CS 354 Final Project Report

Mrityunjay Mishra

May 7, 2021

Overview

For my final project, I have implemented an L-systems parser. This parser has the ability to generate and interpret/parse D0L-systems under the turtle interpretation. To test my parser, I created a driver file that renders a 2D forest scene with seasons. Please note that this project has only been tested in Ubuntu 18.04. However, it should still work under other operating systems and architectures.

Key Features

This project implments a parser for generating and interpreting sentences of a D0L-System. As such, the following features have been implmented

- Support for 12 alphabets, but more can be incorporated
- Support for parameterization
- Context-free parser
- Based on the “turtle” interpretation as defined by Regina et. al.
- Bresenham’s algorithm for line drawing with the turtle

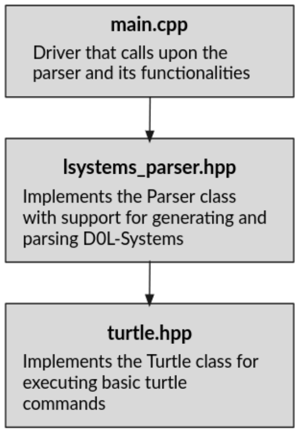
Limitations

By no means is this a full fledged L-Systems parser with all possible capabilities. As such, the following are some key limitations of this system. Please note that they are limitations not because I did not have time to implement them, but because they were never a part of my plan to begin with. These limitations are simply outlined for the understanding of the user, so they may know what they can and cannot do with this parser.

- Active support for 1L, 2L, and pL-Systems in general is not provided. However, building these features in should be relatively easy with how the project is set up (essentially, one would only need to define a couple extra rules/conditionals during the parsing of the setence itself).

- No support for context-based grammar
- One cannot interpret the strings based on chain-coding
- The movement of the turtle itself is currently limited to 2D space. However, extending its movement to 3D space can be done with relative ease

Architecture and Functionalities


main.cpp
Driver that calls upon the parser and its functionalities
lsystems_parser.hpp
Implements the Parser class with support for generating and parsing D0L-Systems
turtle.hpp
Implements the Turtle class for executing basic turtle commands

main.cpp - The driver file that calls upon the parser in order to render a 2D forest scene with seasons using D0L-Systems. The basic outline for this file is as follows:

```
MAIN.CPP

Define necessary functions
Set up GLFW context
Set up other OpenGL requirements
Generate grass
if grass is grown enough
  Generate shrubs
  Generate tree based on season
  if tree is grown enough  then grow leaves based on season
```

lsystems_parser.hpp - Implements the *Parser* class, which can generate string for, and interpret D0L-Systems with parameterization (support for other pL-Systems have not been added). In this parser, paramterization is primarily done by writing parameters directly into the generated string (however, parameters can be encoded in other ways as well with relative ease) It should be noted that this parser assumes context-free grammar. The D0L-System sentences are interpreted under the turtle interpretation as defined by Regina et. al. This parser has support for 12 alphabets (though support for more can be added with relative ease), and they are

- F : move forward (while drawing a line)
- - : turn right
- + : turn left
- [: push / save configuration of the turtle
-] : pop / restore configuration of the turtle
- X : placeholder alphabet
- (: begin paramter for setting thickness
-) : end parameter for setting thickness
- : begin parameter for setting length
- : end parameter for setting length
- $: begin and end parameters for setting rotation (in radians)
- | : rotate turtle by $\pi/2$

A basic outline of *Parser* class is as follows:

PARSER

private : define class variables
public :
Parser() ...
setConfigs() - for setting the configurations of the turtle
parse(std::string sentence) - interprets the input sentence alphabet by alphabet and manipulates the turtle accordingly
kochCurveStringGenerator() - generates a Koch curve, for testing only
grassGenerator() - generates a sentence for a grass based on its production rules and depth
shrubGenerator() - generates a sentence for a shrub based on its production rules and depth
treeGenerator() - generates a sentence for a tree based on its production rules and depth. Note that tree generation is parameterized based on specified parameter alphabets. Refer to source code for more detials.
pentaplexityFlowerGenerator() - generates a sentence for a pentaplexity curve based on its production rules and depth; used as a flower for the Cherryblossom tree
squareSierpinskiLeafGenerator() - generates a sentence for a Sierpinski Square based on its production rules and depth; used as a leaf for the Cherryblossom tree

---

In addition, this file also defines a *Config* struct to represent a certain configuration of the turtle.

```
    struct Config {
    int length;          //the amount to move forward by
    float turnAngle;     //the amount to turn by
    float currentAngle; //the current rotation angle of the turtle
    int x, y;            //the current position (x,y) of the turtle
    int thickness;       //thickness of the line drawn by the turtle

    Config() {}

    Config(int len, float tAngle, float cAngle, int x_, int y_, int thick) {
        length = len;
        turnAngle = tAngle;
        currentAngle = cAngle;
        x = x_;
        y = y_;
        thickness = thick;
    }
  };
```

turtle.hpp - Defines the *Turtle* object. It should be noted that the interpretations of the turtle's movement are limited to 2D space in this implementation. However, this can be extended to 3D space with relative ease. Also, note that Bresenham's algorithm was used for line drawing with *Turtle* from https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm. A basic outline of the *Turtle* class is as follows

---

```
TURTLE

private : define class variables
public :
  Turtle() ...
  define setLength()
  define getLength()
  define setRotationAngle()
  define getRotationAngle()
  define setCurrentRotation()
  define getCurrentRotation()
  define turnRight()
  define turnLeft()
  define turnBackwards()
  define setThickness()
  define getThickness()
  define getPosX()
  define getPostY()
  define setColor()
  define goTo()
  define move()
  define bresenhamLine()
  define bresenhamLineLow()
  define bresenhamLineHight()
  define plotPoint()
```

Final Product

For my final product, I rendered a forest scene through only D0L-Systems. These systems were generated, and then interpreted, by the parser that I implemented. In addition to the forest scene, I also added the notion of seasons; and I also added the notion of time. Essentially, the scene starts out with nothing, and then shows that grasses, bushes, and eventually, a Cherry-blossom tree grows as time passes. For a full demonstration, please refer to the link below:

https://youtu.be/xnu3cjfSvT8

Screenshots

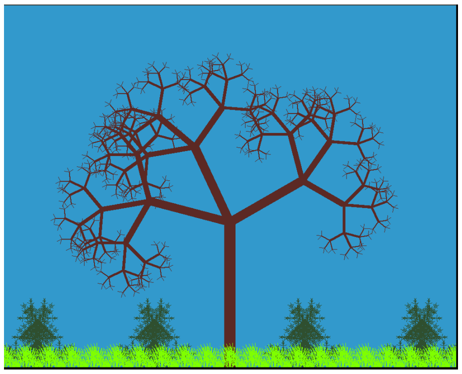


Figure 1: We see that the forest in now starting to grow a bit more

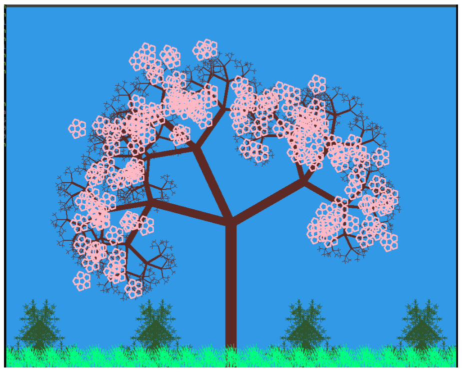


Figure 2: Now, we see a fully grown forest with a Cherry-blossom tree in spring season. Please note that I have used pentaplexity curve for the flowers of this Cherry-blossom tree.

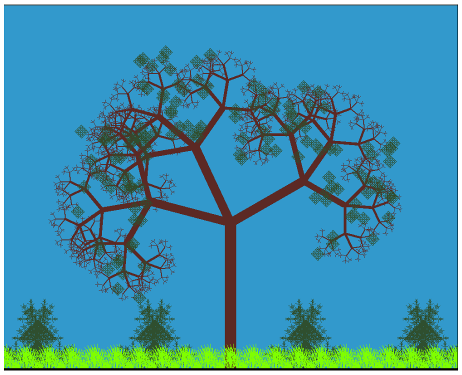


Figure 3: Then, the season changes to summer. Please note that I have used a Sierpienski Square to generate the leaves for this tree during both the summer and fall seasons.

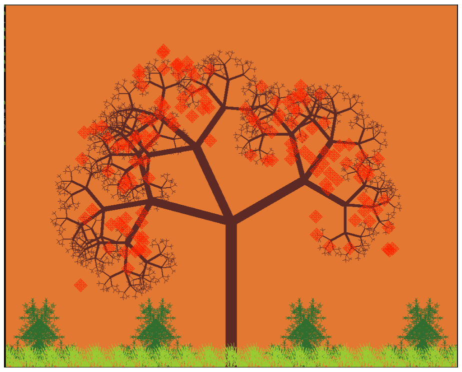


Figure 4: Then, we get fall season.

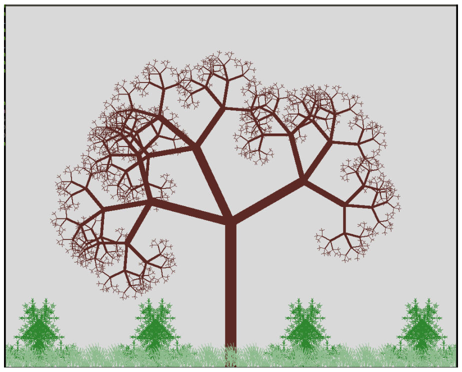


Figure 5: Finally, we get winter, after which we would cycle back to spring.

References

http://algorithmicbotany.org/papers/graphical.gi86.pdf
https://www.cs.utexas.edu/ theshark/courses/cs354/lectures/cs354-22.pdf
http://algorithmicbotany.org/papers/sigcourse.2003/2-1-lsystems.pdf
http://algorithmicbotany.org/papers/sigcourse.2003/2-1-lsystems.pdf
http://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf
http://paulbourke.net/fractals/lsys/
https://en.wikipedia.org/wiki/Space-filling_curve
http://algorithmicbotany.org/papers/graphical.gi86.pdf
https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
https://en.wikipedia.org/wiki/Line_drawing_algorithm
https://www.nodebox.net/code/index.php/Mark_Meyer_%7C_L-system