Generating landscape art with deconstructed fractals

Eura Shin, University of Kentucky

Fractals are often valued for their beauty, both visually and mathematically, as well as their resemblance to objects found in the natural world. Because of their beauty and accessibility, fractal building algorithms can be used to create art. The perfectly symmetrical, self-repeating definition of fractals is opposite to that of the unpredictible elements found in nature. This project modifies basic iterated function systems (ITS) to more appropriately construct a natural looking landscape via mimmicking and randomization. We implement these methods of "deconstructing" ITS with fractal trees and horizons in a landscape generating program and propose further methods for different natural elements.

General Terms: Iterated function systems, Algorithms

ACM Reference Format:
Eura Shin, 2017. Generating landscape art with deconstructed fractals. *ACM Trans. Embedd. Comput. Syst.* 0, 0, Article 0 (0), 6 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Fractals, originally described by Benoit Mandelbrot in 1975, are structures that display self-similarity [3]. Popular examples of fractals include the Mandlebrot set (1) and the Sierpinski triangle (2). Note that because of the self-similar definition of fractals, recursion is a natural candidate when it comes to building these structures.

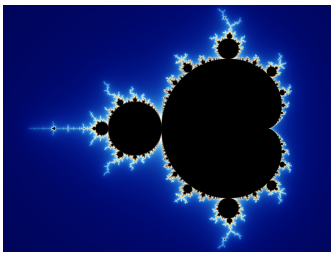

Fig. 1. A Mandelbrot set. [1]

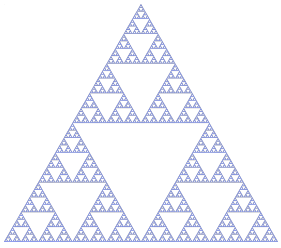

Fig. 2. A Sierpinski triangle. [2]

In this paper, we focus on generating two different natural elements, trees and horizons. Mathematically, these elements can be represented using fractals generated from an iterated function system and midpoint displacment algorithm, respectively.

Focusing on the tree in particular, this project proposes a method to deconstruct these algorithms, resulting in a more ”natural” visual effect:

(1) **Define** an appropriate fractal generating algorithm
(2) **Modify** this algorithm to mirror the behavior of the natural element
(3) **Randomize** variables within the algorithm.

2. SOLUTION

2.1. Defining the algorithms

2.1.1. Binary Trees. Iterated function systems (ITS) are one of the many methods used to recreate fractals. A certain class of fractals can be constructed using a finite set of mappings. Intuitively, this means applying a series of geometric transformations to a given system, each of which break it into smaller and smaller subsets. [4]

An incredibly basic implementation of a fractal tree is the Symmetric Binary Tree. A single iteration of the binary tree is a straight ”trunnk” that has split into two corresponsding ”branches”, each θ degrees from the linear extension of the trunk [8].

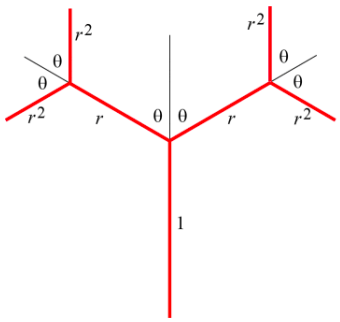

Fig. 3. Diagram explaining the mathematical layout of the binary tree.

In implementation, the length of the two branches are defined by a scalar constant ρ, the angle from the center line by θ, and the branching point (p_1, p_2) at the top of the trunk.

ALGORITHM 1: Binary Tree Algorithm

Input: Length l, length scalar ρ, angle θ, cummulative angle θ', and branching point p
repeat
  l *= ρ
  rightBranch = (p_0 + $l$cos($theta' - \theta$), p_0 + $l$sin($\theta' - \theta$)
  leftBranch = (p_0 + $l$cos($theta' + \theta$), p_0 + $l$sin($\theta' + \theta$)
  line(p, rightbranch)
  line(p, leftbranch)
  BinaryTree($l, \rho, \theta, \theta' - \theta$, rightBranch)
  BinaryTree($l, \rho, \theta, \theta' + \theta$, leftBranch)
until $l < 2$;

This algorithm results in the below trees created with varying values for $\theta = \pi/4$ and $\theta = \pi/8$

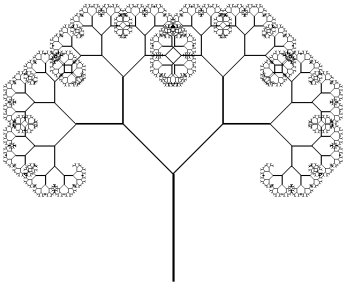

Fig. 4. $\theta = \pi/4$

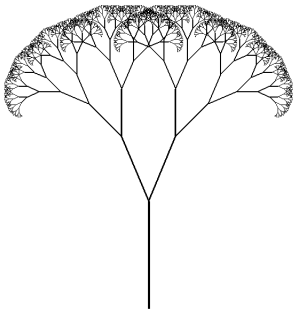

Fig. 5. $\theta = \pi/8$

2.1.2. Midpoint displacement algorithm. The midpoint displacement algorithm is a straight-forward method of generating randomized two-dimensional terrains. Because this algorithm already covers step 2 and 3 of the proposed method, it was not altered for this project. [5]

ALGORITHM 2: Midpoint Displacement Algorithm

Input: Displacement factor δ, endpoints p_1 and p_2
repeat
m = midpoint(p_1, p_2) m_2 = random(0,1)δ
line(p_1, m)
line(m + p_2)
MidpointDisplacement(δ, p_1, m)
MidpointDisplacement(δ, m, p_2)
until *Until fractal is sufficiently detailed*;

This results in a randomly generated terrain such as in the below figure

Fig. 6. Randomly generated terrain.

2.2. Mimmmicking nature

In order to alter the Binary Tree Algorithm, it should be noted that the resulting trees split into sub-branches only at the end of a given ”trunk.” In reality, trees will split along the length of a given branch. For this reason, we modify 1 to recurse multiple times along the given branch, instead of just once at the very end.

This results in the modifed algorithm in 3.

ALGORITHM 3: Modified Binary Tree Algorithm

Input: Length l, length scalar ρ, angle θ, cummulative angle θ', and branching point p
repeat
  for $branchPoint$ *in* ***number of branches*** **do**
    l *= ρ
    branch = ($branchPoint_0$ + $l$cos($theta' \pm \theta$), $branchPoint_0$ + $l$sin($\theta' \pm \theta$)
    line($branchPoint$, **branch**)
    BinaryTree2($l, \rho, \theta, \theta' \pm \theta$, **branch**)
  end
until $l < 2$;

2.3. Randomizing values

Finally, in order to replicate the ”randomness” that occurs in nature, many variables within 3 were randomized. These random values were chosen for the following constants and their respective ranges.

— $Numberof Branches$ = 0 to 5
— θ = π/4 to π/8
— ρ = 0.5 to 0.7
— $\pm\theta$ (The direction of branching) = 0 or 1

These ranges were chosen by method of trial and error. They were subjectively determined to result in the most tree-like and visually balanced images.

3. RESULTS AND CONCLUSION

Using the final algorithm in 3, along with randomized values, the following branched, flowering, and pine trees were implemented in a Python application.

Fig. 7. Branching, flowering, and pine tree generated by the app.

It should be noted the complexity of the Midpoint Displacement Algorithm 2 is $O(logn)$ because it finds find the midpoint and recursively calls itself, resulting "endpoint" distance to be halved each time.

The complexity of both the Binary Tree Algorithm and it's modified version (1, 3) is $(O(nlogn)$.

Note that this method of deconstructing fractal generating algorithms is incredibly scalable to whatever generation algorithm is required to create a given natural element. For example, the Koch snowflake is another IFS product that may be deconstructed to form a cloud element.

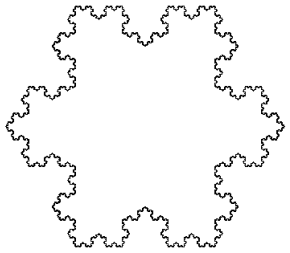

Fig. 8. The Koch snowflake.[6]

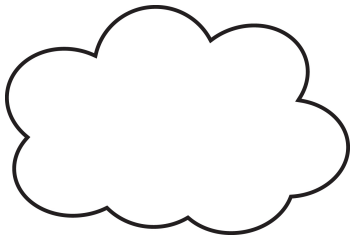

Fig. 9. A sample cloud. [7]

This method of generating elements to populate fractal art landscapes is adaptable in many ways and could result in incredibly life-like images if applied further.

4. REFERENCES

[1] Wikipedia. Mandelbrot Set. Retrieved December 12, 2017 from https://en.wikipedia.org/wiki/Mandelbrot_set

[2] Wikipedia. Sierpinski Triangle. Retrieved December 12, 2017 from https://en.wikipedia.org/wiki/Sierpinski_triangle

[3] Weisstein, Eric. Fractal. Retrieved December 12, 2017 from http://mathworld.wolfram.com/Fractal.html

[4] Bradley, Larry. 2010. IFS. (June 2010). Retrieved December 12, 2017 from http://www.stsci.edu/ lbradley/seminar/ifs.html

[5] Bites of Code. 2016. Landscape generation using midpoint displacement. (December 2016). Retrieved December 12, 2017 from https://bitesofcode.wordpress.com/2016/12/23/landscape-generation-using-midpoint-displacement/

[6] Wikipedia. Koch Snowflake. Retrieved December 12, 2017 from https://commons.wikimedia.org/wiki/Koch_snowflake

[7] Clip Art Panda. Clip Art. Retrieved December 12, 2017 from http://www.clipartpanda.com/categories/cloud-clip-art-outline

[8] Riddle, Larry. 2017. Symmetric Binary Tree. (October 2017). Retrieved December 12. 2017 from http://ecademy.agnesscott.edu/ lriddle/ifs/pythagorean/symbinarytree.htm