A Garden of Statistically Self-Similar Plants

Anne M. Burns
Mathematics Department
Long Island University, C.W. Post Campus
Brookville, NY 11548, USA
E-mail: aburns@liu.edu

Abstract

A simple fractal tree has the property of self-similarity, meaning it can be subdivided into parts, each of which is a reduced copy of the whole tree; its fractal dimension can be calculated from the scaling factor. Trees in nature exhibit an "approximate self-similarity". Using discrete time steps and probabilities, an algorithm for drawing plants that are not strictly self-similar, but that appear to belong to the same family, is described. The algorithm is based on a thesis of de Reffye and can be used in artwork and teaching. It has no claim to botanical accuracy, but creates images of purely decorative plants.
.

1. Modeling the growth of trees using probabilities and random numbers

The basic idea behind this model of plant growth is as follows: a tree is created from scratch, starting with a single node called the root *node*. The tree grows in a sequence of discrete time steps according to a set of probabilities. We start with simple rules; later, in order to more closely mimic nature, we will make these numbers functions of time. I encountered the basic idea in an article based on a thesis by Philippe de Reffye [2]. The article presents a method of modeling plants and trees faithful to their botanical nature; the figures in the paper are rendered on sophisticated computers using complicated rendering techniques and advanced data structures so as to obtain images with great realism. My goal is to distill and simplify the underlying mathematical ideas from articles like this and to cast them in a way that can be understood by undergraduate students. Inventing data structures and algorithms for modeling plant growth presents an interesting challenge for students in mathematics and computer science. The plant models described in this paper are purely imaginary.

A simple fractal tree has the property of self-similarity, meaning it can be subdivided into parts, each of which is a reduced copy of the whole tree; small portions of the tree when magnified look exactly like the whole tree. Its fractal dimension can be calculated from the scaling factor. In Figure 1 the binary trees look as if they might belong to the same "family" even though they are not identical. We might call them *statistically self-similar*. Several mathematical definitions of statistical self-similarity can be found in [1].

Figure 1: *Statistically self-similar binary trees*

2. A Simple Binary Tree

Our first tree is a *binary tree*. A binary tree is a finite set of nodes that is either empty, or consists of a *root* and two disjoint binary trees called the *left subtree* and the *right subtree*. A node contains information. The roots of the subtrees of a node are called its left and right *children*. If node A has node B as a child, then A is the *parent* of B. A node that has two empty subtrees is called a *leaf* or *terminal node*.

We assign to each node in the tree a positive integer called the *level* of the node. The root node has level 1; if the level of a node is *n*, the level of its child nodes will be *n*+1. Each parent node will be connected to its child nodes by *edges*, which will be interpreted as branches. Several examples of binary trees are rendered in Figure 1. The terminal nodes are rendered as disks.

Starting at the root node the tree grows in discrete time steps, $t_1, t_2,\ldots t_n$. At each time step a node in the tree can do one of three things: (i) ramify (create two child nodes), (ii) die and flower (that is, it cannot produce any more child nodes) or (iii) "sleep" for one time period. If a node dies it will become a terminal node. In a first simplified version we assign three nonnegative real numbers p_1, p_2 and p_3 whose sum is 1. These numbers represent the probabilities that a node (1) ramifies (creates two child nodes), (2) flowers (and dies), or (3) sleeps for one time period.

What information should each node contain? In our first attempt, in order to keep things simple, when a node is created it is assigned a positive integer representing its level. We also need to keep track of its (*x*, *y*) coordinates and the angle θ, the angle at which the line connecting its parent node to itself is drawn, so that if it has child nodes we know where to place them. And, finally, we need to know whether a node is capable of bearing child nodes; I use a Boolean variable, *alive*, to indicate that it is or is not able to ramify.

Before entering the loop that draws the tree, one node is created, the root node; it is assigned level 1, an initial point (x_0, y_0), and an initial angle θ_0 such as $\pi/2$. The Boolean variable, *alive*, is set to true. We also initialize three nonnegative real numbers whose sum is 1: p_1, p_2 and $p_3 = 1- (p_1+p_2)$.

The program enters a loop where each pass through the loop represents a tick of the clock or time period; initially we specify the number of stages, t_{max}, that the algorithm is carried out. At each pass we examine the list of nodes that have been created in previous passes through the loop whose *alive* status is true. For each live node we generate a random number *r* between 0 and 1.

Figure 2: *Three binary trees generated with the same program;* p_1=.5, p_2=.15 *and* p_3= .35. *Nodes are connected by lines; flowers are represented by disks,* $\theta_0 = \pi/2$ *and if a parent node's angle is* θ*, its child nodes are assigned* $\theta = \theta \pm \pi/12$.

Case 1: If $r < p_1$ the node ramifies and two new nodes are created. The information for the child nodes is entered into the new nodes. For each of the two new child nodes we assign: its level (the level of its parent plus 1), and its (x, y, θ) coordinates (if (x_p,y_p,θ_p) are the coordinates of its parent, then $x = x_p+l*\cos(\theta_p)$, $y = y_p+l*\sin(\theta_p)$ where l is the length of the connecting segment, also a function of the level), and set its *alive* status to true.

Case 2: If $p_1 \leq r < p_1+ p_2$ the node dies and flowers. We place a "flower" (a disk in Figure 1) at the node's (x,y) position and set its *alive* status to false.

Case 3: If $p_1+ p_2 \leq r \leq 1$ the node "sleeps" for one time period, so we leave it alone and proceed to the next node in the list.

Notice that if $p_1= 1$ then every node ramifies and we produce a *perfect* (and self-similar) binary tree.

Figure 2 illustrates the result of this simple program with the probabilities constant: $p_1=.5$, $p_2=.15$ *and* $p_3= .35$.

3. Probabilities as a function of time

Although I am not a botanist, it seems reasonable to me to expect that the probability of a node ramifying would be higher in the initial stages of growth of a plant. So to make the plants look more realistic we might consider making the probabilities a function of the time steps. We might want the probability of ramifying to be 1 at the beginning of the plant's life and the probability of flowering to be 1 in the last stage (call it t_{max}). Figure 3 shows three parabolic functions with this property and Figure 4 shows four trees using these probabilities with $t_{max} = 10$.

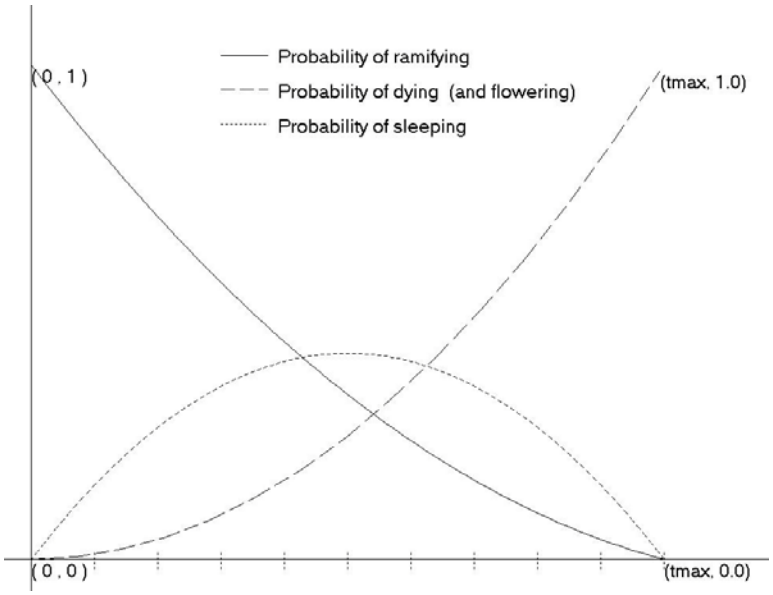

Figure 3: $p_1(t) = \frac{2t^2}{3(t_{\max})^2} - \frac{5t}{3(t_{\max})} + 1$, $p_2(t) = \frac{t^2}{(t_{\max})^2}$, $p_3(t) = 1 - p_1(t) - p_2(t)$

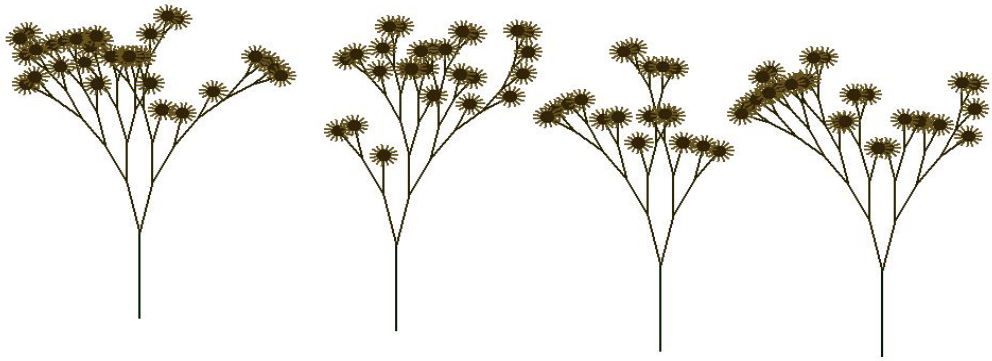

Figure 4: *Using the functions described in Figure 3 with t_{max} = 10.*

4. Other enhancements

There are many ways to modify the original model to achieve different styles of plant growth. In the original model if a node has level *n*, when it ramifies we assign level *n*+1 to its two child nodes. Instead we can assign level *n* to one of the child nodes (randomly) and level *n*+1 to the other. Then, when rendering the plant, we draw the child branch that has the same level as its parent at the same angle as its parent, as in Figure 5. This yields a "trunk" and alternate branching. In Figure 5 some curvature has been introduced into the segments connecting nodes whose level is greater than 1 and we have made the thickness of the connecting segments depend on the level of the parent node.

Figure 5: *A variation on assigning level to a child node.*

The tree in Figure 6 is drawn from the same program as the trees in Figure 5, but the parameters have changed; the curvature and thickness of the branches has been increased and includes the trunk, and a different rendering of the branches is shown. The coloring of the branches is achieved by a simple trick. Each branch is a curve drawn as a sequence of very short line segments whose width depends upon the

level of the parent node, and the RGB value of the color of each segment changes slightly from that of the previous segment. Flowers have been placed at the terminal nodes.

Figure 6: *A fanciful tree*

Figure 7 shows colorful versions of the tree in Figure 6. The short segments that make up the branches are colored randomly rather than "continuously" as they are in Figure 6.

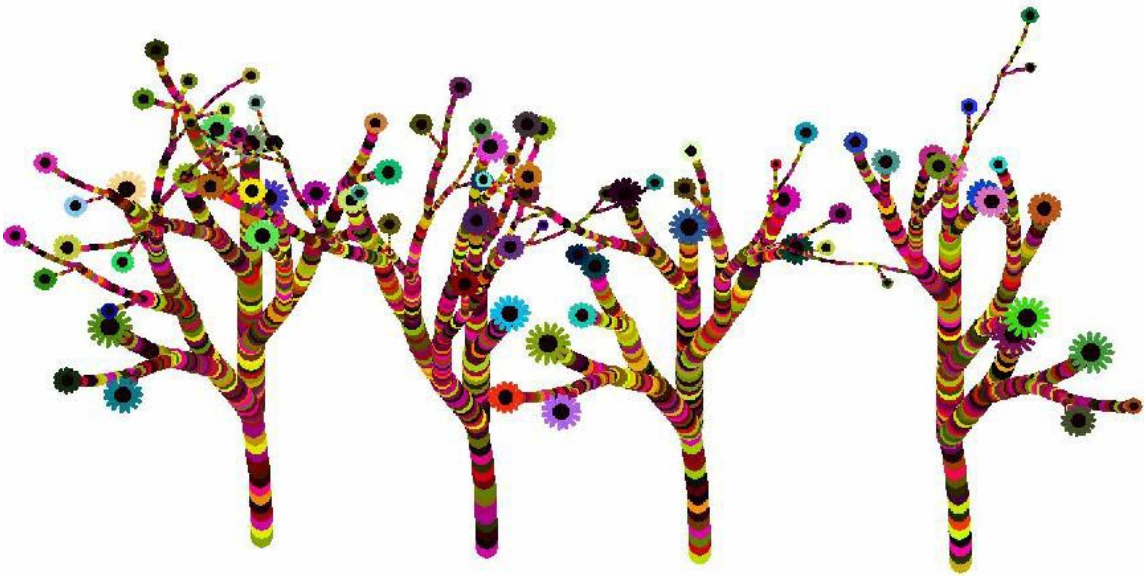

Figure 7: *Trees from a Star Trek planet*

Another variation can be seen In Figure 8. It is an easy task to modify the program so that when a node ramifies it gives birth to 3, 4 or more child nodes.

Figure 8: *Each node gives birth to three child nodes.*

Figure 9: *A more realistic looking plant.*

In Figure 9 a more realistic-looking plant is achieved by allowing the branching angles to vary with controlled randomness and inserting leaves along the branches that connect the nodes.

Once the basic structure of a plant has been created the fun begins. Now the imagination takes over. We can experiment with different branching angles and different flower and leaf shapes. We can go outside and look at the incredible variety of shapes found in nature and try to imitate them or to make up our own. To create these shapes we use basic mathematical functions like lines, polynomials or trigonometric functions. For example, a leaf shape can be defined by a pair of parametric equations: $x = a*t^2$, $y = b*\sin(\pi t)$, $-1 \leq t \leq 1$. Knowledge of the math in a basic pre-calculus course is sufficient to program these shapes. And a bonus is that often one's mistakes turn out to look better than what was originally intended.

Figure 10 shows a garden of statistically self-similar plants.

Figure 10: *A garden of statistically self-similar plants.*

5. Future work

Animation of scenes created from computer graphics is one of my projects for the future. The method of generating plant growth described in this paper may be useful for producing animations.

Recall that in the algorithm that created the tree, when a node dies (flowers) the node is not deleted, but merely has its *alive* status changed to false. Initially I had the program delete a node from the list after it died and flowered. Allowing each node to remain in the list after it dies but changing its status from *alive* = true to *alive* = false means that at the end of the algorithm we have a record of the age (level at which the node died) and coordinates of each of the nodes. Thus there is the possibility of animating the nodes; one

idea for an animation is to incrementally change the position of the nodes and their connecting branches in order to simulate trees blowing in the wind. Another idea is to create an animation of changing seasons. Figure7 depicts four frames of a crude attempt at an animation of falling leaves. I have rendered the nodes as leaves (originally green). As the animation progresses the leaves gradually change color from green to red, yellow and brown, and eventually fall from the trees and drift slowly to the ground.

Figure 11: *Four frames from an animation of falling leaves.*

References

[1] Heinz-Otto Peitgen and Dietmar Saupe, Editors, *the Science of Fractal Images*, Springer-Verlag, New York, 1988.

[2] Philippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger, Claude Puech, Plant Models Faithful to Botanical Structure and Development, *Computer Graphics*, Volume 22, Number 4, August 1988