Simulating Trees using Fractals and L-Systems

Eric M Church and SK Semwal
Department of computer Science, University of Colorado, Colorado Springs
semwal@cs.uccs.edu

Abstract

Algorithmic simulation of natural environments in a convincing manner presents an ongoing challenge to modelers and developers. Of particular challenge in simulating natural environments is the dynamic creation of botanical forms, that is, plants. Many plants exhibit complex structural forms that defy traditional geometric patterns, and as such, are difficult to simulate in a convincing way using traditional geometric and mathematical constructs. Using statically modeled techniques is cumbersome, especially when faced with the challenge of producing multiple similar, yet unique plant forms, such as found in a forest of trees. To adequately recreate a convincing organic environment, some dynamic and stochastic model is necessary. While the shape and structures of plants, particularly trees, often appear irregular and chaotic, they also exhibit a high degree of self-similarity. Self-similarity suggests a recursive or iterated approach to the dynamic modeling of botanical forms. As such, constructs such as fractal geometry and Lindenmayer systems seem prime candidates for the generation of such forms. This paper explores the use of fractal geometry as the basis of simulating the forms of trees. Lindenmayer systems (L-Systems), iterated function systems (IFS), their relationships, and their use in simulating trees are discussed. The primary focus of this paper is to present and explain the tree simulation software written for our implementation by extending the Honda model to include stochastic simulation.

Keywords: L-Systems, Plants.

1. Introduction

Algorithmic simulation of natural environments in a convincing manner presents an ongoing challenge to modelers and developers. The ultimate goal is to present a computer-generated artificial environment that is indistinguishable from a photographed natural landscape. This paper explores the use of fractal geometry as the basis of simulating the forms of trees. In the course of this paper, I will investigate Lindenmayer systems (L-Systems), iterated function systems (IFS), their relationships, and their use in simulating trees. Finally, the primary focus of this paper will be to present and explain the tree simulation software written for this project.

Plant structures often show very complex, yet well-defined structures. One of the primary factors that organize plant structures and contributes to their beauty is the concept of *self-similarity*, as characterized by Mandelbrot[7]: *When each piece of a shape is geometrically similar to the whole, both the shape and the cascade that generate it are called self-similar*.

Simply defined, fractals are geometric patterns that exhibit self-similarity on all scales, at any level of magnification. Fractals achieve self-similarity by being recursive in nature – each piece of a fractal is defined based upon other pieces of the fractal. Because more detail may be revealed at any level of magnification, fractals are often referred to as "infinitely complex"[10]. A canonical example of fractal geometry in action can be seen in the synthesis of the Koch snowflake Rather, fractals simply need to display the same *type* of structures on all scales. Most fractals exhibit approximate self-similarity rather than precise self-similarity, as seen in the Mandelbrot set.

Approximate self-similarity is a recurring theme in nature. Many natural forms fit neatly into the paradigm of self-similarity, including higher plants (such as trees), seashells, clouds, lightning, river networks, and blood vessels). This property is highly mathematical, and as such, provides a mechanism for algorithmically defining or simulating natural phenomena. Fractal geometry can be artificially generated using a number of methods. Two methods are discussed in the following sections: Lindenmayer Systems and Iterated Function Systems.

2. Previous Research

Lindenmayer systems, or L-systems for short, are a formalism introduced by Aristid Lindenmayer in 1968 as a theoretical framework for studying the structure and development of simple multicellular organisms. As Lindenmayer was a biologist, his original motivation for developing L-systems was in fact to model and study biological forms and growth processes. Since their introduction, L-systems have been successfully employed to generate models of many natural forms, including higher plants.

The parallel application of productions distinguishes L-systems from other formalisms, such as Chomsky grammars, which apply productions sequentially. Parallel production application has an essential impact on the formal properties of rewriting systems. For example, there are languages that can be generated by context-free L-systems but not by context-free Chomsky grammars.

The implicit parallelism of L-systems is directly based upon how biological growth processes function (e.g. multiple cells dividing in parallel during embryonic development). The result is an autonomous, massively parallel emergent system, where each agent is autonomous, but the resulting system may show emergent patterns. The first results in applying L-systems to graphical models were published in 1974 by Frijters and Lindenmayer, and Hogeweg and Hesper [7].

An **I**terated **F**unction **S**ystem, of IFS for short, is any system which recursively iterates a function or a collection of arbitrary functions on some base object. Iterated function systems are a common method used to generated fractals. This method was popularized by Barnsley, based on the theoretical work by Hutchison and Dekking [1].

IFS fractals can be defined on any number of dimensions, but are commonly computed and rendered in 2D. An IFS fractal is a solution to a recursive set of equations, and is made up of the union of several copies of itself, each copy being transformed by a function. The functions are normally contractive – they bring points closer together and make shapes smaller. The shape of an IFS fractal is made up of several possibly-overlapping smaller copies of itself, each of which is also made up of copies of itself, ad infinitum, which creates a self-similar structure. As can be inferred from this discussion of L-systems and iterated function systems, both formulations are very similar, and can be used to generate fractals. In fact, an L-system can be thought of as an iterated function system whose functions are productions and inputs are symbols from the alphabet of the system. Inversely, one can think of an iterated function system as a non-deterministic L-system whose alphabet is the set of all real numbers, and whose productions are functions or geometric transformations. While not strictly conforming to the definitions of either, the two can be perceived in a very similar manner. Many schemes for ordering branches in axial trees have been suggested and implemented. One such example is that proposed by Horton and Strahler, which was originally proposed to model erosional topologies and drainage basins for river networks. The Horton-Strahler method can also be used as the basis for generating botanical trees, as shown by Prusinkiewicz and Lindenmayer. The generated tree structure appears very organic. While this model is very simplistic, and only uses binary branching, it manages to convey the sense of a botanical tree quite well, and does not appear artificially fabricated. Such trees were good first steps toward achieving believable photo-realistic tree models, and served as the basis for much future work in the area of botanical modeling.

The plant kingdom is dominated by branching structures. For modeling purposes, a mathematical description of tree-like shapes and methods for generating them is required. To begin searching for such descriptions, we start in the realm of graph theory. In graph theory, a *rooted tree* has edges that are directed and labeled. The edge sequences form paths from an initial node, called the *root* or *base*, to *terminal* or *leaf* nodes. In a botanical context, these edges are referred to as *branch segments*. A segment that is followed by at least one more segment in at least one path is called an *internode*. A terminal segment is called an *apex*.

An *axial tree* is a special type of rooted tree with the botanically motivated notion of branch axis. At each node of an axial tree, at most one outgoing *straight* segment is distinguished. All remaining edges are called *lateral* or *side* segments. A sequence of segments is called an *axis* if:

- the first segment in the sequence originates at the root of the tree or as a lateral segment at some node,
- each subsequent segment is a straight segment, and
- the last segment is not followed by any straight segment in the tree.

Together with all its descendants, an axis constitutes a *branch*, which is itself an axial (sub)tree [7]. Within an axial tree, axes and branches are ordered. The axis originating at the root node of the tree has order zero. An axis originating as a lateral segment of an *n*-order parent axis has order *n*+1. The order of a branch is equal to the order of its lowest-order or *main* axis. A graphical representation of an axial tree is presented in Figure 1(a) [7].

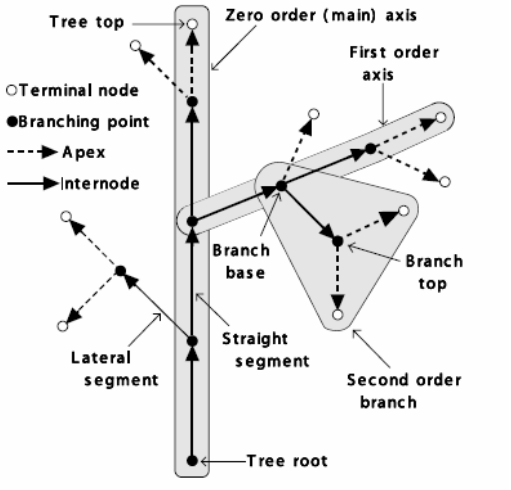

Figure 1(a): An axial tree

Many early models devised to simulate trees ignore interactions between various elements of a growing structure, as well as the interaction between the structure and its environment. Although interactions clearly influence the development of real plants, they also add significant complexity to such models. The first such simple model was developed by Honda, who studied tree forms with the following basic assumptions:

- Tree segments are straight, and their girth is not considered (i.e. the fact that branch diameter decreases proportionally to its order is not considered)
- A mother segment produces two daughter segments in one branching process
- The lengths of the daughter segments are shortened by constant ratios with respect to the mother segment (r_1 and r_2)
- The mother segment and its two daughter segments are coplanar, in a plane called the *branch plane*, and the daughter segments form constant angles (a_1 and a_2) with the mother segment
- The branch plane is fixed with respect to the direction of gravity so as to be closest to a horizontal plane. More formally, the line perpendicular to the mother segment and lying within the branch plane is horizontal. The one exception to this rule is for branches attached to the main trunk. In this case, a constant divergence angle, α, is maintained between consecutive lateral segments.

Honda's model geometry is displayed in Figure 1(b).

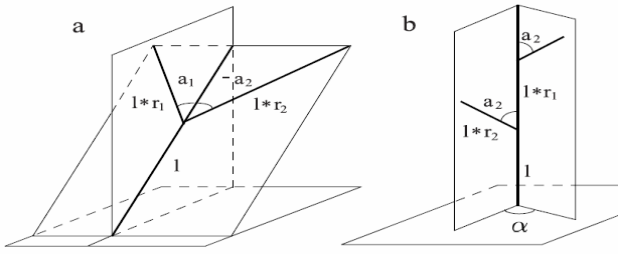

Figure 1(b): Geometry of Honda's simple tree model

By varying numerical parameters inserted into his model, Honda was able to produce a wide variety of tree-like shapes (see Figure 2). With improvements and extensions, Honda's model has been applied to investigate the structure of real trees. The most important extension to this model was applied by Aono and Kunii, in which branch positions are biased in a particular direction, in order to simulate the effects of wind, phototropism, and gravity. These extensions, and others subsequently applied, have successfully improved Honda's model to create even more realistic models.

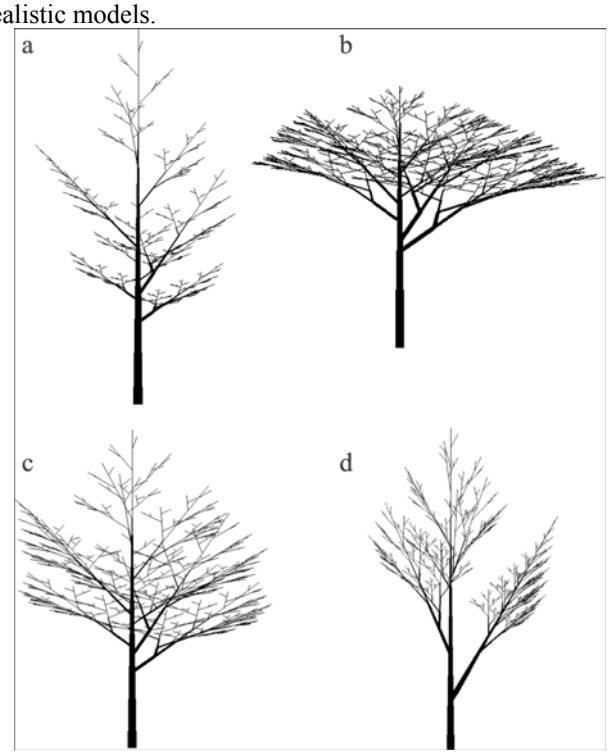

Figure 2: Trees generated using L-systems and Honda's model parameters

While Honda's models and those extensions immediately afterward were completely deterministic, many later modelers introduced stochastic methods to enhance the realism of models as well. Stochastic mechanisms are essential to the models developed by Reeves and Blau [10].

Employing stochastic processes, many different varieties of tree can be simulated by employing a single algorithmic model with different numeric parameters. Stochastic laws characteristic to different tree varieties can be employed, varying geometric parameters such as the length and diameter of internodes, branching angles, number of branches at intersections, etc. It is with this idea in mind that I developed my initial model for simulating tree structures, presented in the next section.

3. Implementation Results

The tree generation software written for this implementation was developed in C++ in a semi object-oriented manner. I debated using Java or C++, as I initially wanted to develop a graphical user interface to allow the user to dynamically change parameter values used in the generation algorithms, and be able to instantly view results. On a global scale, a tree is defined by:

- Radius of the main trunk
- Starting height (the height of the main trunk – all other braches are scaled in some way based on this starting height)
- Number of daughter branches coming from each mother branch

On a local scale, each branch is defined by:

- Scaling factor (all branches are scaled according to order)
- Height/length of a branch
- Lean angle (from the lower order branch)
- Rotation angle (around the lower order branch)

Simulator Global Parameters

Varying the number of daughter branches on a tree has a significant impact on the resulting appearance of the tree. Often, incrementing the number of daughter branches can enhance the appearance of the resulting image. For example, with the tree displayed in Figure 3, increasing the number of branches in sequence makes the tree appear as if it is growing, or as if the foliage is budding out after winter. This is a very desirable result, but is unfortunately an often-unpredictable result. In fact, in many cases, increasing the number of branches may result in a loss of detail, which can produce a very ugly, non-realistic looking tree. For example, in Figure 4, we see that using five branches creates an appealing tree structure reminiscent of an elm tree. However, if we increase the number of branches to six or seven, the resulting tree loses its detail, and is overrun by the rendering of foliage. The resulting image looks more like a cloud than a tree (in fact, this fractal generator could easily be modified to generate cloud structures!).

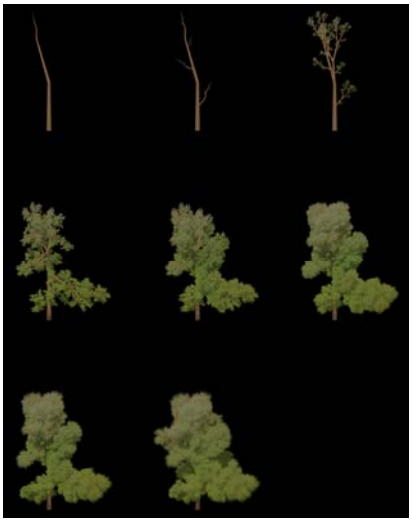

Figure 3: Increasing number of branches produces appearance of tree growth or foliage budding, In this sequence, the number of branches k is increased from 1 to 8. Note how the tree structure is the same, but the fullness of the foliage increases**.**

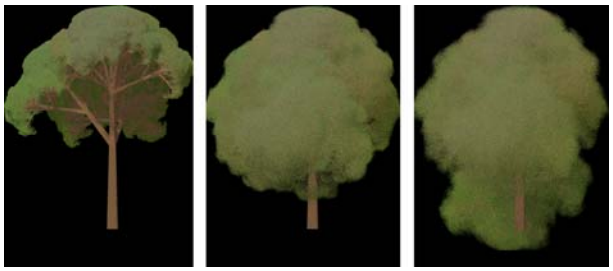

Figure 4: Increasing number of branches sometimes results in loss of detail and realism. In this image, the number of daughter branches is varied from $k = 5$ to $k = 7$.

Since the simulator is based on stochastic processes, we often get undesirable images. Excessive branches, and thus excessive foliage, can overshadow the actual branching structure of the tree, as we saw in Figure. In these cases, the images begin to take on the appearance of a cloud or ball of cotton, having a "puffy" quality. Additionally, the generated trees may sometimes be too sparse or too geometric in nature, revealing the mathematical underpinnings of the model generation algorithm used. In these cases, the resulting structures display too much precise self-similarity, which gives an unnatural appearance, as seen in the trees displayed in Figure 5. Fortunately, these cases are rather rare, being seen (empirically) in approximately 1% or less of all generated trees. Furthermore, adding additional branches to such trees will often change the structure of the tree enough to make it appear natural.

Figure 5: Generated trees that show too much self-similarity, revealing the mathematical underpinnings of the IFS algorithm.

Beyond the number of daughter branches, the simulation includes three global Boolean flags that affect the structures of generated trees.

The first of these parameters is *Use Branch Heights*. If this flag is set to true, then daughter branches will sprout at random points from a mother branch. In nature, this is the

case with many conifers, such as spruce varieties. If this flag is set to false, then daughter branches will all sprout from the apex of their mother branch. In nature, this is the case with many deciduous trees, such as elm and birch varieties. The primary effect of changing this flag is to create more "top-heavy" trees versus more irregular, branch-distributed trees. An example is displayed in Figure 6.

Figure 6: Different tree structures generated by changing *Use Branch Heights* flag. Left tree: flag off, right tree: flag on.

The next parameter is *Global Scaling*. If this flag is set to true, then all branches and foliage are scaled down based on their order and a global scaling factor (which is stored in the trunk branch). If set to false, then all branches are scaled based on a scale stored in each branch, but which is chosen randomly on each iteration. If set to true, this flag tends to make trees more regular, and "tighter" in structure. If set to false, the tree structures generated are more irregular, but this can also cause excessive "puffiness" of the tree foliage, making it again cloud-like. An example is displayed in Figure 7.

Figure 1: Different tree structures generated by changing *Global Scaling* flag. Left tree: flag off, right tree: flag on.

Finally, the third parameter is *Scale By Order*. If this flag is set to true, then branches and foliage decrease in scale more dramatically based on their order. This flag can be a bit unpredictable on the generated tree structure. However, in general, setting this flag to true will prevent trees from getting too "puffy" and cloud-like. Additionally, setting the flag to true will generally result in a slimmer, decreasing tree structure, like a willow bush, while setting to false will make trees more top-heavy and full, like many deciduous trees. Another interesting effect of this parameter is the apparent age of the tree. If set to true, the tree will often have the appearance of a younger seedling tree. Setting the flag to false *on the same tree* will generate an older looking tree, or a tree that appears in full bloom (like spring or summer) as opposed to barren (like fall or winter). An example is displayed in Figure 8.

Figure 2: Different tree structures generated by changing *Scale By Order* flag. Left tree: flag off, right tree: flag on.

There actually exists a fourth global parameter that controls the tree structures generated. This flag is *Use Twist*, and adds an extra element of random rotation to the branches. While this parameter does in fact change the tree structure, it mostly just randomizes the tree structure further, and does not have any real useful properties (such as changing the apparent age of the tree) other than that. As such, it is not considered a vital parameter.

Figure 3: Trees generated using the same local parameters, but with different states of the global flags. Letting the four global flags (*Use Branch Heights, Global Scaling, Scale By Order, and Use Twist*) form a binary string representing the flag states, from the top left to bottom right, the flags vary from 0000 to 1111.

The same tree has been used in the figures above to explain the apparent differences in the generated tree structures obtained by varying the global flags. The effects of the parameters in combination with each other are displayed in Figure 9. As can be seen in this figure, a wide variety of tree structures can be generated using the *exact* same local parameters, but with different states of the global flags. This is a very useful property of the simulator, and helps to generate trees with the same general structure, but with varying local structures within the overall global structure of the tree. This can, for example, be used to generate different instances of the same variety of tree, or younger and older versions of the same tree. A nice future extension to this software would be the ability to slightly change the stochastically generated values in order to produce even more instances of the same variety of tree, which show generally the same global structure, but with local differences that help define a single living tree entity.

4. Conclusions and Future Research

This paper has examined the theoretical background of fractals generated using L-systems and Iterated Function Systems. Using such formalisms, we can describe botanical forms, including the forms of many trees. While mathematical in nature, the tree forms generated using stochastic processes combined with fractal generation can be quite convincing, even approaching photo-realism. The simulation software written for this implementation provides a good starting model for generating believable tree structures using fractal geometry. The software utilizes stochastic processes to great effect, generating an incredible variety of convincing trees that have striking resemblance to living, breathing organic trees. While not all trees generated using the software are attractive and believable, nonetheless the fact that *any* trees generated by the software show that fractals are useful tools in describing the botanical form of trees.

5. References

[1] Barnsley, M. *Fractals Everywhere*. Academic Press, San Diego. 1993. Second edition.

[2] Bogomolny, A. *The Collage Theorem*. Cut the Knot website. Online at http://www.cut-the-knot.org/ctk/ifs.shtml. March, 1998.

[3] Dutil, N. *Construction of Fractal Objects with Iterated Function Systems.* McGill School of Computer Science website. Online at http://www.cs.mcgill.ca/~ndutil/project.pdf. 2000.

[4] Frame, M., Mandelbrot, B., and Neger, N. *Fractal Geometry.* Yale University website. Online at http://classes.yale.edu/fractals/. 2006.

[5] Green, E. *Fractals and Iterated Function Systems.* University of Wisconsin Computer Science website. Online at http://www.cs.wisc.edu/~ergreen/honors_thesis/contents.html. 1998.

[6] Hickerson, K. *Iterated Function Systems.* California Institute of Technology website. Online at http://www.ugcs.caltech.edu/~kevinh/iterated.html. Date unknown.

[7] Honda, H. *Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body.* Journal of Theoretical Biology, Issue 31. 1971.

[8] Lindenmayer, A. and Prusinkiewicz, P. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York. 1996.

[9] Prusinkiewicz, P. *et al*. *Algorithmic Botany: Publications*. Algorithmic Botany at the University of Calgary website. Online at http://algorithmicbotany.org/papers/. 2006.

[10] Reeves, W.T., and Blau, R. *Approximate and probabilistic algorithms for shading and rendering structured particle systems*. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985) in *Computer Graphics*, 19, 3 (July 1985). ACM SIGGRAPH, New York, 1985.

[11] Wolfram, S. *Fractals.* Wolfram Research Mathworld website. Online at http://mathworld.wolfram.com/Fractal.html. 2006.