Random Forests with Lindenmayer Systems

Project by Daniel Matheson, for PMATH 370: Chaos and Fractals

Table of Contents:

Chapter 1: Introduction

Scientists, Mathematicians, Religious Clergy, and Philosophers have always been drawn to the patterns found in nature. In some places there is marvelous symmetry, in others there is what appears to be absolute chaos. At times there are specific numbers which coincidentally – or perhaps for a reason – can be found in many places, such as the Fibonacci numbers or the golden ratio, phi – which we will make use of. We continue to struggle towards an understanding of these mysteries, and some learned people of the past have made some progress.

One such scientist was a man named Aristid Lindenmayer who developed a type of formal language that is now called Lindenmayer systems (L-systems for short). A formal language is a set of strings of symbols, which can be constrained or manipulated in some way by rules that are specific to the language. We will see more of this later.

Lindenmayer had originally used L-systems to model the behavior of plant cells, but in modern times they are typically used to model whole plants. [2]

In this paper we will be applying L-systems to their modern context of modelling entire plants. But first, we must introduce what L-systems are, and how they can be used to create such models.

Chapter 2: Basics of L-Systems

The most basic version of an L-system is a simple rewriting system, and it is the type we will be focusing on.

Definition 2.1:

We will be focusing entirely on context-free L-systems: that is, the iterations, and productions (see below) are independent.

An L-system consists of the following tuple: {V, w, P} where:

- V is the *alphabet* of the system: the symbols which can be manipulated in each iterative step. We let V* be the set of all words over V, and V+ be the set of all non-empty words.
- w is the *axiom*: the "seed" of the iteration. Where $w \in V+$
- P is the set of *productions,* which are the maps that form the L-system, much like an IFS. Specifically, a production (a, X) is a map $a \rightarrow X$, where $a \in V$ and $X \in V^*$. It is assumed that for any letter $a \in V$, there exists an $X \in V^*$ such that $a \rightarrow X$. This will not always be the case, when we use some letters in V as placeholders.

Before we continue, it is important to explain some common notation that we will be using for our alphabets and productions. Specifically, we will be using notation that refers to two-dimensional (and later three) movement. For this purpose, we define the following:

- The letter ‘F’ represents a command to move forward by a given distance, while drawing a line

- The letter ‘f’ represents a command to move forward by a given distance, without drawing a line

- The letter ‘+’ represents a command to turn right by a given angle.

- The letter ‘-‘ represents a command to turn left by a given angle.

Letter	**Command**
F	Forward
f	Forward, no line
+	Turn Right
-	Turn Left

The obvious question now arises: What are these “given distances” and “given angles”? Well, that depends on the L-system we are trying to draw. Let’s look at a well-known example, the Koch Curve:

Example 2.2: The Koch Curve with L-Systems

Alphabet: {‘F’, ‘+’, ‘-‘}

Axiom: ‘F’

Productions: ‘F’ $\rightarrow$ ‘F-F++F-F’

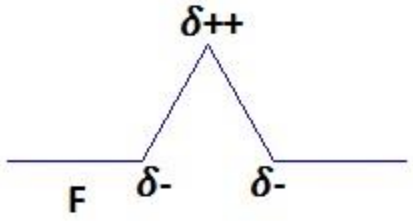
Distance: 1 Angle: 60° *(normally denoted as δ)*

Distance Factor: 1/3 *(explained shortly)*

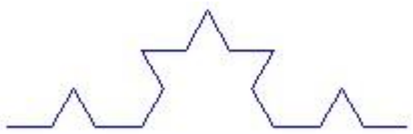
Clearly, the axiom is just a straight line, drawn by ‘F’, as shown below. Since no iteration has taken place, we call this the n = 0 case. The length of ‘F’ in the axiom is the distance given, 1.

Now when we run the production rule ‘F’ $\rightarrow$ ‘F-F++F-F’ on this axiom ‘F’, it is easy to see that it gets mapped to ‘F-F++F-F’, which represents the next figure below. The δ’s represent the angle turned at each point, to illustrate further what the system is doing. Only one ‘F’ has been shown for cleanliness, but all other drawn lines are also ‘F’. This is now the n = 1 case.

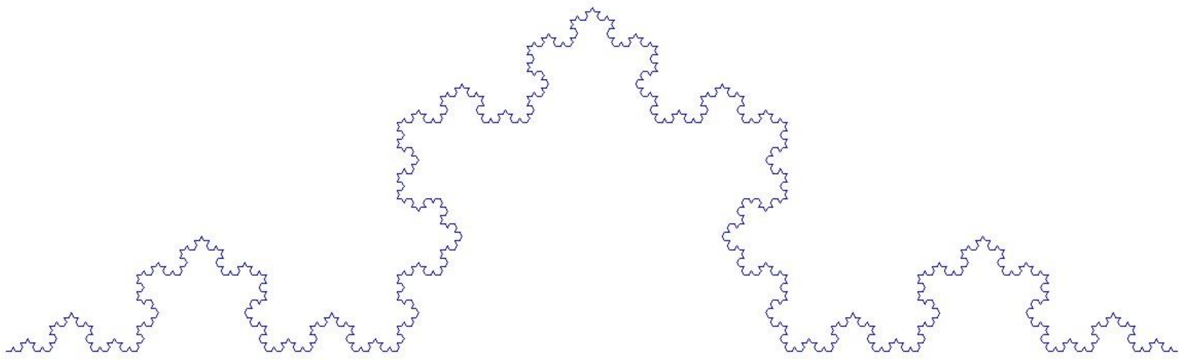
Note that 'F' now draws a distance of 1/3, rather than 1. This is due to the *Distance Factor* above. The distance travelled by 'F' is then $(1/3)^n$ where n is the number of iterations performed.



If we proceed to n = 2, we find that 'F-F++F-F' has become 'F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F' which is quite complex already! The following is the image generated by the n = 2 case:

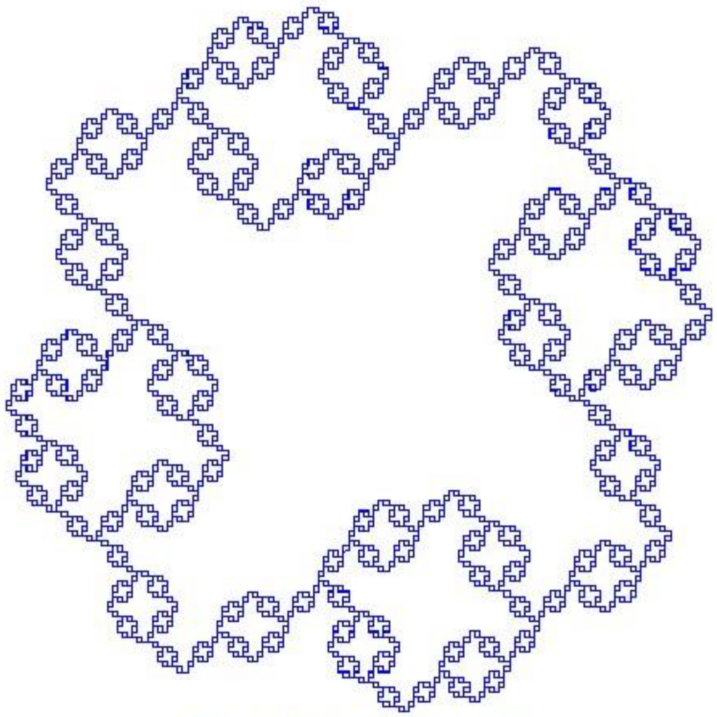


Finally, we can continue this process over and over again, until we run out of computational power or patience (which happens quite quickly). The following is the image for n = 5, clearly recognized as approaching the Koch Curve.

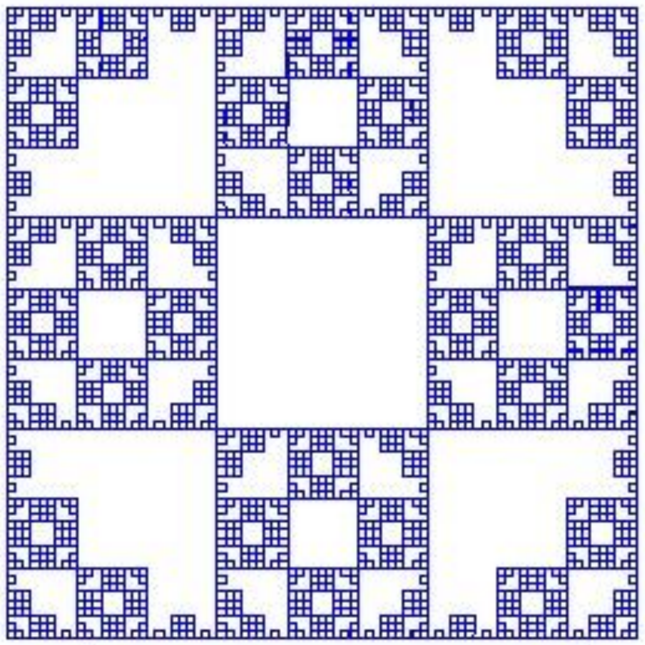


2.3 More Examples

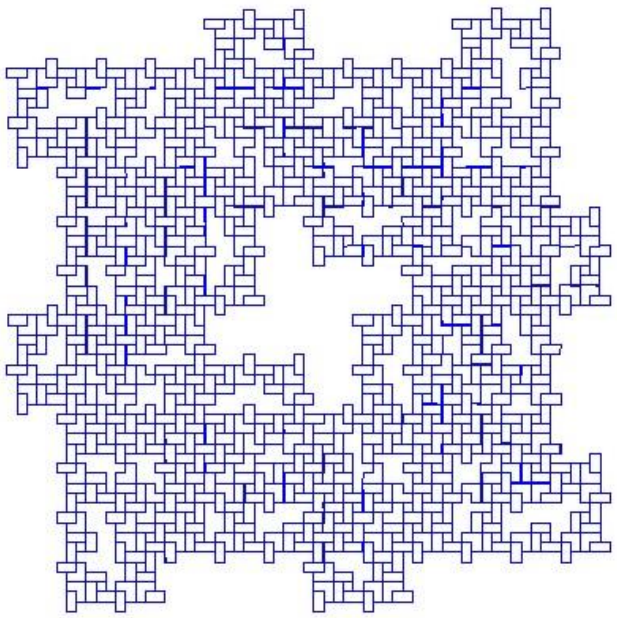
Given this simple example, with such a simple axiom and only one production, one can imagine that there are an immense number of complex objects that can be created in this way. And indeed there are. Here are a few examples, with the explanation of the system below the object. The first item is the axiom, then δ, then the re-write rule for 'F', and the number of iterations performed.



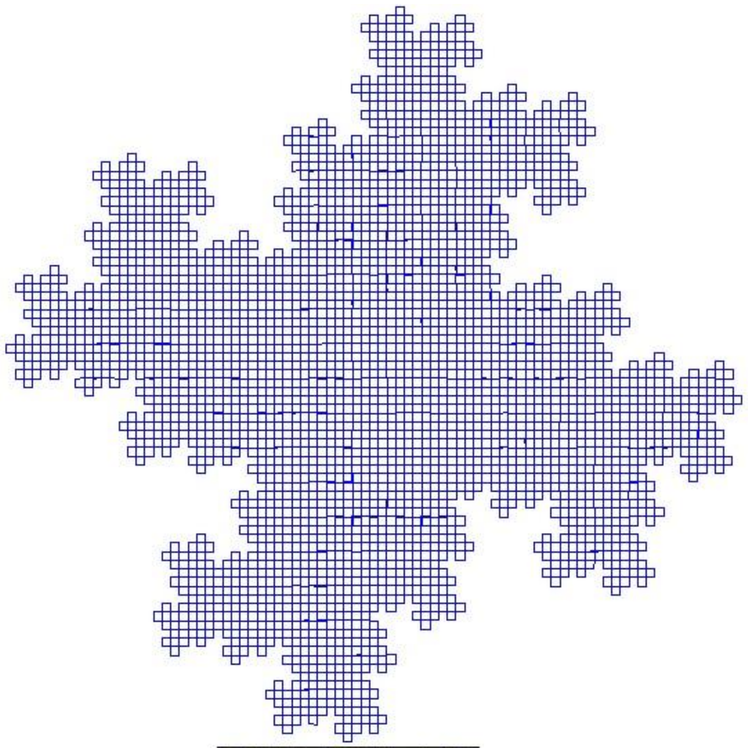
```
"F+F+F+F", 90, "FF+F+F+F+F+F-F", 4,
```



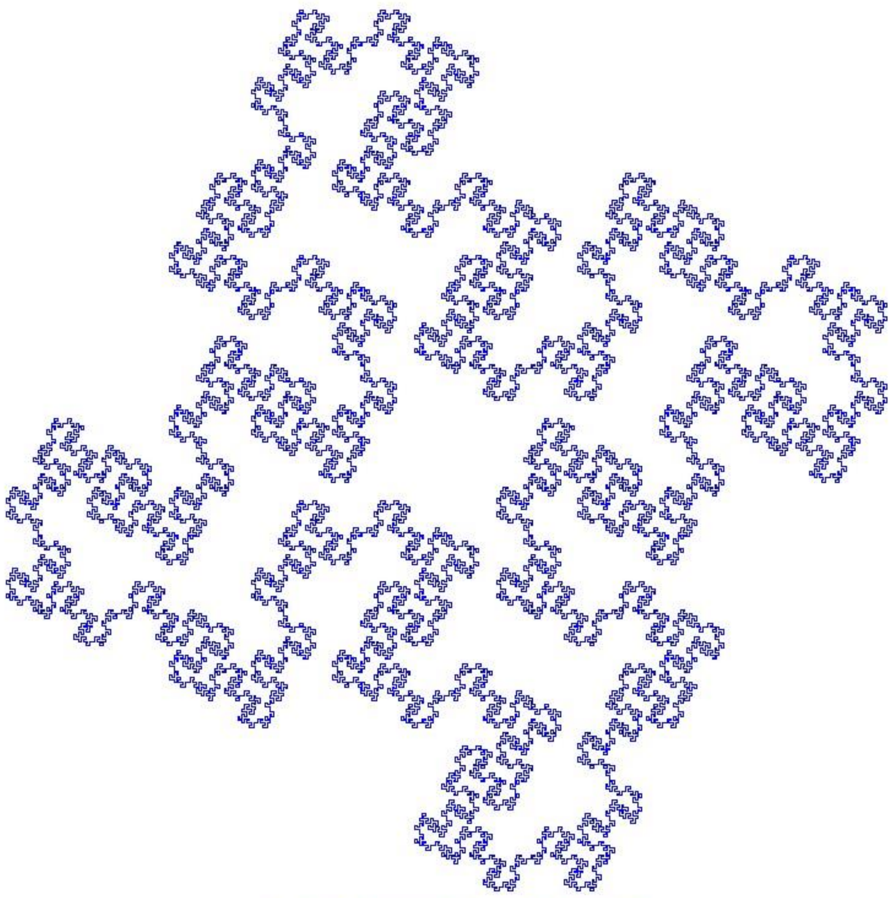
```
"F+F+F+F", 90, "FF+F+F+F+FF", 4
```

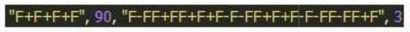


```
"F+F+F+F", 90, "FF+F-F+F+FF", 4,
```



```
"F+F+F+F", 90, "F+FF++F+F", 5,
```



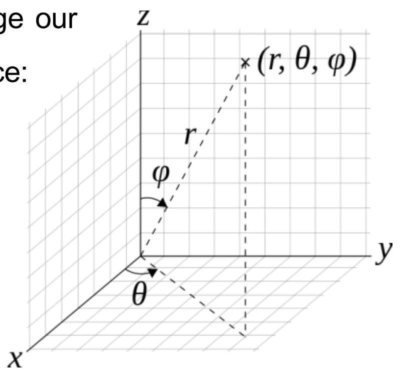


```
"F+F+F+F", 90, "F-FF+FF+F+F-F-FF+F+F-F-FF-FF+F", 3
```

The axioms in all of these systems was a simple square ("F+F+F+F"), and δ was $90°$, so we can see that with L-systems we can draw a very large number of objects. However, that is not the focus of this paper, so we continue along

Chapter 3: Lindenmayer Systems in 3 dimensions:

In order to create L-systems in 3 dimensions, we clearly must change our alphabet to another which represents movement in 3-dimensional space: {F, f, +, -} is no longer sufficient. We will instead be using a system analogous to spherical co-ordinates (shown in the figure to the right)



We also introduce the concept of bracketed expressions found either in the axiom or the productions.

3.1: Notation for 3D movement

- ‘F’ and ‘f’ remain the same.
- ‘+’ represents a counter-clockwise turn around the z-axis, or in terms of spherical co-ordinates; an increase in θ
- ‘-‘ is a clockwise turn around the z-axis, or decrease in θ
- ‘<’ represents an upward turn towards the z-axis, or a decrease in ϕ
- ‘>’ represents a downward turn away from the z-axis, or an increase in ϕ

F, f	no change
+	counter-clockwise turn along θ
-	clockwise turn along θ
<	counter-clockwise turn along ϕ (up)
>	clockwise turn along ϕ (down)

With this notation, and using the same mechanics as before, we can now create models of plants and trees. Some additional mechanics such as trunk/branch width, color, initial values of angles etc. can also be utilized for aesthetic purposes. We will see more of this later.

As before, then, we have the following parameters as input:

- A distance ‘d’, with distance factor ‘d_f’ (‘d’ is the ‘r’ from spherical co-ordinates)
- Two angles: θ and ϕ
- The axiom, productions, and number of iterations as in the 2-D case

3.2: Bracket notation and Placeholders

We now introduce the concept of bracket notation which will allow for the following:

1. Bracketing allows for different distances to be travelled in the same L-system by the command ‘F’. In the examples above, all ‘F’ were the same distance; this would make drawing a tree very difficult
2. Branching becomes more obvious in the productions, since we are dealing with trees after all
3. Bracketed L-systems more closely resemble a tree, and therefore make it easier for the person inputting the system to achieve the type of “look” they are going for with the tree
4. It makes the programming much easier

The essential function of brackets in L-systems is the same as in Mathematics: the commands within the brackets are done first, and then we exit the brackets to continue left-to-right. This will become clear in the next example.

We will be using placeholders in our modelling of plants because it allows for simpler re-writing operations. In this paper, there will only be one placeholder and it will be called ‘X’. However, you could have more than one.

A placeholder, quite simply, is what it sounds like: it is a member of the alphabet of the system whose production rule is to simply replace the ‘X’ with something else, but ‘X’ does not execute as a command of any sort (to move, or turn for example).

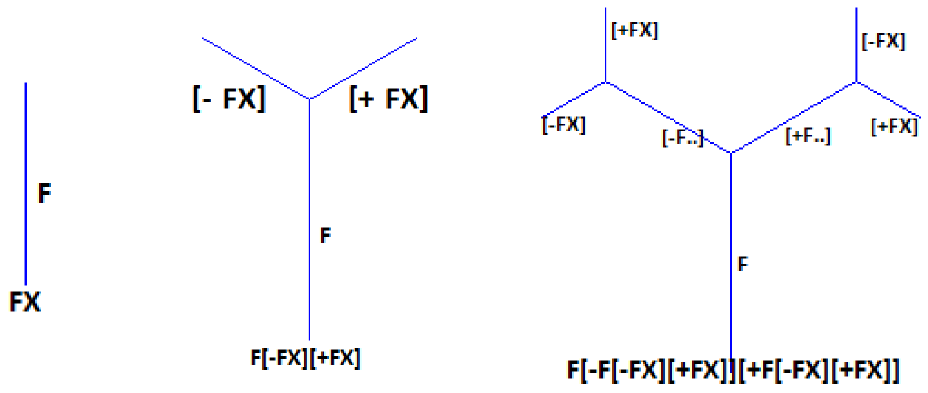
<u>Example 3.3</u>: Consider the following L-system (in only two dimensions, to explain brackets and placeholders)

Axiom: ‘FX’

Production: ‘X’ → ‘[-FX][+FX]’

d = 1, d_f = 0.6, δ = 60° and we allow the initial angle to be 90° (up)

This system produces the following: (n=0, n=1, and n=2 figures)



Hopefully it is clear from the figures what is happening. We start with ‘FX’, the axiom, in the first figure. As mentioned before, ‘X’ is a placeholder and does not execute a command, so it is ignored for now.

In the second figure, we have mapped ‘X’ $\rightarrow$ ‘[-FX][+FX]’. So now the entire L-system, as indicated at the bottom of the figure, is ‘F[-FX][+FX]’, mapped from ‘FX’. We will now break down, one letter at a time, how this L-system is executed:

F[-FX][+FX]

1. F : move forward by d
2. [: enter a new branch, with distance/length 0.6 (d * $(d\_f)^1$)
3. - : turn left by 60°
4. F : move forward (now with distance/length 0.6)
5. X : do nothing
6.] : close branch: return to where branch started, reset distance back to what it was before we entered the branch (1)
7. [: enter new branch, with distance/length 0.6 (d * $(d\_f)^1$)
8. + : turn right by 60°
9. F : move forward (distance/length 0.6)
10. X : do nothing
11.] : close branch, return to where branch started and reset distance to 1
12. L-system done.

If we were to complete this analysis for the third figure, we would find that as we enter a branch, we may enter another branch before closing it. In this case the distance travelled, for the second branch, then becomes 0.6^2 (d * $(d\_f)^2$). We can clearly generalize this to the *nth* branch of length (d * $(d\_f)^n$). This is one of the challenges computationally: to keep track of what the current distance is, and what it should return to once we encounter a ‘]’ in the L-system.

What we can also take away from this example is that the placeholder ‘X’ can be interpreted as the “tip of the branch”, where the next branches will “grow” when we iterate, assuming that our production is on ‘X’.

<u>Example 3.4</u>: Here is another, more complex example of a bracketed L-system using one placeholder, in three dimensions.

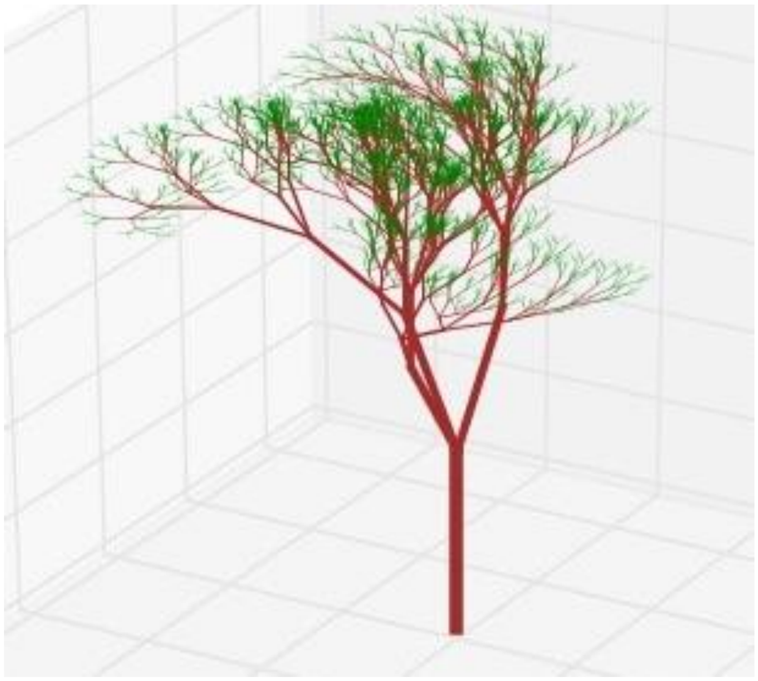
Axiom: ‘FX’

Production: ‘X’ $\rightarrow$ ‘[<--[FX]][<+++[FX]][>++[FX]]’

Note that from this production it is clear that each branch should split into 3, since there are three such bracketed expressions […][…][…]

d = 10, d_f = 0.8, θ = 15, ϕ = 10, and the number of iterations n = 8

We have drawn our first tree, which is great! But it looks a little too 'perfect', and a little awkward. In nature, it does not appear that plants are so deterministic and perfectly self-similar. So, to account for this, we will now add in elements of randomness to the system.



Chapter 4: Stochastic L-systems

4.1: What is a stochastic L-system?

Traditionally, statistical variance in L-systems is thought of as variance in productions. That is to say, rather than having a production like 'F' → 'F+F', we have the following as a production on 'F':

'F' → 'F+F' {0.33}

'F' → 'F-F' {0.5}

'F' → 'FF+F-' {0.17}

Where the numbers in the {} brackets represent the probability of the associated production rule being executed on 'F'. While this is a powerful tool, we have decided in this paper to take a more computationally efficient route, which also more closely resembles natural forms.

When we look at trees found in nature, it seems more typical that there is a slight variation in the *parameters* of the branches (angles, lengths) rather than variation in the actual *pattern* of the branching which appears consistent from trunk to the tips of the branches. At times, we can even notice similarities between the leaves and the tree itself (look closely at a maple tree next time you see one!). We show this self-similarity with the following example of the pine tree. The tree has a large number of branches, exiting the trunk radially at regular, small intervals (left). The pine tips are structured in the exact same way (right) but at a different scale. This self-similarity should be preserved in our models.

The traditional formulation of variance for L-systems above changes the patterns of branching, leaving the parameters (d, d_f, θ, and ϕ) constant. This does not resemble nature, in our view.

Therefore, in our models we will be deviating from Lindenmayer's ideas and introducing our own notion of variance.

4.2 Our alternate formulation of stochastic L-systems:

In order to closely model natural forms, we will not be manipulating the patterns of branching (i.e the productions), but the parameters associated with the model:

d: The distance travelled by ‘F’. Its initial value is the only relevant value, as the subsequent values in branches are determined by d_f (the distance factor)

This value will be randomized by a normal distribution, as heights in nature (among plants, and people) seem to follow the bell curve of a normal distribution. There is some debate about this, but it is at least *approximately* normally distributed. [3]

d_f: For our purposes this will remain constant, and will typically be the inverse of the golden ratio; often found to be the ‘beauty constant’ of nature.

θ and ϕ: These will also be approximated by appropriate normal distributions.

The reason for using normal distributions is that we want the plant to be self-similar with slight variations. So our standard deviations will be relatively small. Because any large variations (greater than three standard deviations) can make the model look very awkward, we eliminate these mathematically within the program. This is also done in order to eliminate any possibility of a negative starting distance for d which would make no sense.

Using this formulation of variance, we move on to our next example:

Example 4.3: In this example, we are modelling the following L-system:

Axiom: ‘FX’

Production:

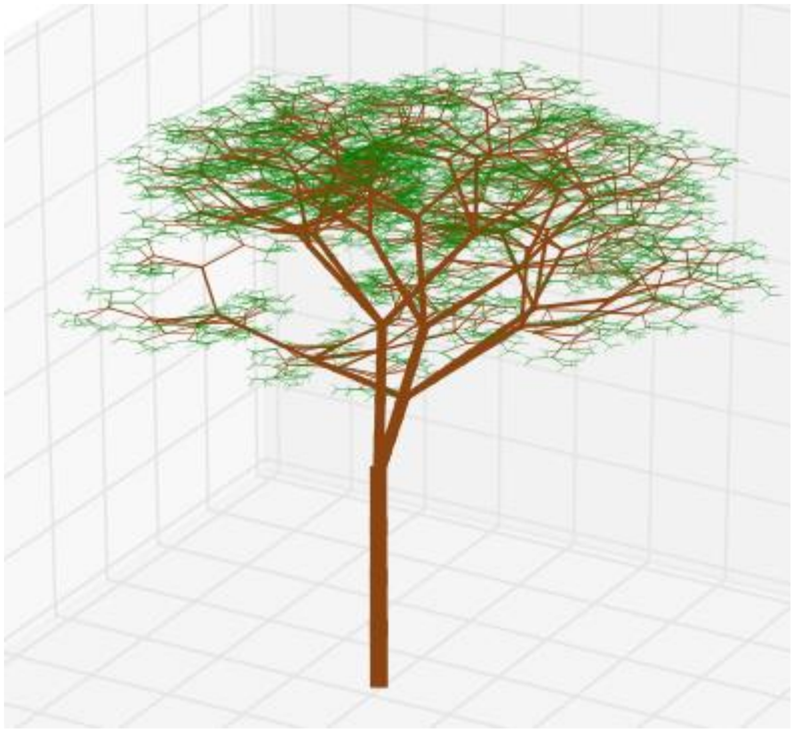
‘X’→‘[>---FX][>++++FX][>+++FX]’

d ~ N(10,2)

d_f = 1/g where g = (1+$\sqrt{5}$)/2, the golden ratio

θ ~ N(22,4), ϕ ~ N(12,2)

Note: N(x,y) denotes a Normal Distribution of mean x and standard deviation y



This model more closely resembles a real tree than the model in example 3.4, so we are making progress. Note that the branch widths get smaller as the branches get shorter as well (by a factor of d_f) and once the branch length is below a certain threshold, they are colored green. We will use this model of a tree for our final forest model along with some other plants:

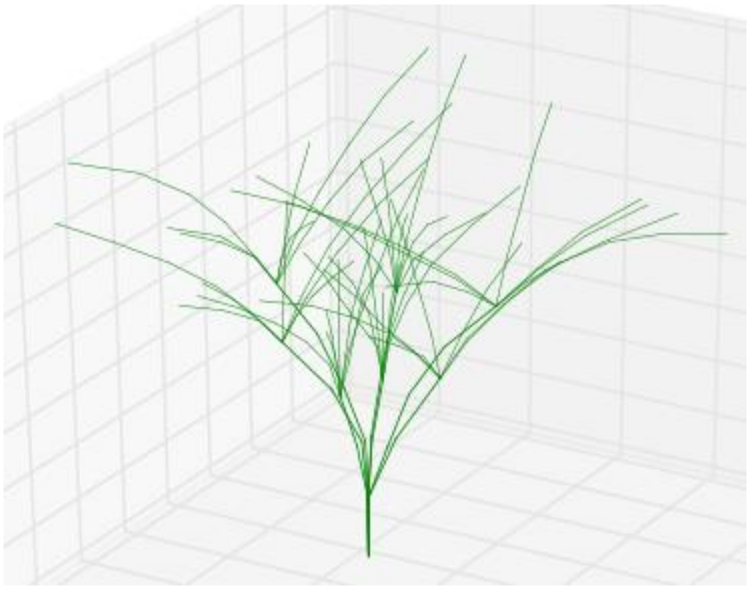
<u>Example 4.4</u>:

1. Small grasses: (shown below)

   Axiom: ‘X’

   Production: ‘X’ → ‘[F>F>F>F>FX]+[F>F>FX]++[F>F>F>FX]+++[F>F>F>F>FX]--+++[F>F>FX]++++[F>F>FX]+++++[F>F>F>FX]’

   d ~ N(1,0.2), d_f = 0.6, θ ~ N(35,5), ϕ ~ N(7,1)

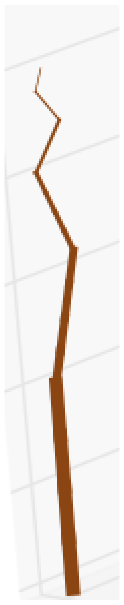


2. Broken Trees: (shown to the right)

   Axiom: ‘FX’

   Production: ‘X’ -> [>+FX]

   d ~ N(9,1), d_f = 1/g (golden ratio), θ ~ N(180,30), ϕ ~ N(8,0.5)



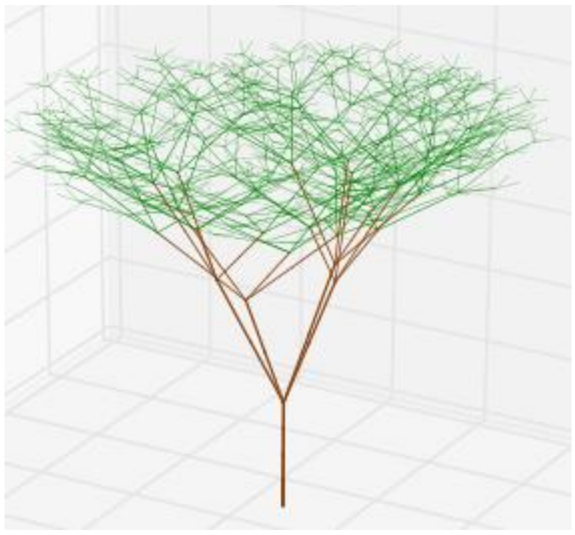
3. A second type of “bushier” and shorter tree:

Axiom: ‘X’

Production: ‘X’ →

‘F[>--FX][>+++FX][>----FX][>+++++FX][FX]’

d ~ N(4,1), d_f = 1/g, θ ~ N(20,3), ϕ ~ N(10,2)

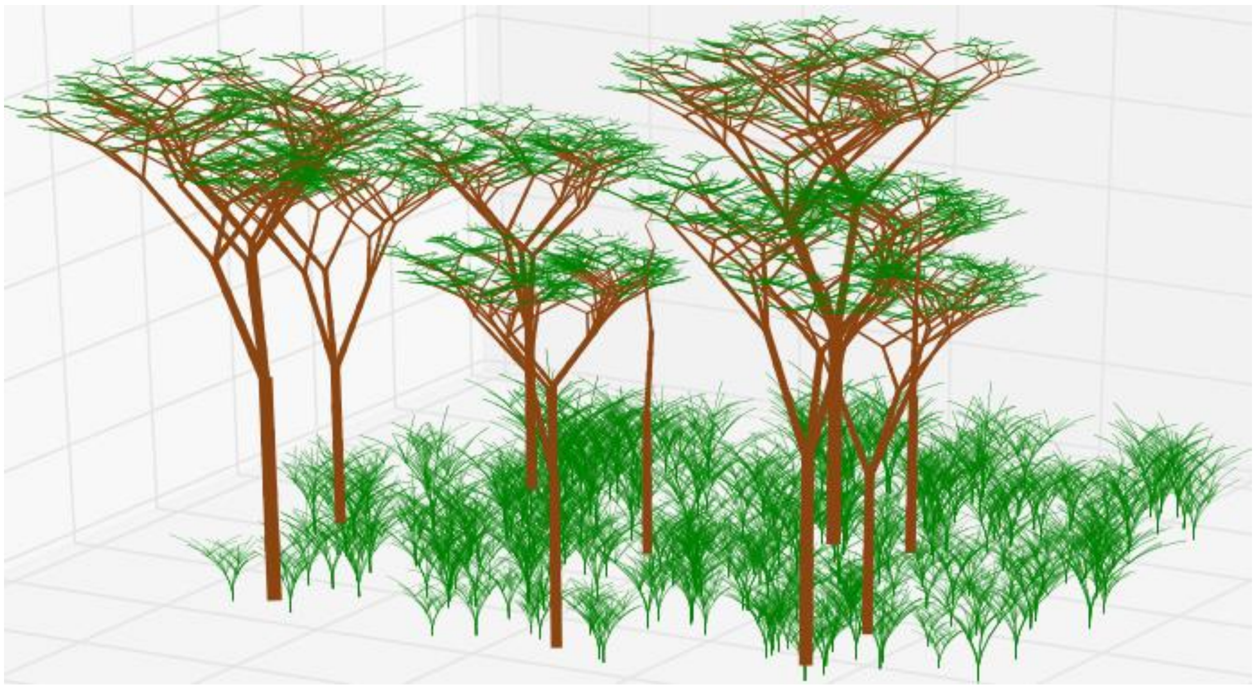


4.5: The forest model

Before we put all these plants together into a forest, we must discuss a few more things:

1. In a real forest, while there may be plants of the same type, they are facing in different directions. As such, we will also randomize the initial value of θ by using the Normal distribution N(180,60), but ensuring the values are positive.
2. The plants are positioned on the xy-plane, with no elevation. But if we were to simply place them on a square grid, it would not look real. Therefore, we will place the plants on a square grid, with the positions altered slightly in both the x and y directions by a factor between -30% and 30% of the size of the square that contains them.
3. The entire row/column of plants will also be moved slightly (-20% to 20%) in order that the forest does not appear as a square on the outside.
4. With all of this information on hand, we will generate the model with certain probabilities given to the trees. That is: we want 50% of the first type of tree, 40% of the shorter second type, and 10% broken trees. We also add in the grass on the entire forest floor (by running the program again on the same set of axes).

The result is the following:



As much as one would love to render a larger, more detailed forest… the computational power required is too great.

Chapter 5: Conclusion and Further Research

We have successfully created a relatively realistic model of a forest using several stochastic L-systems in one rendered image. Some further improvements could be made to the aesthetics, of course; and there is a significant amount of work being done on this by companies like Pixar. However, there is one significant mechanical aspect missing to our models: gravity.

In order to incorporate gravity, one would have to break each branch into small segments and measure the downward force being applied to each segment. This poses several challenges:

1. The issue of calculating the total mass of the branch which "hangs off of" (i.e comes after) the segment seems nearly impossible given the way the program was set up (reading one character at a time in the expanded L-system)
2. Gravity makes the trees "slump" more, meaning we would have to adjust our values for ϕ, and additionally: we would have to account for the cases where the branches start to point directly down to the ground. We now encounter the biggest issue in incorporating gravity: the tensile force of the branch and the force of gravity always

battling one another. Of course, this always applies, but it becomes more evident in the case where the branch is pointing directly down.

3. It would be extremely computationally intensive. The full forest model above already took *twenty minutes* to render without all of these extra computations. This clearly imposed some severe limitations already, as the model could only be rendered and modified three times per hour, at most.

There are many people in Biology, Computer Science & Mathematics, and in the Movie Industry who study these types of systems for creating large models of natural systems. They don't always model plants; sometimes it is mountains or water. L-systems are not particularly useful in the latter two models, but the ideas are very similar if we consider the L-system as akin to a type of Iterated Function System.

Specifically, there is a very large body of research being done in Canada at the University of Calgary on this subject, led by Professor Przemyslaw Prusinkiewicz, found at http://algorithmicbotany.org/papers/

Bibliography:

[1] Prusinkiewicz, Przemyslaw; Lindenmayer, Aristid (1990). *The Algorithmic Beauty of Plants.* Springer-Verlag.

[2] https://en.wikipedia.org/wiki/Aristid_Lindenmayer

[3] Mark F. Schilling, Ann E. Watkins, William Watkins. *Is Human Height Bimodal?* The American Statistician, Vol. 56, No. 3, (Aug., 2002), pp. 223-229
http://faculty.washington.edu/tamre/IsHumanHeightBimodal.pdf