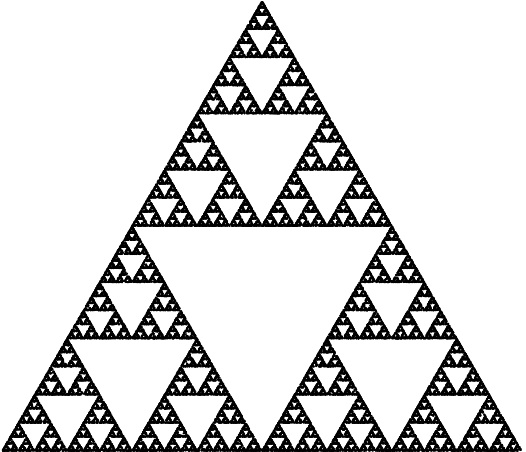
Fractal Trees

Gandhi Games

Fractals

A fractal is a pattern that repeats at different scales. We call these shapes “self-similar”. While we have only called fractals since the 1970s, the first description of a what we now know as ‘fractal patterns’ in nature came Leonardo da Vinci in the 15th century.



A self-similar/fractal shape.

Nature is full of fractal patterns including trees, lightning bolts, and river networks. Spirals such as hurricanes and even galaxies are also considered fractals.

Fractal Trees

A tree is approximately self-similar. Small parts of the tree look generally similar to the larger tree. A tree is formed by repeating a simple process. This is also the basic principle that is used when generating fractals programmatically. While the fractals generated can appear complex and detailed, they are created by repeating a number of simple steps.  
  
The algorithm for growing a tree is roughly like so:

1. The tree sprouts.
2. The sprout eventually splits into branches.
3. These branches themselves split into further branches.

At each step of this process it is as if two new smaller trees emerge. These smaller trees can be conceptualized as the trucks of a new generation of trees. This repetition of branching that forms the tree is also the cause for trees approximate self-similarity. In nature this process eventually stops and the end products are no longer fractals.

 Fractal Trees in Nature.

There are a number of different methods to generate fractal trees programmatically.

Lindenmayer Systems

Computers are important tools in the study of the structural patterns in natural and computer generated organisms. Lindenmayer systems, shortened to L-systems, (introduced in 1968) were one of the first (if not the first) use of computational power to study these patterns.

An L-system creates sets of strings based on a rule set. The system starts with an axiom (or seed), and then rules are recursively applied to the string to produce an output string. The output string can then be fed into other systems to produce graphical output.

For example:

Axiom: AB

Rules: A -> AB, B -> A

Gen 1: AB

Gen 2: ABA

Gen 3: ABAAB

Gen 4: ABAABABA

Gen 5: ABAABABAABAAB

Generation one starts with the axiom. In subsequent generations any instance of ‘A’ is replaced with ‘AB’ and ‘B’ with ‘A’.

By giving each character a specific action and implementing turtle graphics (<https://en.wikipedia.org/wiki/Turtle_graphics)> we can use this system to generate trees.

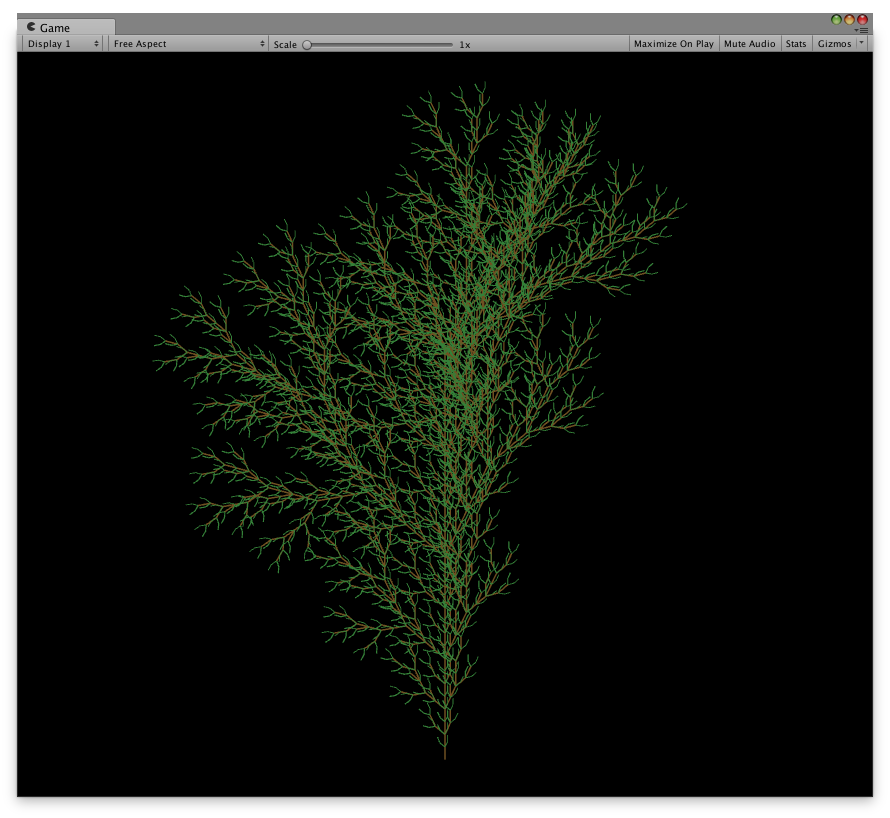
The system uses a simplified version of the d0L-system Grammar Description Language, shows in the table below.

|  |  |
| --- | --- |
| **Character** | **Rule** |
| C0, C1… Cn | Change all subsequent drawn lines to colour n. |
| F | Draw line in current direction. |
| - | Rotate direction left by n degrees. |
| + | Rotate direction right by n degrees. |
| [ | Store current state |
| ] | Restore saved state |

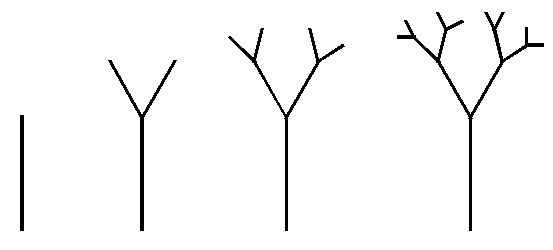
Axiom: FX

Rules: F -> C0FF-[C1-F+F]+[C2+F-F], X -> C0FF+[C1+F]+[C3-F]

Using those actions and the above axiom and rule set over five generations, you can create the tree shown in the image below.



Each generation produces a tree more complex than the last. An example of the first four generations of a similar rule set is shown below.



Space Colonization

It starts by defining the volume the crown of the tree will take. The simplest volume is a sphere, just a point and a radius. The volume is then filled with random points. You can think of these points like targets the colonization algorithm will try to reach.  
  
Then we add one segment at the base of the tree. From this point the tree will grow.  
  
A segment has two ends and some length. Soon you will find that the average segment length will be in part responsible of the overall appearance of the tree. Smaller segments will result in curvaceous and intricate trees while larger segments will make for straight trunks and branches.  
  
The two ends of the segment are of great importance too. For each segment end the algorithm will compute an attraction vector towards the cloud of target points. If there are target points close enough to the segment end, a new segment is added. The new segment will follow the same direction as the attraction vector at that point.

Whenever a segment end is too close to a target point, the target point is removed. As new segments are added in the direction of the target points, they end up eating all the points. Once there are no more points left, or they are too few of them, the algorithm is finished.   
  
The results are very realistic. Branches naturally avoid each other, each one appears to have developed as the result of seeking sunlight. The same method can be used to create roots. Roots also expand in some form of space colonization.

How many different trees can be achieved with this technique? Well there are many factors you can play with, like the size of the segments, the attractor vector cut-off zone and the distance where segments remove target points. On top of that you can introduce space warps that will mess up with attraction vectors. This can be used to simulate gravity for some heavy branches.  
  
As you can see, the algorithm is pretty simple, still the results are quite good. I think this beats L-Systems for large trees. Next, I plan to use it for generating the chaotic layouts of old cities. When I get there I will surely post about that.

<http://www.sea-of-memes.com/LetsCode26/LetsCode26.html>