Implementation of Deterministic and Stochastic L-Systems

Duncan Grubbs, 20.2.2020, Procedural Content Generation

Introduction

This lab contains a basic implementation of a Deterministic L-System, a Stochastic L-System, and a L-System Interpreter. Along with this, there is a demo class that contains functions to generate some example L-Systems and run them through the interpreter. Specifically, the focus is on generating plant-like structures with a modified implementation of the Turtle graphics library.

Description

The basic idea behind an L-System is a formal grammar used to draw natural looking shapes in procedural content generation. This could be vegetation, roads, and much more. The algorithm works by beginning with an “axiom” character, and then iteratively applying the formal grammar rules specified by the user. These rules transform a single character into either another character or a string. One example could be ‘a ‘-> ‘b’, and ‘b’ -> ‘ab’. The user has control over how many times the string is transformed, and thus has control over the final length of the generated string. In this case, the string is then interpreted by a graphical interpreter. This interpreter iterates character by character through the string, applying certain draw commands based on the character in the string. For example, ‘F’ will draw in the “forward” direction by an amount, x, specified by the user. The user also can specify an angle, alpha, which ‘-‘, ‘+’ will rotate the drawing head by. ‘[‘ and ‘]’ control a stack that stores past locations of the drawing head. The ‘[‘ character tells the interpreter to push the current location of the drawing head on to the stack, and the ‘]’ character pops the stack, moving the drawing head back to location that was popped off the stack. These five operations allow for powerful grammars that can draw almost any shape one could wish for.

On top of this basic L-System implementation, I also implemented a Stochastic L-System[1], in which there are multiple rules applying to the same character, for example the rules ***'F' -> 'FF-'*** and ***'F' -> 'FF+'***. Each one of these rules is assigned a probability by the user. When the algorithm encounters a character that has many possible rules that could be applied, it chooses randomly between them accounting for each weighted probability. This behavior allows for pseudo random generation each time the algorithm is run, and therefore very unique shapes.

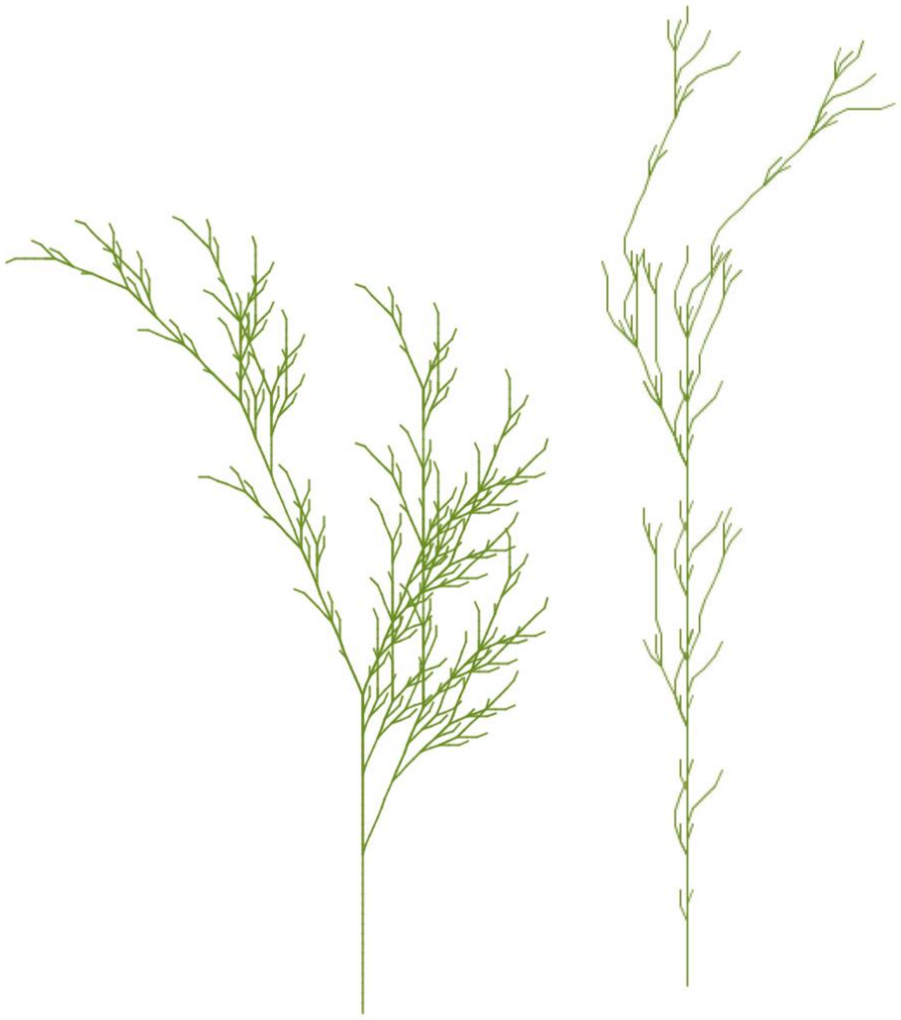
Analysis

While basic, this L-System implementation has the power to scale. Because the user has control over the rules of the system, the complexity of the generated string is completely under their control. The main limitation of the system is the interpreter. One factor of this limitation is the Turtle graphics library, which while easy to use, is very basic. Another factor is that the interpreter itself only recognizes the five operations specified above. These factors limit the program to 2D graphics and L-Systems that have a very basic alphabet (five operations). The interpreter can parse the alphabet: "[]F+-". In a true production scenario, there would be *many* more characters in the alphabet, specifying 3D rotations, color gradient changes, and much more. In the future, I would like to extend the functionality of the interpreter to recognize more complex commands such as color switching and fills.

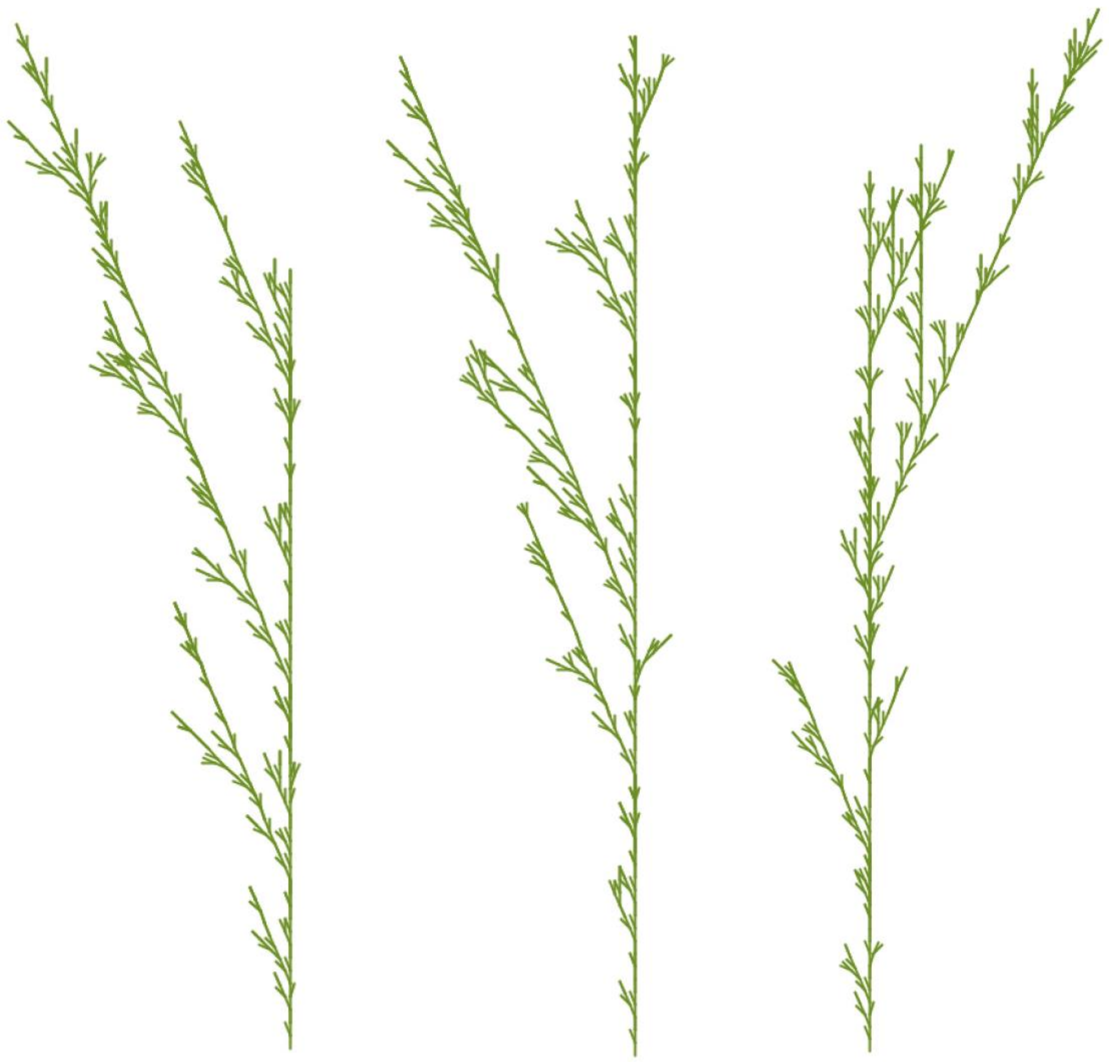
[1] https://www.csee.umbc.edu/~ebert/693/TLin/node18.html

Results

Below are examples of plant-like structures generated with the L-System. All were generated with five iterations and an angle of 22.5°.



*A simple two rule deterministic L-System, **Tree 1** (left), three rule deterministic L-System, **Tree 2** (right)*


*(Above) All three images were generated from **Tree 3**, a three-rule stochastic L-System*