Goldsmiths
UNIVERSITY OF LONDON

L-Systems made with Unity

MSc Computer Games Programming

Roger Ruiz Tristancho

Basics of Mathematics for Games and VR/AR

25, November 2022

INDEX

Introduction and objectives

The main purpose of the project was to implement a simple model of L-System based on the work of Aristid Lindenmayer.
In this case, the project was made with Unity3D using C# as a programming language. The plan was to develop a L-System renderer fully customizable with 10 different presets. The user is able to choose from these 10 different L-Systems and to modify some parameters like the length of the shape, the number of itineration and the angles (having a randomizer option where the user can set a range for it).

As secondary objectives, I wanted to be able to build the trees as a 3D objects so they could be used as videogame assets, and also I wanted to experiment with the colors and shaders to have a more realistic feel at the end.

Features

To draw the different trees, I used a Unity component called Line Renderer which is the one that allows me to manipulate it creating different lines as they were branches from a tree.
The renderer has different options in the UI that will let the user modify some parameters from the 10 original presets.

Figure 1. picture of the L-System Renderer

First of all, I chose these parameters to be able to be modified because they are the once that can change the shape of a l-system but without changing it entirely. In my plans the user wouldn't manipulate the rules of each system but, he/she could change entirely the shape of the tree experimenting with the angles, lengths and amount of itineration.

On the top site of the figure 1, there are 10 different buttons with tree shape lines that indicates the user which tree does each one renders. Basically, each one has an specific rule (e.g. the first one's rule is: F -> F[+F]F[-F]F). In addition, when a button is pressed and it renders a tree, it will have already the default values of its angle, length, etc.
On the right site we have 3 different drop downs and one checkmark that is disabled by default.
The first dropdown is the one that sets the number of itineration the shape will have. The range is from 1 to 5 because it can fit correctly in the screen. The user will be able to change this value and to see the chances in real time, since when you make a small change, it reloads again the shape.
The second drop down is the one in charge of setting the length of the shape. Following the same formula as the first dropdown, it changes the value in real time and in this case the range is from 1 to 7 for the same reasons as the one mentioned before.
At last, there is the dropdown that adds to the angle another value so the user can increase it. You can add from 5 to 30 more degrees to the default one and if you select 0 it became normal again. In this case there is a difference between the other two dropdown.

Figure 2Randomize is selected

Below it, as it is visible in the figure 2, there is a checkmark that says *randomize angle.* If this is selected, the dropdown that adds value to the angles is disabled and now two input text items appear. The user can set a range of the possibilities the random value can have, and that would generate a render with not always the same angle, but every branch will have a random one between the range set. Once the user has set the range of values, then he/she has to press the button accept to generate the shape with the randomizer. If we disable the checkmark, they are now able to add value to the actual angle and the input camps get disabled again.

Data Classes

I created two different classes to store information and to render the l-system. The class Trans is a class that I use to create a transform variable with a position and a rotation in order to set these parameters to the branches generated.

```
public class Trans
{
    public Vector3 position;
    public Quaternion rotation;

}
```

The second class is the one in charge of generate the renders and to listen to the changes the user makes to the model.

```
public class LSystem : MonoBehaviour
{
    public Transform spawn;
    public Toggle toggle;
    public Dropdown dropdown;
    public GameObject toggEnabled;
    private int preAngle = 0;
    private float originalAngle;
    private int doOnce = 0;

    [SerializeField] private int itinerations = 5;
    [SerializeField] private GameObject branch;
    [SerializeField] private float length = 7;
    [SerializeField] private float angle = 25.7f;

    private Stack<Trans> transformStack;
    private Dictionary<char, string> rules;
    public float randomvalue1 = 0;
    public float randomvalue2 = 0;

    private string currentString = string.Empty;
```

Figure 3. L-System class variables

In the figure 3 there are all the variables used to develop all the features from the generator to the randomizer. There we have all the UI components that will be used to change the values and the default settings for the first tree: The itineration, the branches (as different gameobjects), the length of it and the angle.

In the Update function, we will be checking if the toggles are on or not. If it is, we will enable the randomizer input fields and disable the dropdown of the angles. If its disabled, we will do the opposite. The input fields will be disabled again, the dropdown enable and lastly, the angle values will be set to default again.

```
private void Update()
{
    if (toggle.isOn)
    {
        originalAngle = angle;
        dropdown.interactable = false;
        toggEnabled.SetActive(true);
        doOnce = 1;
    }
    else
    {

        dropdown.interactable = true;
        toggEnabled.SetActive(false);
        if (doOnce == 1)
        {
            angle = originalAngle;
            Generate();
            doOnce = 0;
        }


    }
```

Figure 4. Update function

```
public void Generate()
{
    DestroyObjects();
    transform.position = spawn.position;
    transform.rotation = spawn.rotation;

    if (rules.Count == 1)
    {
        currentString = "F";
    }
    else
    {
        currentString = "X";
    }

    StringBuilder sb = new StringBuilder();

    for (int i = 0; i < itinerations; i++)
    {
        foreach (char c in currentString)
        {
            sb.Append(rules.ContainsKey(c) ? rules[c] : c.ToString());

        }

        currentString = sb.ToString();
        sb = new StringBuilder();

    }
```

Figure 5. Generate function

The generate function is the one that is called every time we want to regenerate the model for any changes. First, the function DestroyObjects is called to destroy the previous shape and after that we set the position of the new shape in the spawn position. After that, we check if it has one rule of two because if not it won't work properly and just the ones that have an X axis would be drawn.

After it, now its time to check the rules to generate every single model. If there is an F we start building the branches. If its an X nothing happens. If it's a + it rotates backwards, and if – it rotates forwards. And at last but not least, if it is a [or a], the position and the rotation of the class Trans is set. (Check figure 5).

```
Debug.Log(currentString);
foreach (char c in currentString)
{
    switch (c)
    {
        case 'F':
            Vector3 initialPos = transform.position;
            transform.Translate(Vector3.up * length);
            GameObject segment = Instantiate(branch, spawn);
            segment.GetComponent<LineRenderer>().SetPosition(0, initialPos);
            segment.GetComponent<LineRenderer>().SetPosition(1, transform.position);

            break;
        case 'X':
            break;
        case '+':
            RandomAngle();
            transform.Rotate(Vector3.back * angle);
            break;
        case '-':
            RandomAngle();
            transform.Rotate(Vector3.forward * angle);
            break;
        case '[':
            transformStack.Push(new Trans()
            {
                position = transform.position,
                rotation = transform.rotation
            });

            break;
        case ']':
            Trans ti = transformStack.Pop();
            transform.position = ti.position;
            transform.rotation = ti.rotation;
            break;

        default:
            throw new InvalidOperationException("Invalid L-System");
    }
```

Figure 6. Check the rules

The SetItineration function only change the itineration value to the chosen one and it regenerates the L-system again with the new value.

```
public void SetItineration(int val)
{
    itinerations = val + 1;
    Debug.Log(val);
    DestroyObjects();
    Generate();
}
```

Figure 7. SetItineration

The function SetRule is the one that totally changes the shape of the model. When it is called the parameters (angle, itineration, and length) are totally changed apart from the rules and after that it is generated as a new shape. Is this case I added 10 different presets to play with. Some of them has just one rule and the rest has two including the X axis.

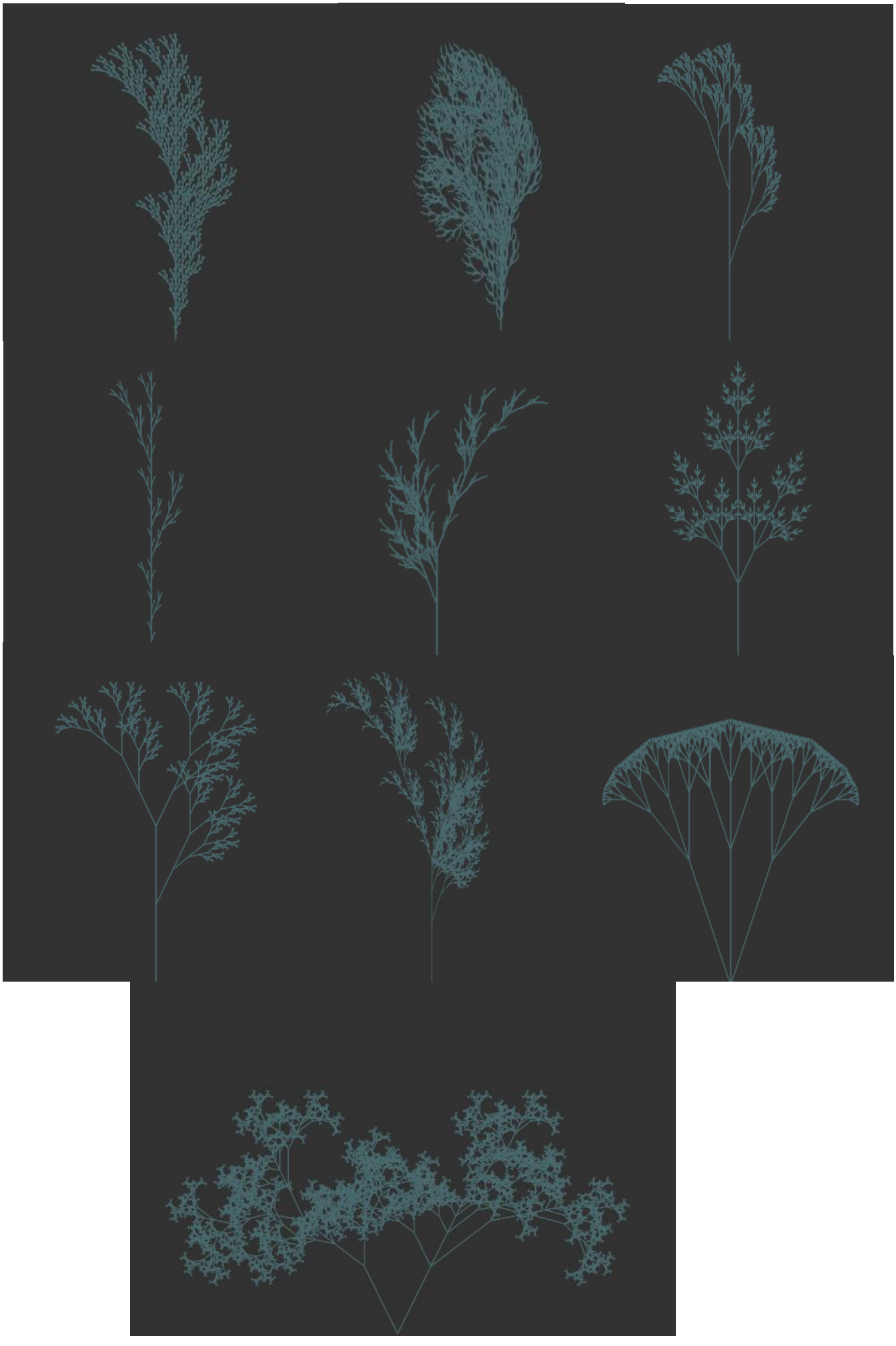
```
public void SetRule(int val)
{
    Debug.Log(val);
    if (val == 0)
    {
        transformStack = new Stack<Trans>();
        rules = new Dictionary<char, string>{
        {'F', "F[+F]F[-F]F" }
        };
        angle = 25.7f;
        length = 7;
        itinerations = 5;
        Generate();
    }
    else if (val == 1)
    {
        transformStack = new Stack<Trans>();
        rules = new Dictionary<char, string>{
        {'F', "F[+F]F[-F][F]" }
        };
        angle = 20f;
        length = 7;
        itinerations = 5;
        Generate();
    }
    else if (val == 2)
    {
        transformStack = new Stack<Trans>();
        rules = new Dictionary<char, string>{

        {'F', "FF-[-F+F+F]+[+F-F-F]" }
        };
        angle = 22.5f;
        length = 7;
        itinerations = 4;
        Generate();
    }
```

Figure 8. SetRule

Results

At the end I ended up with a program that successfully can print the different shapes I set as default and the user can play with them using some UI parameters. The UI is fully functional and its quite flexible. This are the final results: L-System video example



Conclusions

To conclude this report, I want to add some points that I want to explain about advantages, disadvantages, and problems I had during the process of developing this program.
First of all, I feel very happy with the final result since my main goals were achieved. I have a randomizer that change the values of the branches rotation, the user can change 3 parameters and I could manage to create 10 buttons with the shape of a different tree so the user can chose the one that prefers. But if I have to say something bad is that I would have liked to implement more functions. In my mind I was hoping to have something similar as a tree generator as real as possible including differences between colors, shapes, etc… and that is the par I missed. In addition, this development had evolved a lot since the first idea was to use the L-system and fractals to develop a realistic cancer evolution analysis changing some parameters etc… But after some research and being it a very ambitious idea I decided to come back with a simpler idea which has been the final result.
In addition I would have liked to implement a 3D view to make the results more realistic so you could implement this code for a real videogame. But at the end I ended up doing it in 2D using the unity component called line renderer.
Overall, I am satisfied since the basic goals where achieved even I would have liked to implement more presets, customization options or the cancer analysis function.

References

L-Systems Unity Tutorial [2018] -> https://www.youtube.com/watch?v=tUbTGWl-qus&t=901s