The Cactus (L-systems)

Year 2024 – 2025

Briscan Maria, Furdui Daria, Bujoi Denisa, 10th grade

Burca Andrei, Hangan Miruna, Matei Bărbat, 11th grade

School: Colegiul Național “Emil Racoviță” Cluj-Napoca

Teacher: Ariana-Stanca Văcărețu

Researcher: Yves Papegay-Inria, INRIA - Sophia Antipolis Méditerranée Research Center

1. Introduction

Our research topic was based on Lindenmayer systems, which are a type of formal language primarily used to simulate a simplified version of the growth of plants. These types of systems were introduced by Aristid Lindenmayer, a Hungarian biologist, in 1968.
The key components of this type of systems are the following:

- The alphabet, which is made of a number of characters, each of them having a specific meaning, which are used to create strings.
- The rule that has the purpose of determining what characters get replaced and with what string when we go from a generation to another.
- The axiom, that represents the string we are starting with.

1.1. The problem

We define an alphabet:

- F (take one step forward)
- + (turn left 90°)
- - (turn right 90°)
- [(Remember the position)
-] (Return to last remembered position)

We define a rule: an F becomes FF[+F-F]FF[-F+F]FF.
We start from F (generation 0).
What happens to the next generations? What is the number of symbols from the alphabet used, but the height of the cactus? Can you make a graphic representation of the cactus?

1.2. Results

We found the following formulas to answer the questions from the research topic:

Number of generation	Height (if F=1 unit)	Number of F's	Number of +, -, [,]
1	6	10	2
2	36	100	22
3	216	1000	222
⋮	⋮	⋮	⋮
n	6^n	10^n	$2\frac{10^n-1}{9}$

Figure 1 - Formulas

A representation of the growth of the cactus for better understanding the topic is in image 1.

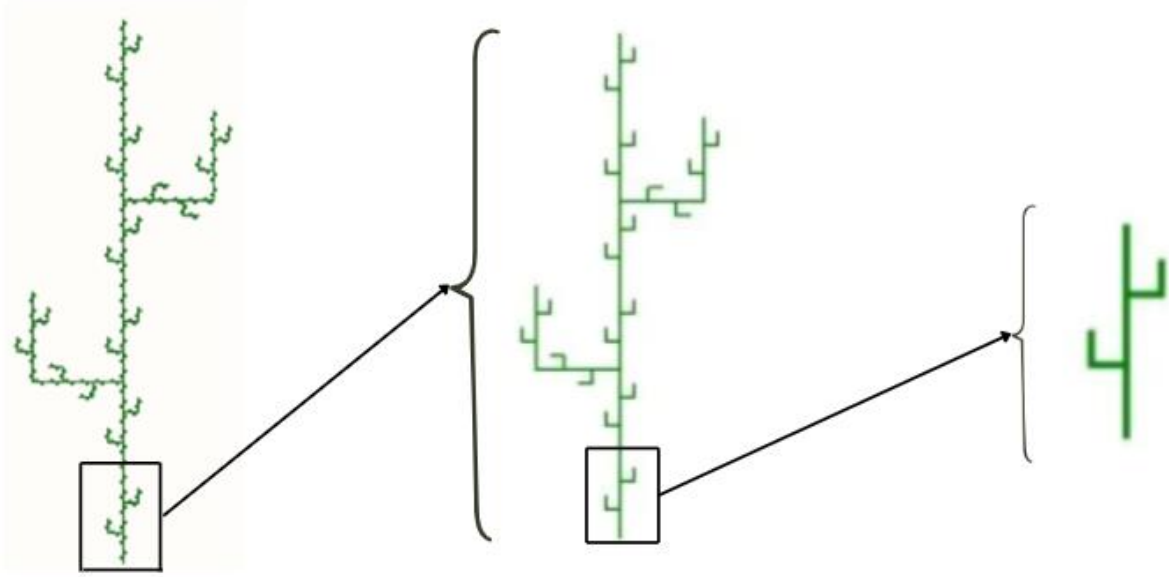

Figure 2 - The growth of the cactus

2. Objective

Our objective is to be able to simulate the growth of a cactus using a programming language and to analyse its height and the way the simulation behaves. Essentially, we generate as many different cactuses as possible in order to understand the basis of its growth.

3. Approaches and Problem Solving

3.1 First Approach
Analysing the cactus' rules and watching it evolve

We started by drawing the cactus on a sheet of paper so we could understand better how the growth of the cactus occurs. After, we made a physical representation of the first three generations of the cactus using playdough and toothpicks. From there we noticed a few patterns, so we tried to find some formulas, which are presented in table 1.

After that, we looked up information about fractals and L-systems and made a comparison between them and found that L-systems are generated by rewriting symbols based on rules, whereas fractals are typically generated by applying mathematical functions iteratively. While L-Systems excel in capturing natural forms with hierarchical structures, fractals are mathematical sets with self-similar patterns at varying scales, often described using equations or iterative algorithms. Both have applications in nature modelling, but their approaches and underlying principles differ.

Then, using the python programming , we created a code that shows the n^{th} generation of the cactus (n is a given number that we choose)

By experimenting with different generations we observed that the number of F's was always equal to 10^n (n is the number of the generation) and the height 6^n and the number of the symbols + - [ ] is a number of the following form 222…2, the number of 2's being n which has the formula we demonstrated by proving that the numbers 1, 10, 100, 10^{n-1} are part of a geometric sequence.

$$22..2 = 2 \times \underbrace{(1 + 10 + 100 + \ldots + 10^{n-1})}_{\begin{matrix} || & || & || & || \\ b_1 & b_2 & b_3 & b_n \end{matrix}}$$

$$\frac{b_n}{b_{n-1}} \stackrel{?}{=} constant \qquad \frac{10^{n-1}}{10^{n-2}} = \frac{10^{n-2} \times 10}{10^{n-2}} = 10 = constant$$

$$=> 1, 10, 100, \ldots 10^{n-1} \quad \text{are part of a geometric sequence}$$

$$1 + 10 + 100 + \ldots + 10^{n-1} = \frac{10^n - 1}{9}$$

$$=> 22..2 = 2 \times \left(\frac{10^n - 1}{9}\right)$$

3.1.1 Explanation of the first code

```
import turtle
def createLSystems(iterations, axiom):
    startString = axiom
    endString = ""
    for i in range(iterations):
        endString = processString(startString)
        startString = endString
    return endString
def processString(oldStr):

    newstr = ""
    for ch in oldStr:
        newstr = newstr + applyrules(ch)
    return newstr

```

Figure 3 - First code

This Python sequence defines part of a program for generating L-systems (Lindenmayer systems) using turtle graphics.

This program builds an L-system string by:

1. Starting with an axiom.

2. Applying transformation rules (applyrules, which is in the next sequence) over several iterations.

```
16 def applyrules(ch):
17
18     newstr = ""
19     if ch == "F":
20         newstr = 'FF[+F-F]FF[-F+F]FF'
21     else:
22         newstr = ch
23
24     return newstr
25
```

Figure 4 - Sequence that applies the rules

3. Returning the final string.

```
26 def drawLSystems(turtle,instructions,angle,size):
27     stack=[]
28     for cmd in instructions:
29         if cmd == 'F':
30             turtle.forward(size)
31         if cmd == '+':
32             turtle.left(angle)
33         if cmd == '-':
34             turtle.right(angle)
35         if cmd == '[':
36             stack.append((turtle.position(), turtle.heading()))
37         if cmd == ']':
38             position, heading = stack.pop()
39             turtle.penup()
40             turtle.goto(position)
41             turtle.setheading(heading)
42             turtle.pendown()
43
```

Figure 5 - Defining the symbols

In image 5 the shown function defines what every symbol indicates.

3.1.2 Explanation of the second code

This Python code basically implements a certain list of actions and allows the user to create their own L-System by using them. The user has to:

1. type their desired rule (for example “FF[+F-F]FF[-F+F]FF”);

2. set an angle at which the L-System will make turns;

3. set the number of generations they want generated.

After doing so, the code will show a 3D model based on the rules given by the user.

3.2 Second Approach

Simulating Nature with Python: Evolving Cacti and L-Systems into 3D Models

Our journey began with a simple idea: simulating the evolution of a cactus using Python. We quickly developed a basic program capable of generating cactus-like growth through iterative simulations. As we analysed the patterns emerging from our code, we noticed a resemblance to forms found in nature. This curiosity led us to discover Lindenmayer systems (L-systems).

Figure 6 - L-systems and Nature (https://en.wikipedia.org/wiki/L-system)

The realization that our cactus simulation was essentially a rudimentary L-system sparked a shift in our approach. We decided to explore this mathematical model further, aiming to mimic the complexity and elegance of natural structures more faithfully. To accomplish this, we refined our Python program by **introducing a variable to control the angle of branching.** This seemingly simple change made a profound difference: the generated forms began to resemble real plants more closely.

Encouraged by these results, we expanded our program's capabilities. Our goal was to develop a tool capable of generating L-system-based objects of **arbitrary complexity**. We incorporated a **wider range of branching commands and introduced randomness in angular rotations** to enhance natural variability. The output is not just visually compelling—it's functional. Our program produces 3D-printable .stl files that can be viewed, edited, or fabricated using standard modeling tools.

One of the key breakthroughs came through the development of new **L-system rules** tailored for more **realistic, organic growth**. For example, we used the following rule set:

```
rules = {
   "F": "F[+F][&F][F/F^F][-F]F"
```

}
This rule introduces **multi-directional branching** and **layered depth**, mimicking how real plants expand in 3D space. The syntax works as follows:

F: Move forward and draw a branch.

[ ]: Save and restore the turtle's position and orientation—critical for branching structures.

+ / -: Rotate left or right around the Z-axis.

& / ^: Tilt down or up around the X-axis.

/ / \\: Roll left or right around the Y-axis.

By combining these in a single rule, our grammar now simulates **spiral, forked, and cascading plant structures**, adding depth and symmetry to the model—something traditional 2D L-systems could not easily achieve.

To bring these models to life, we used trimesh for geometry creation and pyrender for visualization. Each branch is rendered as a textured cylinder with variable direction, and **leaf nodes** appear as green spheres at the end of growth sequences. The result is a dynamic, colorized 3D mesh that reflects the intricate beauty of biological growth.

Our final program can:

Generate complex L-system strings from customizable rule sets and axiom inputs

Interpret these strings into 3D geometries with trunk, branches, and leaves

Visualize the results interactively in real-time

Export the model as an .stl **file** ready for 3D printing

These innovations turn a mathematical idea into a living, breathing simulation of nature—powered entirely by code.

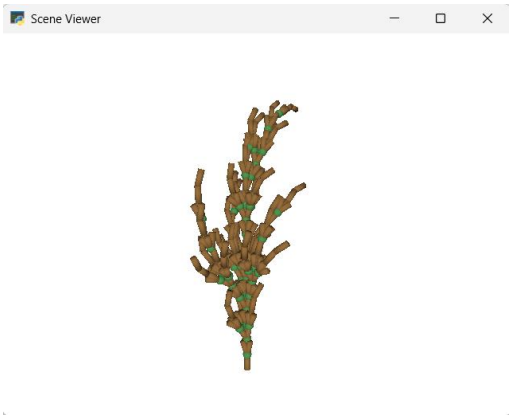

Figure 7 - Rule: F[+F][&F][F/F^F][-F]F

Figure 8 - Rule: F[+F][-F]/F[&F]^F\\F

4. Physical approach

Because of the direct connection of the research problem with Fractals and L-System, we used a 3D printer to bring the resulted patterns from the digital world into the real world.

Fractals, known for their self-similar and intricate structures, can be translated into tangible objects using 3D modelling software. These prints reveal stunning geometric designs, often mimicking natural forms like trees, snowflakes or coral.

Figure 9 - 3D printed L-system

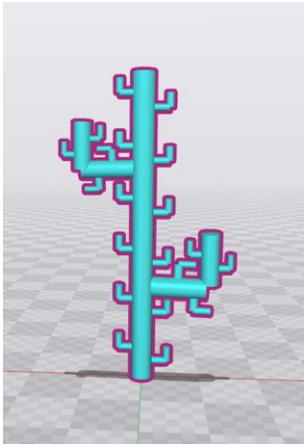

Figure 10 - second generation

Figure 11 - 3D model

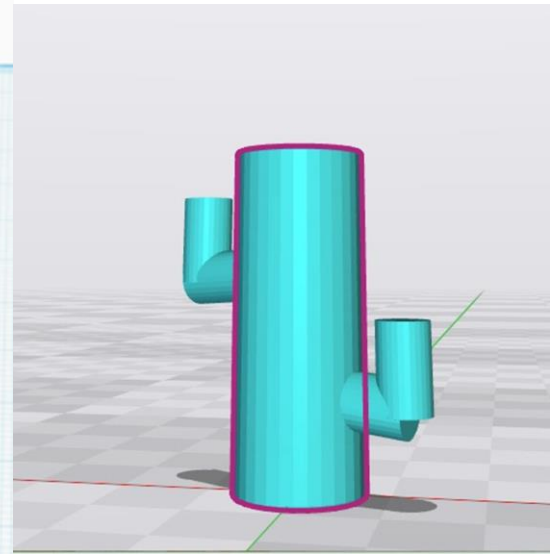

Figure 12 - First generation 3D model

5. The code

By scanning the QR Code you can see and analyze the code used.

6. Future Plans

Building on the foundation of our current work, our next goal is to bring the digital world even closer to the natural one. We plan to **integrate image recognition** techniques to analyse photographs of real plants, branches, and natural formations. By identifying structural patterns in these images, we hope to automatically extract characteristics that can inform and refine our L-system rules.

Beyond pattern extraction, we are also **seeking new and unexpected applications** of L-systems—possibilities that may not yet be visible through traditional modelling. Our hope is that, through continued experimentation, **L-systems might reveal hidden mathematical structures in nature**, or even inspire entirely new forms in art, design, or biology.

This next phase combines computational modelling, machine learning, and observation of the real world—and we're excited to see what insights and innovations may emerge.

7. Conclusion

Our exploration into Lindenmayer systems began with a simple goal: to simulate the growth of a cactus. Along the way, we discovered that these formal systems offer a powerful and flexible way to model not just plants, but the very logic behind natural growth. Through experimentation and iteration, we gained deeper insight into how structured rules can produce surprisingly organic results.

We started with a **manual and visual approach**, sketching out generations of cacti on paper and building physical models using playdough and toothpicks. This hands-on method helped us understand the fundamental structure of our rules and observe recurring patterns.

Next, we turned to **mathematical modelling**, analysing symbol growth, height progression, and rule expansion over generations. This gave us formulas to describe the behaviour of the system and offered measurable insights into how complexity evolves over time.

From there, we moved into **programmatic simulation**, writing Python code that could generate any number of cactus generations using L-system rules. By visualizing these systems with turtle graphics and later in 3D using trimesh and pyrender, we created rich, detailed representations of plant-like structures.

Finally, we advanced to a **3D generative design approach**, where we not only simulated plant growth in virtual environments but also brought them into the physical world through 3D printing. This bridge between digital and tangible worlds helped us appreciate the real-world applicability of abstract systems.