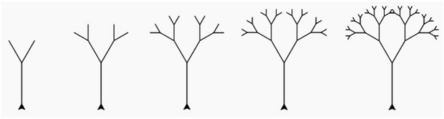
Yuhan Liu

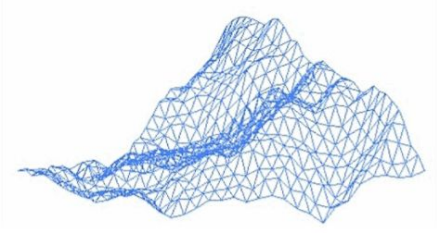
Algorithms and Data Structures Semester Project: **Fractal Mountains**

Introduction:

In this exploration, I generated two different fractals in an attempt to graphically illustrate landscape scenes. Fractals are essentially defined by characteristics of self-similarity and complexity. Fractals are defined as fragmented geometric shapes, of which have forms that repeat upon magnification. Fractal geometry explains the formation of naturally occurring patterns, as commonly seen in snowflakes, ferns, shells, the bronchioles of a lung, etc. Fractals are commonly applied in modern computer graphics to depict natural shapes in areas of CGI, video games (generating backgrounds and environments), topography (mapping surfaces of land), and scientific research (analyzing biological processes and growth of bacteria and related organisms). Recursion is the technique of dividing a problem into subproblems, often involving a function that repeatedly calls itself. Because of their self-similar nature, recursion can be used to generate fractals.



Beyond illustrating perfectly self-similar patterns, fractals can be randomized to form structures with patterns that are seemingly irregular. This type is known as statistical self-similarity, as the fractal's shape is still preserved even through random repetition. The algorithms used to generate such designs are known as stochastic algorithms because they use random probability distributions to imitate natural terrain. This project will explore the random recursive generation of two-dimensional mountainous structures, focusing on the twisted sierpinski and midpoint algorithms while utilizing the random and turtle graphics libraries.



Program:

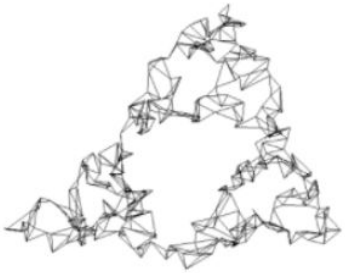
Twisted Sierpinski:

The Sierpinski triangle is a three-way recursive algorithm that divides a triangle into four new triangles indefinitely. The textbook provides a Sierpinski algorithm that divides an equilateral triangle by traversing from the lower left to the top. The algorithm works by connecting the midpoints of a triangle to subdivide it.

The twisted Sierpinski triangle is a modification of the original. While the number and relative placement of the triangles, as dictated by the degree variable, remains constant, the twisted Sierpinski uses randomization to shift the midpoint of the inner triangles. Despite shifting vertices, the algorithm maintains those new points for adjacent triangles.

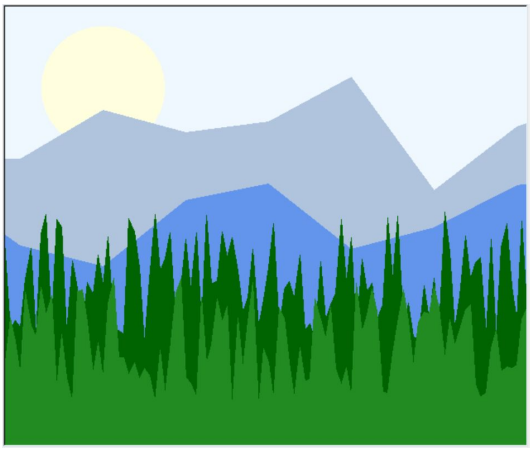
Degree: 4

Degree: 5



Degree: 6

Midpoint Landscape:



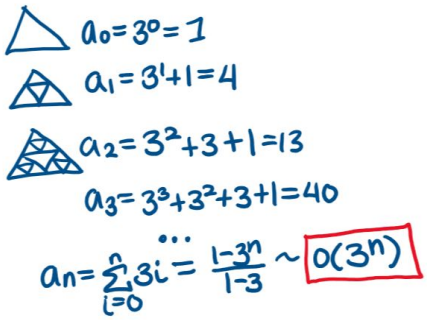
The Midpoint Landscape algorithm takes a similar approach to generate a randomized landscape scene. The algorithm begins with a straight horizontal line that is recursively divided into midpoints depending on the degree variable. The midpoints become x-coordinates that are paired with a random y-coordinate, creating a line with randomized offsets. While this method of creating mountainous fractals, also known as the midpoint displacement method, was first introduced by twentieth century mathematician Benoit Mandelbrot (the same mathematician who coined the term ‘fractal’), this version of the algorithm is original.

Note about Turtle Library: To avoid terminator errors caused by the external window, the window must be closed (window closes upon clicking) before running the program again. In addition, the program might need to be run twice to restart the window.

Asymptotic Evaluation of Runtime:

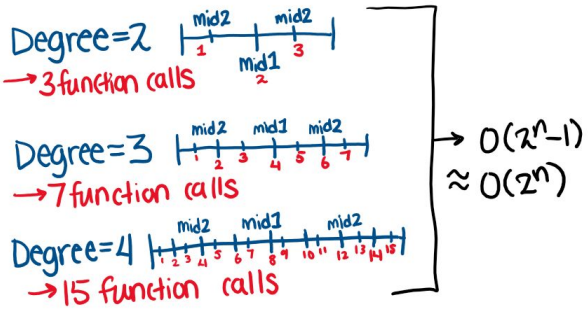
Twisted Sierpinski:

The Big-O complexity of this algorithm comes down to O(3^n), or exponential time, because the recursive function must be called for each triangle. Abiding to the design of the Sierpinski triangle, the algorithm must draw one and three times more triangles (thus creating the triangle number pattern 1, 4, 13, 40, etc) with each increment of the degree variable.



Midpoint Landscape:

The Big-O complexity of this algorithm comes down the exponential time of O(2^n) because the recursive algorithm must call itself twice until it has split the line into 2*degree partitions (see below).



Because of the exponential complexity of these algorithms, the graphics are difficult to generate when the degree variable exceeds a certain threshold. Thus, the next section will analyze implementations of these algorithms that are more efficient, but exceed the scope of the project.

Other Examples:

Twisted Sierpinski:

The efficiency of the Twisted Sierpinski algorithm depends on the implementation of the standard, equilateral triangle. This program modifies the version of the Sierpinski that completes triangles in order of placement (bottom left to top). Although the algorithm does not draw the larger triangles (as mentioned above, only the inner triangles are drawn), it must traverse over them in order to calculate the placement of the next vertex (for this reason, the turtle seems to rapidly traverse the board before beginning to draw upon running the program). If the program were modified to precalculate all vertices before drawing, the runtime would decrease because the turtle has less distance to cover. However, this will not change the overall efficiency of the Sierpinski algorithm as the number of triangles increases sequentially by its inherent design.

Midpoint Landscape:

Algorithms for one-dimensional midpoint displacement in Python utilize various functions to increase the efficiency at which the algorithm draws the segment.

One of these techniques is min-max normalization, which converts all the values in a dataset, or array, to fit a certain range. Min-max normalization can be conducted by using the min and max NumPy (Python library for linear algebra) functions which are used to transform vectors. Normalizing the array of points before drawing is useful for converting the points' values so that they fit on the image generator's canvas. However, this necessitates the y-values of the points to be calculated and transformed. In the original midpoint landscape algorithm, the randomized y-values are not calculated until the program is about to begin the drawing. By normalizing the y-values, more memory is required to record the transformations.

Another way of increasing algorithm efficiency is to decrease the number of recursive calls by implementing a function that predicts the midpoints in a sequence. The (2^n)+1 property states that the midpoint between 1 and (2^n)+1 equals (2^(n-1))+1. By combining this function

with a deque, the midpoints of a given range can be calculated by inserting values to the left and right side of the deque. Thus, recursion is no longer required for simple segments.

When these techniques combine, the resulting algorithm can have a linear asymptotic runtime because one function call will generate the entire segment as opposed to using recursion. However, this algorithm may require a larger memory to record the results of additional functions. This algorithm is found under the "64bitdragon" link in References.

References:

Aschenbach, N. (2014). 2D Fractal Terrain Generation. Retrieved December 08, 2020, from http://nick-aschenbach.github.io/blog/2014/07/06/2d-fractal-terrain/

Boyadzhiev, I. (2016, November 16). Twisted Sierpinski Triangle. Retrieved December 08, 2020, from https://www.geogebra.org/m/WEU9sRGA

Junge, K. (2017). How can naturally occurring mountain terrains be modeled using a recursive subdivision process and 3D visualization systems? Retrieved December 08, 2020, from http://mrbertman.com/EE/recursion.pdf

Landscape generation using midpoint displacement. (2020, February 15). Retrieved December 17, 2020, from https://bitesofcode.wordpress.com/2016/12/23/landscape-generation-using-midpoint-displacement/

Miller, B. N., & Ranum, D. L. (2011). Problem Solving with Algorithms and Data Structures using Python. Retrieved December 08, 2020, from https://runestone.academy/runestone/books/published/pythonds/index.html

The Diamond Square Algorithm. (n.d.). Retrieved December 17, 2020, from https://learn.64bitdragon.com/articles/computer-science/procedural-generation/the-diamond-square-algorithm

Wyvill, B., & Dodgson, N. A. (2010). Recursive scene graphs for art and design. Retrieved December 08, 2020, from http://webhome.cs.uvic.ca/~blob/publications/rec16.pdf