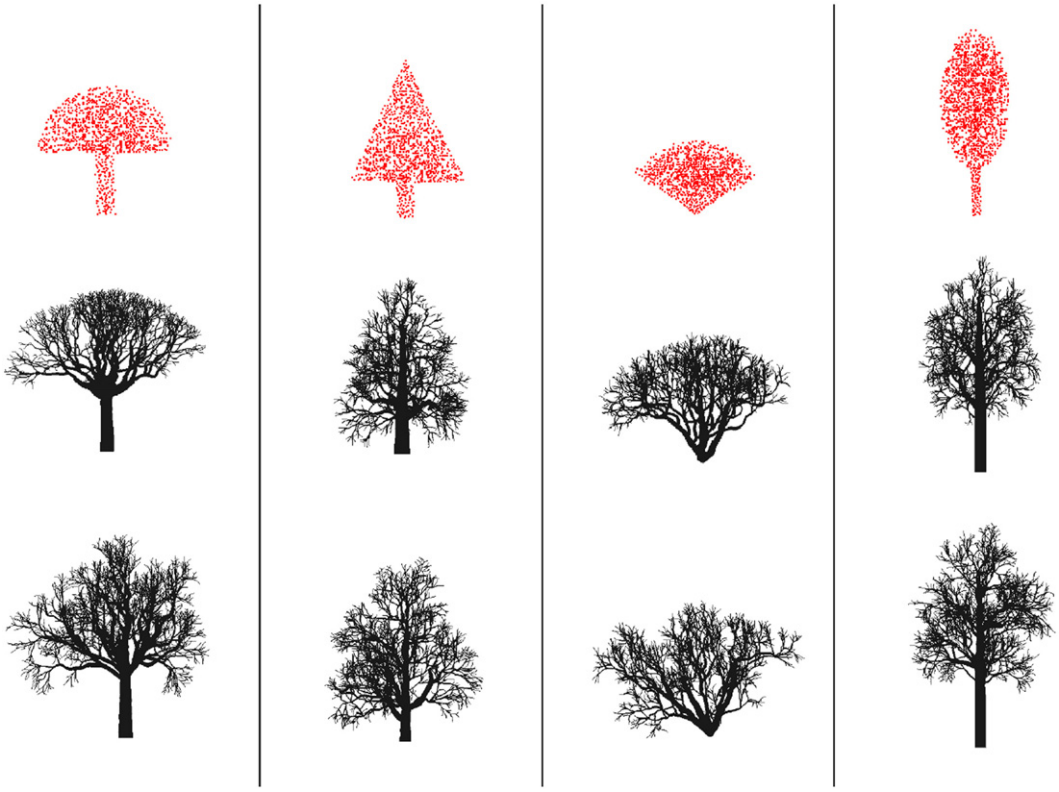

Fig. 13. Top row: basic volume to build graph; middle row: tree models obtained without a refinement of graph volumes; lower row: tree models obtained with a refinement of graph volumes.

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. Fig. 14 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.

Fig. 15. Left: tree with $a=0.7$; right: tree with $a=1$.

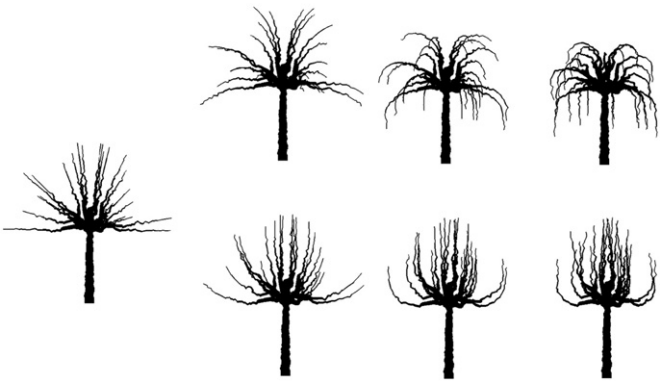

Fig. 16. Curving branches from progressively rotating subgraphs.

Fig. 17. Irregular bush and tree obtained by use of lifespan.

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 decreases 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. Fig. 15 shows two trees obtained with different a. The left tree, with $a=0.7$, demonstrates a clear hierarchical relationship in branch 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. Fig. 16 shows an example of structures obtained by the above method, with four iterations applied.

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

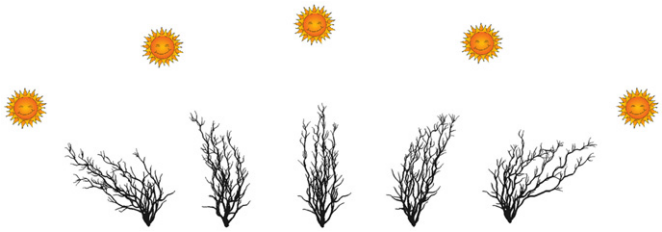

Fig. 19. Effect of light on the lifespan of branches.

Fig. 18. Effect of light on the number of endpoints. Left: a tree without light influence; right: trees affected by the light direction.

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.

Fig. 17 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.

3.4. Environmental effects

Environmental factors, such as temperature, daylight, and moisture, play important roles during the process of tree growth. Temperature governs the growth rate; light variations in intensity, quality, and duration affect the growth process; water is crucial [26]. Other factors such as wind direction and space restrictions (e.g., obstacles) may also influence tree growth. Each factor functions separately, while their composition decides the final tree structure. Although the compound influence of various environmental factors is a complicated problem [27], the impact from individual environmental factors is clear. Shading forces a

Fig. 20. Left: our tree model; right: real photograph.

tree to grow towards whatever light is available [28]. Branches with good access to sunlight are always denser and grow faster than those in darkness. Persistent strong winds can hinder branch growth. Faced by space restrictions, such as a wall close to the plant, plants respond by growing towards the direction with less restriction.

We can incorporate environmental factors into our algorithm by controlling specific parameters including number of endpoints, lifespan, and orientation $\overrightarrow{v_g}$ of the subgraph. We concentrate on the effect of light, but similar modifications could be done to account for other environmental factors, if desired. We want branches aligned with the light direction to have greater lifespan and to sprout more sub-branches. Suppose we have n branches at a given level. Then, we obtain n lifespan decrements and n sub-branch counts as random numbers from a distribution (say, a uniform distribution over some range). We compute the angle between the light direction and the subgraph orientation; to the branch with the smallest angle, we assign the smallest lifespan decrement and the largest sub-branch count, and continue for each branch in increasing order of angle. Thus, the branches aimed away from the light will have the lowest lifespan (greatest decrease in lifespan) and the fewest sub-branches.

The results are illustrated in two figures. Fig. 18 shows the effect of modifying endpoint count according to light direction. On the left, we show a tree without light influence, where the crown developed uniformly in all directions. On the right, the number of branches varies with the direction of light; branches are denser towards the sun and sparser when further from the light direction. Fig. 19 shows the effect of modifying lifespan according to light direction. Branches close to the light direction survive longest, while those further from the light direction more quickly cease growing.

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 previous methods. In Fig. 20, synthetic tree images are compared with photographs. Our trees have similar structures to the photographed trees: similar tree crown shapes, 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.

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. Fig. 21 shows three variations on a base tree 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.

Our use of the Yao graph allows us to produce good quality trees despite limited edge count, saving on memory. Our previous approach was to link nodes whenever their separation fell below a threshold; this technique was problematic because the outcome depended on the threshold. A small threshold risks creating a graph with gaps or even entirely disconnected subgraphs. While a larger threshold avoids these specific problems, it incurs memory costs due to large numbers of edges per node.

Fig. 21. Three trees of the same type.

Fig. 22 illustrates the tradeoff with three sets of trees; all have the same number of endpoints (1500) and use the same node count (33,000) in the underlying graph. The trees in the top row of the figure use the Yao graph; each node averages approximately six edges. The middle row uses a distance threshold approach to assign edges, with a threshold manually tuned to give good results while minimizing edge count; while the final trees are quite good, the graphs contain nearly double the edges of the Yao graphs. The bottom row shows the outcome of a low threshold, here selected to produce a graph with the same edge count as the Yao graph. The trees are somewhat distorted, and worse, the underlying graphs have become badly disconnected, in many cases lacking paths from endpoints to the root. Conversely, the Yao graph allows us low edge counts without compromising output quality.

By varying graph shape and available parameters, we can create a wide variety of trees; examples are shown in Fig. 23. 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.

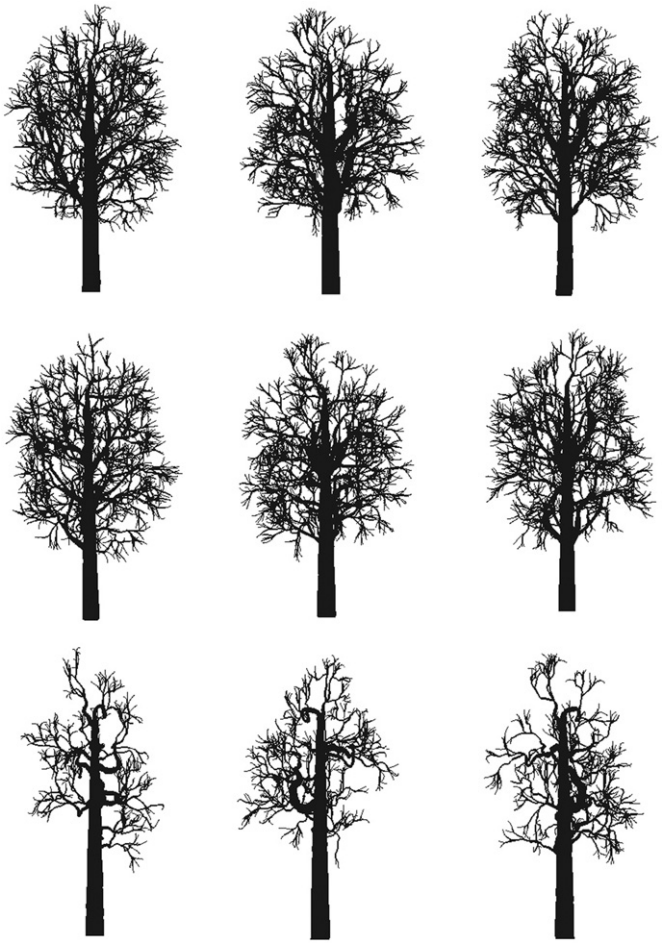

Fig. 22. Top row: trees obtained in Yao graph; middle row: trees obtained by edge connection with a proper distance threshold in graph; lower row: trees obtained by edge connection with a small distance threshold in graph.

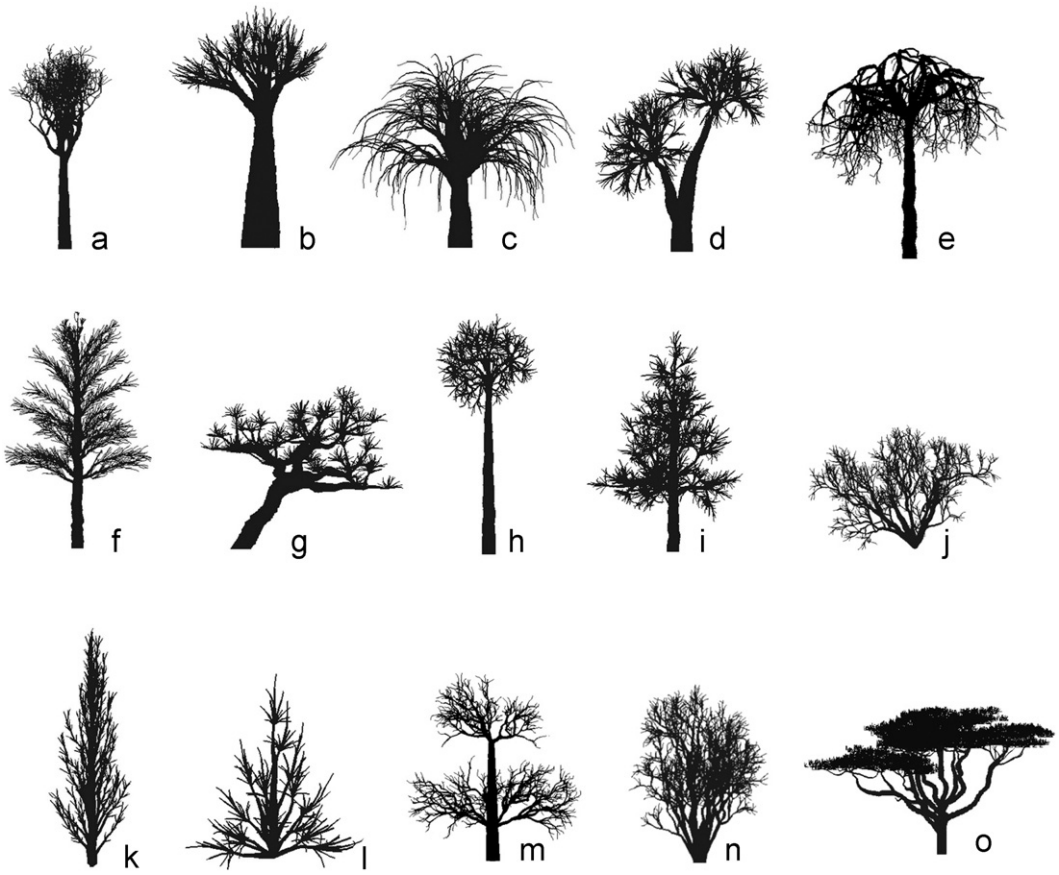

Fig. 23. Different types of trees.

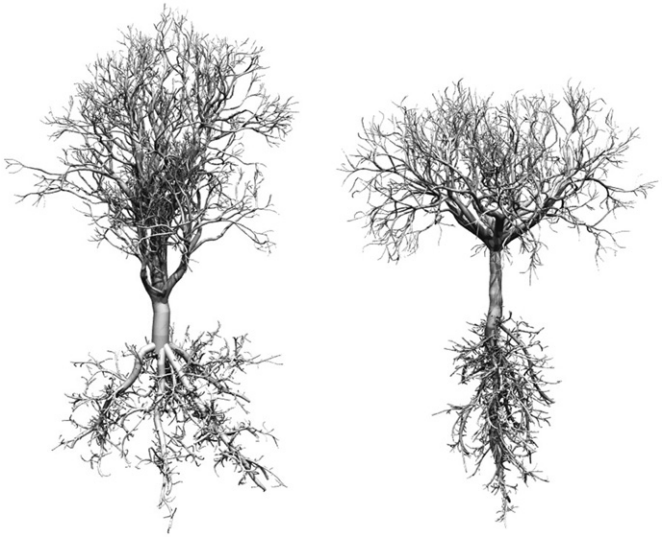

Fig. 24. Two trees with different root systems.

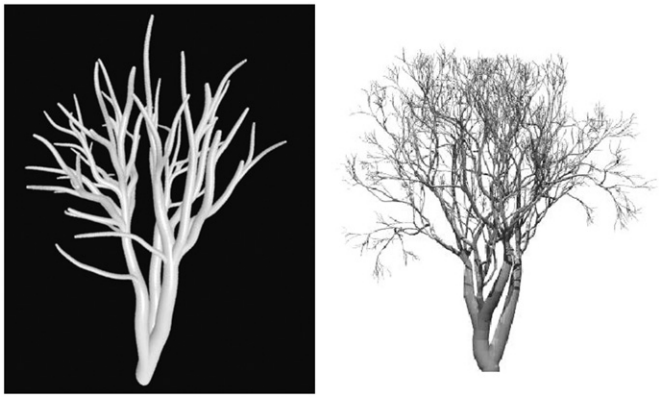

Fig. 25. Left: a model created by Xu and Mould; right: our tree model.

Fig. 26. Left: a model from Neubert et al.; right: our tree model.

In addition to the wide variety of trees shown, our algorithm can be used to model the root system. Fig. 24 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 Fig. 6.

Fig. 27. Left: a self-organizing tree model; right: our tree model.

Table 1
Timing and construction data for selected models.

Tree	Number of graph nodes	Number of endpoints	Timing (s)
Fig. 20 (top left)	351,810	34,740	32.4
Fig. 25	84,755	3732	4.5
Fig. 27	207,458	9330	20.0

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 Figs. 25–27. 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.

Fig. 25 shows our tree model compared with a tree model generated by Xu and Mould [2]. 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. Fig. 26 compares our tree model to an example by Neubert et al. [13] 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,

Table 2
Parameters for the models in Fig. 23.

Tree	Level i	Graph shape	b	Note	Timing (s)
a	1	Cylinder and portion of sphere with $\alpha=0.3\pi$	12		6.4
	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		7.0
	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	Rotate $\vec{v}_g$ of level 2–8 downwards with 0.12π	13.4
	2, 5	$\alpha=0.25\pi$	2		
	3–8	$\alpha=0.25\pi$	1		
d	1	Cylinder and portion of sphere with $\alpha=\pi$	2		5.6
	2	$\alpha=0.4\pi$	10		
	3, 4	$\alpha=0.4\pi$	6		
e	1	Portion of sphere with $\alpha=0.5\pi$	10	$\vec{v}_g$ of level 2 are oriented in the negative vertical direction	7.5
	2, 3	$\alpha=0.5\pi$	4		
	4	$\alpha=0.3\pi$	6		
f	1	Cylinder and cone	32	Rotate $\vec{v}_g$ in levels 2–7 downwards with 0.03π. At final level, build subgraphs around endpoints from all previous levels	3.0
	2–7	$\alpha=0.25\pi$	1		
	8	$\alpha=0.25\pi$	8		
g	1	Tilted cylinder and hemisphere	8		2.1
	2	$\alpha=0.5\pi$	6		
	3	$\alpha=0.5\pi$	12		
h	1	Cylinder and portion of sphere with $\alpha=0.7\pi$	11		0.7
	2,3	$\alpha=0.45\pi$	6		
i	1	Cylinder and cone	16		3.2
	2,3	Cone	8		
j	1	Portion of sphere with $\alpha=0.3\pi$	42		9.6
	2,3	$\alpha=0.3\pi$	3		
	4	$\alpha=0.3\pi$	4		
k	1	Cone	40	Rotate $\vec{v}_g$ of levels 2–4 upwards with 0.05π. At final level, build subgraphs around endpoints from all previous levels	3.3
	2–4	$\alpha=0.25\pi$	1		
	5	$\alpha=0.25\pi$	3		
l	1	Cone	16	Rotate $\vec{v}_g$ of level 2 upwards with 0.03π	1.3
	2	$\alpha=0.25\pi$	1		
	3	Cone	10		
m	1	Cylinder and sphere	50		6.8
	2,3	$\alpha=0.4\pi$	4		
n	1	Ellipsoid	35		6.7
	2–4	$\alpha=0.4\pi$	4		
o	1	Cylinder and portion of sphere with $\alpha=0.4\pi$	10		4.8
	2,3	$\alpha=0.25\pi$	1		
	4	Ellipsoid	12		

and that of Tan et al. [14], 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. Fig. 27 shows our tree model and a tree model by the self-organizing method of Palubicki et al. [9]. 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. We have begun to explore environmental factors, and shown the feasibility of incorporating environmental effects into our framework, but have not yet done this in a general way.

We close this section by providing parameter and timing information for some of our models. Timing information and statistics for selected tree models appear 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 s.

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 Fig. 23 is given in Table 2.

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. The detailed features of tree structures can be created by refining the initial graph volume. 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.

Our main direction for future work involves increasing the degree of user control over the output. 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. Also, our model currently only treats the main structure of the tree, and we would like to investigate phenomena including leaves, bark, fruit, and flowers.

Acknowledgments

We thank Dr. Prusinkiewicz for his comments and advice. Thanks to the photographers who contributed images through Flickr: Daveybot, Niels Linneberg, jsogo, pelican, Sakall and joiseyshowaa. Also thanks to NSERC and the GRAND NCE for funding support.

References

[1] Xu L, Mould D. Modeling dendritic shapes—using path planning. In: GRAPP (GM/R), 2007. p. 29–36.
[2] Xu L, Mould D. Constructive path planning for natural phenomena modeling. In: 3IA 11th international conference on computer graphics and artificial intelligence, 2008. p. 59–69.
[3] Xu L, Mould D. Synthetic tree models from iterated discrete graphs. In: Proceedings of graphics interface, 2012. p. 149–56.
[4] Prusinkiewicz P, James M, Měch R. Synthetic topiary. Comput Graph 1994;28:351–8.
[5] 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, SIGGRAPH'96, 1996. p. 397–410.
[6] 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, SIGGRAPH'98, 1998. p. 275–86.
[7] Talton JO, Lou Y, Lesser S, Duke J, Měch R, Koltun V. Metropolis procedural modeling. ACM Trans Graph 2011;30:111–14.
[8] Runions A, Lane B, Prusinkiewicz P. Modeling trees with a space colonization algorithm. In: Eurographics workshop on natural phenomena, 2007. p. 63–70.
[9] Palubicki W, Horel K, Longay S, Runions A, Lane B, Měch R, et al. Self-organizing tree models for image synthesis. ACM Trans Graph 2009:58:1–10.
[10] 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. p. 107–20.
[11] Weber J, Penn J. Creation and rendering of realistic trees. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques, SIGGRAPH'95. New York, NY, USA: ACM; 1995. p. 119–28.
[12] Rodkaew Y, Chongstitvatana P, Siripant S, Lursinsap C. Particle systems for plant modeling. In: Plant growth modeling and applications, 2003. p. 210–7.
[13] Neubert B, Franken T, Deussen O. Approximate image-based tree-modeling using particle flows. ACM Trans Graph 2007;26(3):88–95.
[14] Tan P, Zeng G, Wang J, Kang SB, Quan L. Image-based tree modeling. ACM Trans Graph 2007;26(3):1–7.
[15] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. MIT Press; 2009.
[16] Ijiri T, Owada S, Igarashi T. The sketch l-system: global control of tree modeling using free-form strokes. In: Smart graphics, 2006. p. 138–46.
[17] Okabe M, Owada S, Igarashi T. Interactive design of botanical trees using freehand sketches and example-based editing. In: ACM SIGGRAPH 2006 courses, SIGGRAPH'06, 2006.
[18] Chen X, Neubert B, Xu Y-Q, Deussen O, Kang SB. Sketch-based tree modeling using Markov random field. ACM Trans Graph 2008:109:1–9.
[19] Wither J, Boudon F, Cani M-P, Godin C. Structure from silhouettes: a new paradigm for fast sketch-based design of treesComput Graph Forum 2009;28(2):541–50.
[20] Xu H, Gossett N, Chen B. Knowledge and heuristic-based modeling of laser-scanned trees. ACM Trans Graph 2007;26(4):19:1–13.
[21] Bucksch A, Lindenbergh RC, Menenti M. Skeltre—fast skeletonisation for imperfect point cloud data of botanic trees. In: 3DOR'09, 2009. p. 13–20.
[22] 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 2010;29(6):151:1–8.
[23] Yao A. On constructing minimum spanning trees in k-dimensional spaces and related problems. Technical Report, Stanford University; 1977.
[24] Germer T, Strothotte T. Orthant neighborhood graphs—a decentralized approach for proximity queries in dynamic point sets. In: GRAPP (GM/R), 2007. p. 85–96.
[25] Pov-Ray Org 〈http://www.povray.org/〉; 2011.
[26] Kozlowski TT. Tree growth. Ronald Press Co.; 1962.
[27] Room P, Maillette L, Hanan J. Module and metamer dynamics and virtual plants. Adv Ecol Res 1994;25:105–57.
[28] Sibley D. The Sibley guide to trees. Alfred A. Knopf; 2009.