

MSc Virtual &Augmented Reality

Simple L-System model

Dua Farrukh

24th Of November 2022

Youtube Video Link

<https://youtu.be/6oCduWzAhBE>

L-System Graphical Modeling

The Lindenmayer system, often known as the L-system, is a parallel rewriting system and a sort of formal grammar. An L-system is made up of an alphabet of symbols that may be used to produce strings, a set of production rules that extend each symbol into some larger string of symbols, a starting "axiom" string from which to build, and a mechanism for translating the generated strings into geometric structures.

Features:

L-systems are important to this project; however, the rewriting and geometric interpretation of such L-systems is handled using Python 3's Turtle package. The fundamental concept of turtle interpretation is presented below. A Turtle state is described as a triplet (x, y, θ) , where the Cartesian coordinates (x, y) represent the turtle's position and the angle, called the heading, represents the direction the turtle is facing. The turtle can respond to commands expressed by a particular collection of symbols provided the step size d and the angle increment (i.e., "F" meaning "move forward").

Using the same rewriting rules, this project may generate Fractals of various shapes using a recursive set of rules provided by the user.

GUI Features:

You can define an L-system by specifying:

- Starting Axiom
- Production Rule
- Number of iterations
- Segment Length
- Initial Angle
- Drawing Angle

Detail:

How does it work?

We choose an axiom, for example: 'A'

Now we have rules like: "A becomes AF" and "F becomes FA"

The next step is to iterate the axiom through the rules by a specific amount, the output will be fed through the rules again.

For example we iterate 3 times:

A -> AF

AF -> AFFA

AFFA -> AFFAFAAF

Our result string is now AFFAFAAF

The processing of the string works by checking char for char against the processing rules and do what they say. If we have something that doesn't fit any rules, it doesn't matter, we will just ignore it.

Processing the upper string would be lame, it will create a straight line. Let's add a few more interesting chars to the production rules:

F -> a2FF-[c1-F+F+F]+[c1+F-F-F]

Axiom: F

Angle: 23

Iteration: 5

Well, that looks a little bit more complicated, and it is. This production rule contains nearly every possible processing rule and will create a colored tree with different-sized lines.

Implementation:

An L-System renderer's job is to take an L-System description (axiom, angle increment, and replacement rules) and an iteration count and generate a list of line segments to draw. I want to be clear that my goal is not to draw line segments rapidly; rather, I want to compute the (x,y) coordinates for all line segments as soon as feasible for a particular L-System and iteration count. Therefore, the metric for benchmarking is *unit time per line segment computed* (lower is better).

These are the supported commands and Variables:

R: recursive string-rewriting where symbols on the right side of the recursive formulas are substituted by corresponding strings on the left side.

L: recursive string-rewriting where symbols on the left side of the recursive formulas are substituted by corresponding strings on the right side.

+: turn angle left.

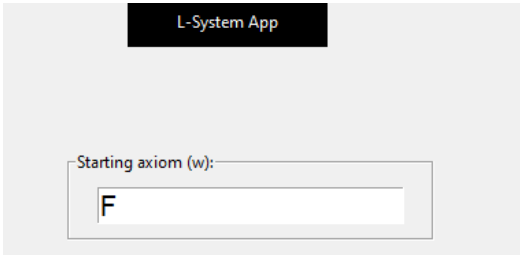
-: turn angle right.

[: push string / state to the memory stack.

]: lift pen up / stop drawing / pop string / state off the memory stack

Starting Axiom:

As stated in the first part, when you initialize your L-System, you can simply set your axiom.



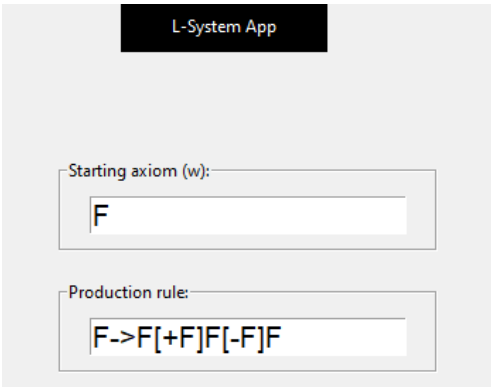
The screenshot shows a web application titled "L-System App". Below the title, there is a label "Starting axiom (w):" followed by a text input field containing the character "F".

Setting Production:

Productions define how an axiom's symbols are converted. For example, if you want all A's in your axiom to be replaced with B or all B's in your axiom to be replaced with A. you may build the following production respectively:

- $A \rightarrow B$
- $B \rightarrow A$

In our project, we have graphical user interface, so input production rule that can be seen below:



The screenshot shows the "L-System App" interface with two input fields. The first field, labeled "Starting axiom (w):", contains the character "F". The second field, labeled "Production rule:", contains the string "F->F[+F]F[-F]F".

Function in our program that apply production rules is given below:

```
def apply_production_rules(axiom, iterations):

    list_of_substrings = [axiom]
    # for each iteration
    for _ in range(iterations):
        # start from the last substring in your list of substrings
        current_substring = list_of_substrings[-1]
        # for each character in that current substring, apply the production rule to it
        new_axiom = [production_rule(char) for char in current_substring]
        # add the new string to the list of substrings
        list_of_substrings.append(''.join(new_axiom))
    return list_of_substrings
```

Code Structure and implementation:

You will almost certainly want to visualize or post-process your L-Systems output in some way. You could iterate and interpret the output yourself, but `lindemayer` already provides a simple mechanism to implement such post processing: final functions. You can specify what should be done for each literal/character in those final functions. The traditional application of L-Systems is to visualize axioms with turtle graphics.

So, in our Python code, let's define some useful functions. We'll need to make our first turtle item or pen.

```
def create_pen(initial_angle):
    ...

    Receives initial orientation / direction given as an draw_angle (ex - 90).
    Returns a turtle object that will be your pen.
    ...

    # Create your pen
    pen = turtle.Turtle()
    # Adjust your pen's width
    pen.pensize(1)
    # Adjust your pen's color
    pen.pencolor('green')
    # Adjust how fast you want to draw (fastest = 0)
    pen.speed(0)
    # Specify a title for your window
    pen.screen.title('L-System')
    # Specify initial direction you want to draw in
    pen.setheading(initial_angle)

    return pen
```

Next, we wrote a helper function to check for the presence of our preceding string. If that's the case, we'll replace it with the string that corresponds to it.

```
def production_rule(sequence):
    ...

    Given a predecessor string
    Check if that string is in the dictionary of production rules
    Return the corresponding successor string,
    Else return what you received
    ...

    # if predecessor string is in the dictionary
    if sequence in DICT_OF_RULES:
        # return the new successor string
        return DICT_OF_RULES[sequence]
    # else if it is not present, just return
    # the predecessor string mapped to itself
    return sequence
```

Then, in the following piece of code, we used that helper function to apply all of our production rules to our given string.

```
def apply_production_rules(axiom, iterations):
    ...

    Given an initial string / axiom and amount of iterations
    Apply all the production rules to that list of symbols
    Return a new list of symbols
    ...

    # turn axiom string into a list of substrings
    # in our case, each of these substrings are
    # an element of our list and each starts out
    # being a single character like F, +, -, etc
    list_of_substrings = [axiom]
    # for each iteration
    for _ in range(iterations):
        # start from the last substring in your list of substrings
        current_substring = list_of_substrings[-1]
        # for each character in that current substring, apply the production rule to it
        new_axiom = [production_rule(char) for char in current_substring]
        # add the new string to the list of substrings
        list_of_substrings.append(''.join(new_axiom))
    return list_of_substrings
```

The function that draw L-System is shown below:

```
def draw_L_system(our_pen, DICT_OF_RULES, segment_length, angle):
    # initialize your stack to push and pop your current L-system states
    stack = []
    # for each symbol in your given string
    for command in DICT_OF_RULES:
        # begin drawing
        our_pen.pendown()
        if command in ['F', 'R', 'L']:
            # draw one line segment length forward
            our_pen.forward(segment_length)
        elif command == 'f':
            # lift pen up / stop drawing
            our_pen.penup()
            our_pen.forward(segment_length)
        elif command == "+":
            # turn left
            our_pen.left(angle)
        elif command == "-":
            # turn right
            our_pen.right(angle)
        elif command == "[":
            # push string / state to the memory stack
            stack.append((our_pen.position(), our_pen.heading()))
        elif command == "]":
            # lift pen up / stop drawing
            our_pen.penup()
            # pop string / state off the memory stack
            position, heading = stack.pop()
            our_pen.goto(position)
            our_pen.setheading(heading)
```

Rules and Results:

In this project, we are given six plant like structures that we implemented and drawn successfully. There is a rule and required parameters for each structure.

These six images are given below along with rules and parameters used:

a.

Starting Axiom: F

Production Rule: $F \rightarrow F[+F]F[-F]F$

Number of iteration: 5

Segment Length: 4

Initial Angle: 90

Drawing Angle: 25.7

Result:



b.

Starting Axiom: F

Production Rule: $F \rightarrow F[+F]F[-F][F]$

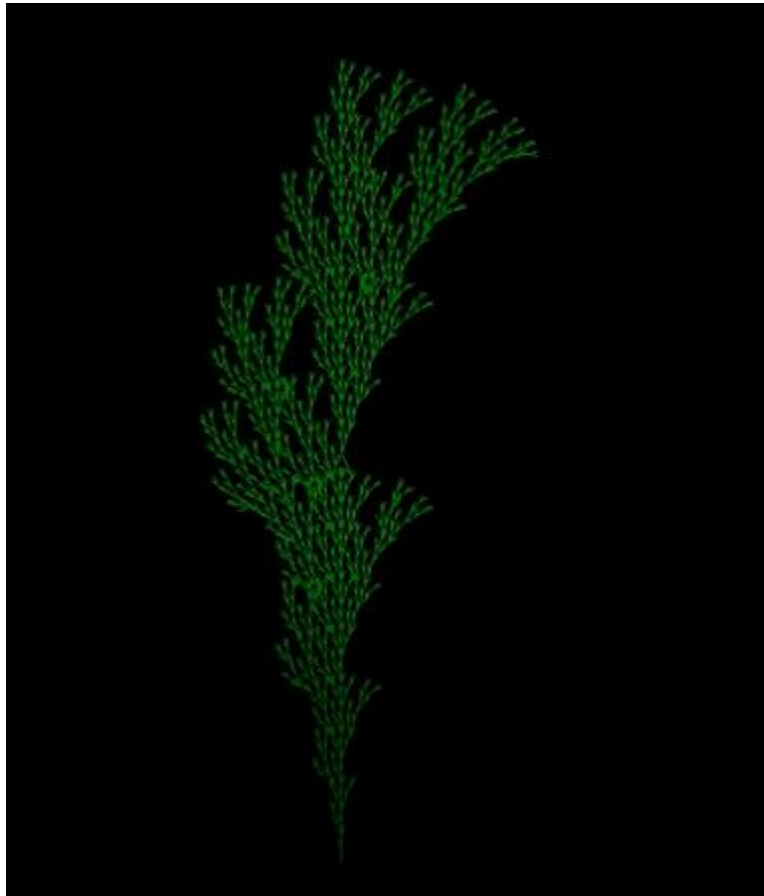
Number of iteration: 5

Segment Length: 6

Initial Angle: 90

Drawing Angle: 20

Result:



c.

Starting Axiom: F

Production Rule: $F \rightarrow FF - [-F + F + F] + [+F - F - F]$

Number of iteration: 4

Segment Length: 5

Initial Angle: 90

Drawing Angle: 22.5

Result:



d.

Starting Axiom: X

Production Rule: $X \rightarrow F[+X]F[-X]+X$

Production Rule (0 if no rule): $F \rightarrow FF$

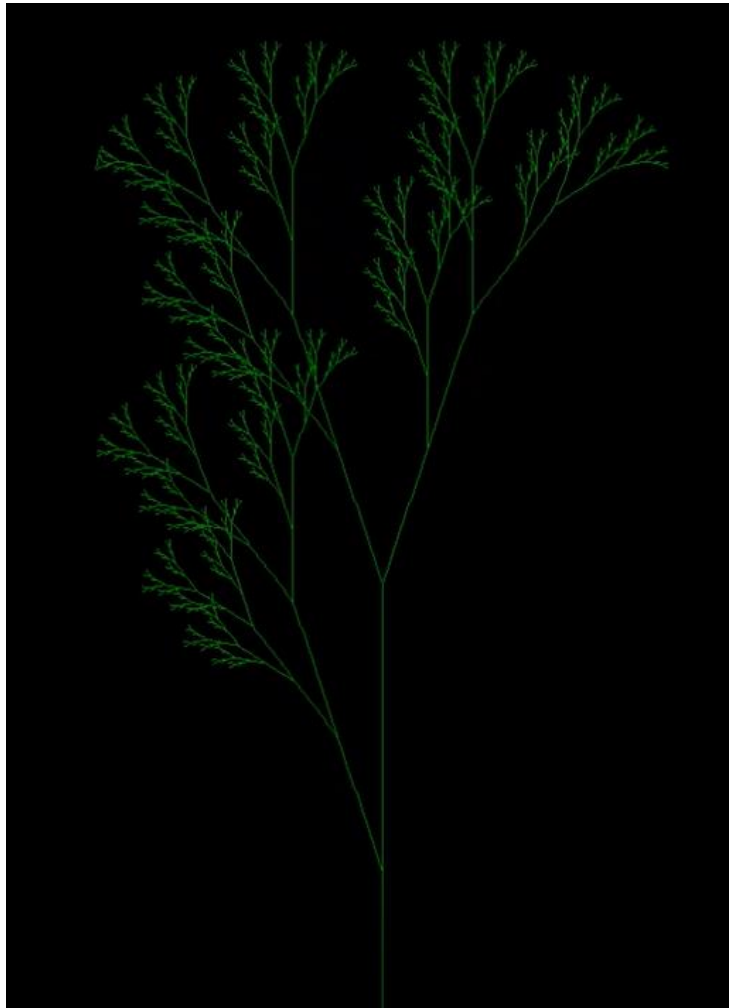
Number of iteration: 7

Segment Length: 3

Initial Angle: 90

Drawing Angle: 20

Result:



e.

Starting Axiom: X

Production Rule: $X \rightarrow F[+X][-X]FX$

Production Rule (0 if no rule): $F \rightarrow FF$

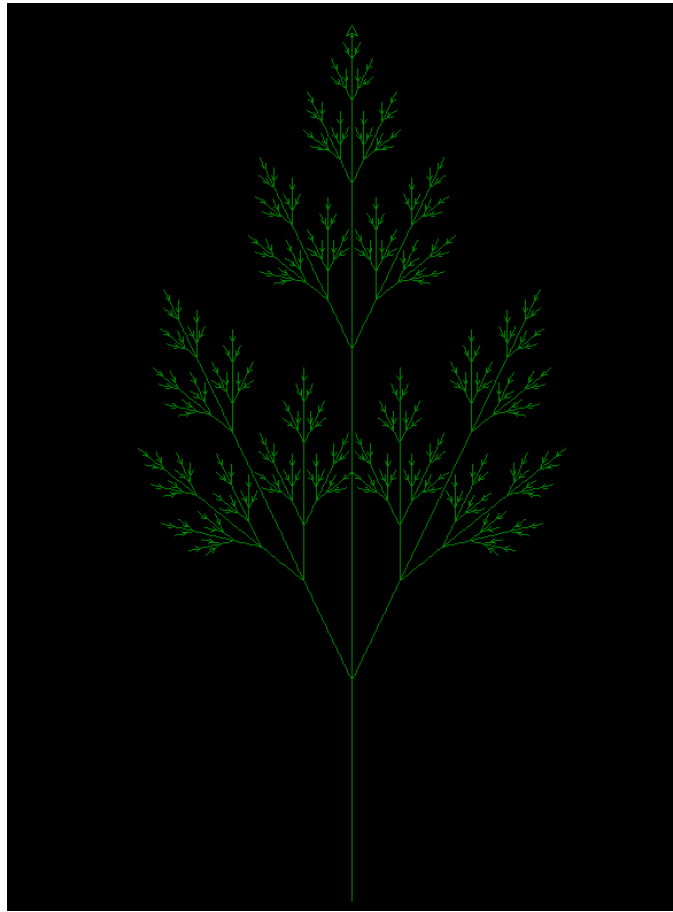
Number of iteration: 7

Segment Length: 4

Initial Angle: 90

Drawing Angle: 25.7

Result:



f.

Starting Axiom: X

Production Rule: $X \rightarrow F - [[X] + X] + F [+FX] - X$

Production Rule (0 if no rule): $F \rightarrow FF$

Number of iteration: 5

Segment Length: 4

Initial Angle: 90

Drawing Angle: 22.5

Result:



g.

Starting Axiom: Y

Production Rule: $X \rightarrow X[-FFF][+FFF]FX$

Production Rule (0 if no rule): $Y \rightarrow YFX[+Y][-Y]$

Number of iteration: 5

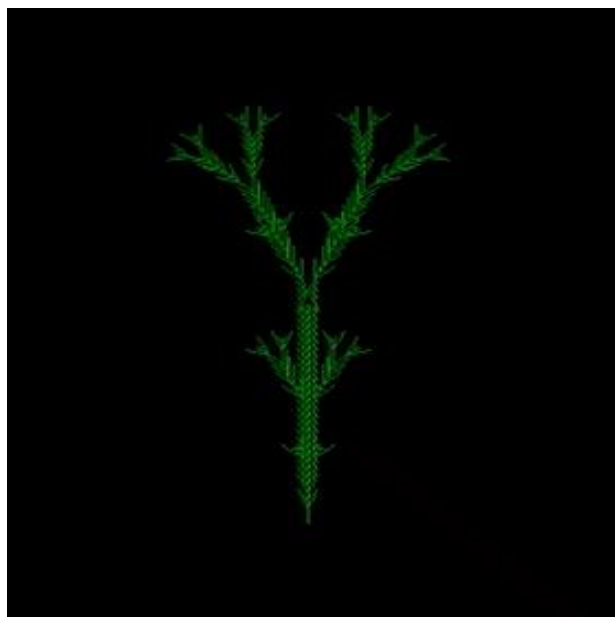
Segment Length: 4

Initial Angle: 90

Drawing Angle: 25.7

Taken From: <http://paulbourke.net/fractals/lsys/>

Result:



h.

Starting Axiom: F

Production Rule: $F \rightarrow F[+FF][-FF]F[-F][+F]F$

Production Rule (0 if no rule): 0

Number of iteration: 3

Segment Length: 5

Initial Angle: 90

Drawing Angle: 35

Taken From: <http://paulbourke.net/fractals/lsys/>

Result:



Conclusion:

Lindenmayer used L-systems to describe the behavior of plant cells and to model the growth processes of plant development. L-systems have also been used to model the morphology of a variety of organisms and can be used to generate self-similar fractals. We successfully implemented L-system which can draw different structures based on rules and parameters. Although this is not a complete solution to this domain.

References:

1. Deussen, Oliver, and Bernd Lintermann. "A modelling method and user interface for creating plants." *Graphics interface*. Vol. 97. 1997.
2. De Reffye, Phillippe, et al. "Plant models faithful to botanical structure and development." *ACM Siggraph Computer Graphics* 22.4 (1988): 151-158.
3. Prusinkiewicz, Przemyslaw, and James Hanan. *Lindenmayer systems, fractals, and plants*. Vol. 79. Springer Science & Business Media, 2013.
4. Perttunen, Jari, and Risto Sievänen. "Incorporating Lindenmayer systems for architectural development in a functional-structural tree model." *Ecological modelling* 181.4 (2005): 479-491.
5. Prusinkiewicz, Przemyslaw, Aristid Lindenmayer, and James Hanan. "Development models of herbaceous plants for computer imagery purposes." *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 1988.
6. My YouTube video link. <https://youtu.be/6oCduWzAhBE>