Lindenmayer System Models

Nikunj Sethi

IS71021C : Basics of Mathematics for Games and VR/AR

MSc Virtual and Augmented Reality

Date : 25/11/22

Github Link : https://github.com/nikunjsethi/L-System-Examples

Youtube Link : https://www.youtube.com/watch?v=DhR9-xfnHAQ

Itch.io Link : https://nikunjsethi.itch.io/l-systems

Abstract

In this assignment, I attempted to make customizable L-System renderer, where users can control angle, length and iterations of all the L-Systems I have created. I wanted to create a versatile tool that could be used by both those familiar and unfamiliar with L-Systems and would hopefully help the user understand L-Systems better. I created the program in Unity, using C# language. I have achieved the basic requirement of the assignment and has worked on some extra functionalities too.

Features

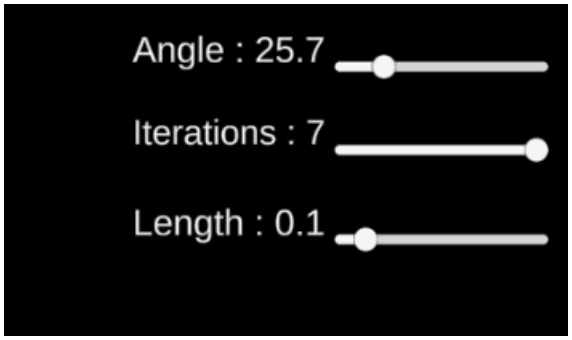
The main functionalities I wanted to work on was the User Interface of the program. It was important that the user was able to understand and modify all the parameters that I added in the program i.e. length, angle and iterations.

I did this by adding sliders for each parameters with different ranges.

For angles, I kept the minimum and maximum values as 10 degrees and 90 degrees respectively.

For iterations, I kept the whole number values from 1 to 7, 7 being the maximum and 1 being the minimum value

Same way, for length, I used 0.1 length for minimum value and 1 length for maximum value as the size varies **because of different rules and number of iterations.**



There are 2 ways to load different L-Systems-

1. One way is through Unity UI Buttons, all the user has to do is click on different color buttons and the respective L-System will load. The color on the buttons indicates the color of that L-System.

2. The other was is through the number keys on the keyboard. The user has to press the number key from 1 – 8 from keyboard to load the respective L- System in the scene.

```
if(Input.GetKeyDown(KeyCode.Alpha1))
{
    LSys1();
}
else if (Input.GetKeyDown(KeyCode.Alpha2))
{
    LSys2();
}
else if (Input.GetKeyDown(KeyCode.Alpha3))
{
    LSys3();
}
else if (Input.GetKeyDown(KeyCode.Alpha4))
{
    LSys4();
}
else if (Input.GetKeyDown(KeyCode.Alpha5))
{
    LSys5();
}
else if (Input.GetKeyDown(KeyCode.Alpha6))
{
    LSys6();
}
else if (Input.GetKeyDown(KeyCode.Alpha7))
{
    LSys7();
}
else if(Input.GetKeyDown(KeyCode.Alpha8))
{
    LSys8();
}
```

I have called this functionality in the Update function so that it can detect input change every frame.

Below are the UI variables I have used for the whole UI implementation functionality-:

```
[Header("UI")]
public TextMeshProUGUI currentLSyst;
public Slider angleSlider;
public Slider iterationSlider;
public Slider lengthSlider;
public TextMeshProUGUI angleText;
public TextMeshProUGUI iterationText;
public TextMeshProUGUI lengthText;
```

For the text component, instead of using Unity's in-build text component, I have used "**TextMeshPro**" package from the package manager. It is way better than the in-build text component in aspect of everything including text effect, font, anchoring.

```
0 references
public void OnSliderValueChange()
{
    float newAngle = angleSlider.value;
    int newIteration = ((int)iterationSlider.value);
    float newLength = lengthSlider.value;
    TreeGenerate(newIteration, newAngle, newLength);
}
```

I have attached this function to the slider bars of all 3 parameters in the UI. Through this way, as soon as the user changes the value in any of the 3 parameters, the TreeGenerate function(which will come in upcoming pages) will get called and L-System will change.

Implementation

For each L-System, I have created a different function with naming convention **“LSys_” where _ tells the number.**

```
public void LSys1()
{
    if (SliderHolders.activeInHierarchy == false)
        SliderHolders.SetActive(true);
    color = 1;
    Camera.transform.position = new Vector3(0, 15, -30);
    transformStack = new Stack<TransformInfo>();
    axiom = "F";
    rules =new Dictionary<char, string>
    {
        { 'F',"F[+F]F[-F]F"}
    };
    currentLSyst.text = "L-System 1";
    TreeGenerate(5, 25.7f,0.1f);
}
```

```
public class TransformInfo
{
    public Vector3 position;
    public Quaternion rotation;
}
Unity Script (1 asset reference) | 0 references
public class LSystem : MonoBehaviour
{
    int color;                                                    //to assign different
    [SerializeField] private GameObject Camera;
    [SerializeField] private GameObject SliderHolders;
    [SerializeField] private GameObject[] BranchObject;
    [SerializeField] private GameObject BranchingParent;

    private string axiom ;
    private Stack<TransformInfo> transformStack;
    private Dictionary<char, string> rules;
    private string currentString = string.Empty;
```

Each function first creates a new stack of “**TransformInfo**”, whose variables are defined in the above image, i.e. position and rotation. Then for each function, we have defined the “**axiom**” string, which basically means, the initial state of the system, from where, the set of production rules defining the way variables can be replaced with the combinations of constants and other variables.

Then we defines the rules for each L-System which is divided into two parts with dictionary. The first part is the axiom, from where the strings will replace, and the second part is the rules,

which will keep replacing the axiom at every call, creating a Tree, with branch like structure, according to the rules given in the function.

In each LSys_ function, camera position has been defined in order to keep the camera in center for each program because every L-System varies in size and starting position because of the production rules.

Color integer has been defined to give different colors to different systems.

```
void TreeGenerate(int iterations, float angle, float length)
{
    BranchCleaning();
    currentString = axiom;
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < iterations; i++)
    {
        foreach (char c in currentString)
        {
            sb.Append(rules.ContainsKey(c) ? rules[c] : c.ToString());
        }
        currentString = sb.ToString();
        sb = new StringBuilder();
    }
    foreach(char c in currentString)
    {
        switch(c)
        {
            case 'F':
                Vector3 initialPosition = transform.position;
                transform.Translate(Vector3.up * length);
                GameObject treeSegment = Instantiate(BranchObject[color],BranchingParent.transform);
                treeSegment.GetComponent<LineRenderer>().SetPosition(0, initialPosition);
                treeSegment.GetComponent<LineRenderer>().SetPosition(1, transform.position);
                break;

            case 'X':
                break;

            case '+':
                transform.Rotate(Vector3.forward * angle);
                break;

            case '-':
                transform.Rotate(Vector3.back * angle);
                break;

            case '[':
                transformStack.Push(new TransformInfo()
                {
                    position = transform.position,
                    rotation=transform.rotation
                });
                break;

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

            default:
                throw new InvalidOperationException("Invalid L-Tree operation");

        }
```

TreeGenerate is the main function of this script.

Here we have assigned all the rules with different functionalities, from instantiating a branch node in the scene with **F**, to rotating it in different directions with angles with **+** & **-,** to pushing the new nodes with **[** and popping out the nodes with **]**.

I have given 3 different arguments to this function with iterations, angle and length so that we can directly call the GenerateTree function from inside LSys_ function with the specific values for iterations, angle and length.

```
1 reference
void BranchCleaning()
{
    for (int i=0;i<BranchingParent.transform.childCount;i++)
    {
        Destroy(BranchingParent.transform.GetChild(i).gameObject);
    }
    transform.position = new Vector3(0, 0, 0);
    transform.rotation = Quaternion.Euler(0, 0, 0);
}
```

There is one extra function I had to create which I am calling inside the Generate function, i.e. it is getting called everytime we switch from one L-System to another.

The **BranchCleaning** function is created to remove all the nodes of previous L-System before start creating nodes for new L-System as it will get override and will generate some weird/messy output.

Also, we are resetting the position and rotation of the gameobject from which the node generation begins, so that the point always remains in center only

Code

```
using System.Collections;
using System.Collections.Generic;
using System.Text;
using UnityEngine;
using System;
using UnityEngine.UI;
using TMPro;

public class TransformInfo
{
    public Vector3 position;
    public Quaternion rotation;
}
public class LSystem : MonoBehaviour
{
    int color;                                                          //to assign
different color to different L-System
    [SerializeField] private GameObject Camera;
    [SerializeField] private GameObject SliderHolders;
    [SerializeField] private GameObject[] BranchObject;
    [SerializeField] private GameObject BranchingParent;

    private string axiom ;
    private Stack<TransformInfo> transformStack;
    private Dictionary<char, string> rules;
    private string currentString = string.Empty;

    [Header("UI")]
    public TextMeshProUGUI currentLSyst;
    public Slider angleSlider;
    public Slider iterationSlider;
    public Slider lengthSlider;
    public TextMeshProUGUI angleText;
    public TextMeshProUGUI iterationText;
    public TextMeshProUGUI lengthText;

    private void Start()
    {
        SliderHolders.SetActive(false);
    }
    private void Update()
    {
        if(Input.GetKeyDown(KeyCode.Alpha1))
        {
            LSys1();
```

```
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            LSys2();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            LSys3();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha4))
        {
            LSys4();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha5))
        {
            LSys5();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha6))
        {
            LSys6();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha7))
        {
            LSys7();
        }
        else if(Input.GetKeyDown(KeyCode.Alpha8))
        {
            LSys8();
        }
    }
    public void LSys1()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 1;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "F";
        rules =new Dictionary<char, string>
        {
            { 'F',"F[+F]F[-F]F"}
        };
        currentLSyst.text = "L-System 1";
        TreeGenerate(5, 25.7f,0.1f);
    }

    public void LSys2()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 2;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "F";
        rules = new Dictionary<char, string>
        {
```

```
            { 'F',"F[+F]F[-F][F]"}
        };
        currentLSyst.text = "L-System 2";
        TreeGenerate(5,20,0.5f);
    }

    public void LSys3()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 3;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "F";
        rules = new Dictionary<char, string>
        {
            { 'F',"FF-[-F+F+F]+[+F-F-F]"}
        };
        currentLSyst.text = "L-System 3";
        TreeGenerate(2, 22.5f,0.5f);
    }

    public void LSys4()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 4;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "X";
        rules = new Dictionary<char, string>
        {
            { 'X',"F[+X]F[-X]+X"},
            {'F',"FF" }
        };
        currentLSyst.text = "L-System 4";
        TreeGenerate(7,20,0.1f);
    }

    public void LSys5()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 5;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "X";
        rules = new Dictionary<char, string>
        {
            { 'X',"F[+X][-X]FX"},
            {'F',"FF" }
        };
        currentLSyst.text = "L-System 5";
        TreeGenerate(7, 25.7f,0.1f);
    }
```

```
    public void LSys6()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 6;
        Camera.transform.position = new Vector3(0, 15, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "X";
        rules = new Dictionary<char, string>
        {
            { 'X',"F-[[X]+X]+F[+FX]-X"},
            {'F',"FF" }
        };
        currentLSyst.text = "L-System 6";
        TreeGenerate(6, 22.5f,0.1f);
    }

    public void LSys7()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 7;
        Camera.transform.position = new Vector3(0, 0, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "FX";
        rules = new Dictionary<char, string>
        {
            { 'X',"[-FX]+FX"}
        };
        currentLSyst.text = "L-System 7";
        TreeGenerate(7, 40, 1f); ;
    }

    public void LSys8()
    {
        if (SliderHolders.activeInHierarchy == false)
            SliderHolders.SetActive(true);
        color = 8;
        Camera.transform.position = new Vector3(0, 0, -30);
        transformStack = new Stack<TransformInfo>();
        axiom = "F+XF+F+XF";
        rules = new Dictionary<char, string>
        {
            {'X',"XF-F+F-XF+F+XF-F+F-X" }
        };
        TreeGenerate(2, 90, 0.3f);
        currentLSyst.text = "L-System 8";
    }

    void TreeGenerate(int iterations, float angle, float length)
    {
        BranchCleaning();
        currentString = axiom;
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < iterations; i++)
        {
```

```
            foreach (char c in currentString)
            {
                sb.Append(rules.ContainsKey(c) ? rules[c] : c.ToString());
            }
            currentString = sb.ToString();
            sb = new StringBuilder();
        }
        foreach(char c in currentString)
        {
            switch(c)
            {
                case 'F':
                    Vector3 initialPosition = transform.position;
                    transform.Translate(Vector3.up * length);
                    GameObject treeSegment =
Instantiate(BranchObject[color],BranchingParent.transform);
                    treeSegment.GetComponent<LineRenderer>().SetPosition(0,
initialPosition);
                    treeSegment.GetComponent<LineRenderer>().SetPosition(1,
transform.position);
                    break;

                case 'X':
                    break;

                case '+':
                    transform.Rotate(Vector3.forward * angle);
                    break;

                case '-':
                    transform.Rotate(Vector3.back * angle);
                    break;

                case '[':
                    transformStack.Push(new TransformInfo()
                    {
                        position = transform.position,
                        rotation=transform.rotation
                    });
                    break;

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

                default:
                    throw new InvalidOperationException("Invalid L-Tree operation");

            }
        }
        angleSlider.value = angle;
        angleText.text = "Angle : "+angle.ToString("00.0");
        iterationSlider.value = iterations;
        iterationText.text = "Iterations : " + iterations.ToString();
```

```
        lengthSlider.value = length;
        lengthText.text = "Length : " + length.ToString("0.0");
    }

    public void OnSliderValueChange()
    {
        float newAngle = angleSlider.value;
        int newIteration = ((int)iterationSlider.value);
        float newLength = lengthSlider.value;
        TreeGenerate(newIteration, newAngle, newLength);
    }
    void BranchCleaning()                                                    //To
destroy all the child branches before creating new branches and resetting the position
and rotation
    {                                                                        //of
tree spawner
        for (int i=0;i<BranchingParent.transform.childCount;i++)
        {
            Destroy(BranchingParent.transform.GetChild(i).gameObject);
        }
        transform.position = new Vector3(0, 0, 0);
        transform.rotation = Quaternion.Euler(0, 0, 0);
    }
}
```

Main Structure

The flow of the program is like this –

1. When we start the program, the user will see 8 buttons with texts mentioning the L-Systems number.

2. The user can click on any button to generate the pre-defined L-System or he can use the number keys 1 – 8 from the keyboard to do the same.

3. Once the user has generated a L-System, there will be 3 sidebars In the top right corners.

4. First one will be of angle, through which the user can modify the angle and can see the respective output accordingly.

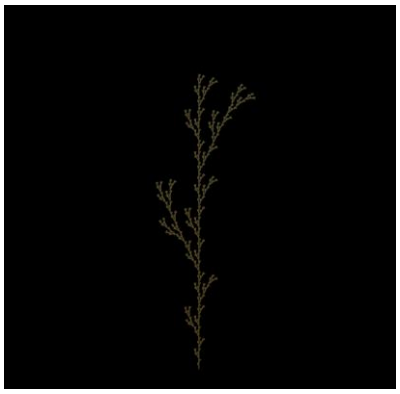
5. Second one is of iterations, where the user can modify the number of times the rules are called and can see the respective output accordingly

6. Third one is of length, where the user can just increase/decrease the length of the L-System as it seems fit.

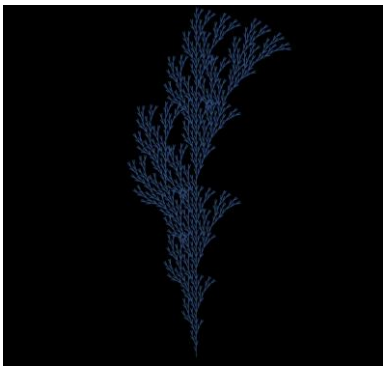
Results

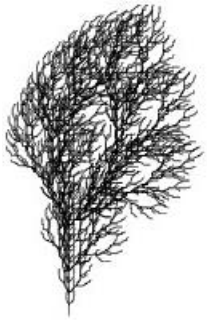
The program I have created allows the user to modify the L-Systems by angle, iterations or length through user interface. I believe this to be a successful example of modifying a L-System with certain parameters. To assure it was rendered correctly, I referred the assignment for first six L-Systems.

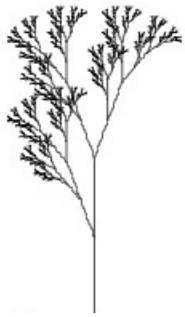
The results are as follows (The black background renderings are mine)-

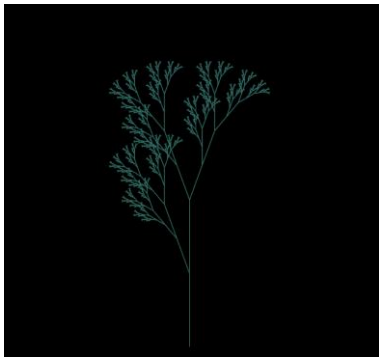


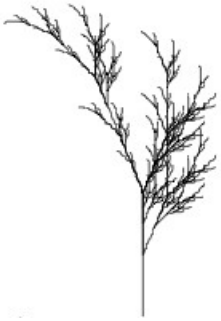
L-System Generation and
Modification



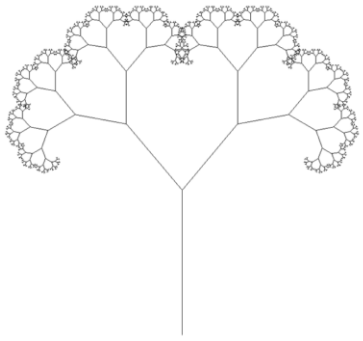


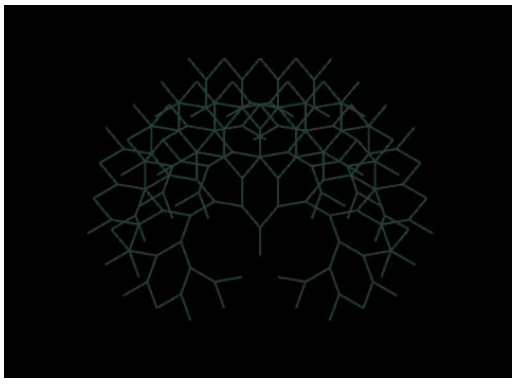




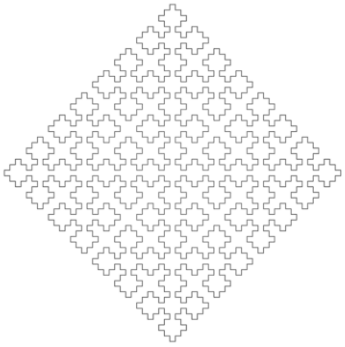


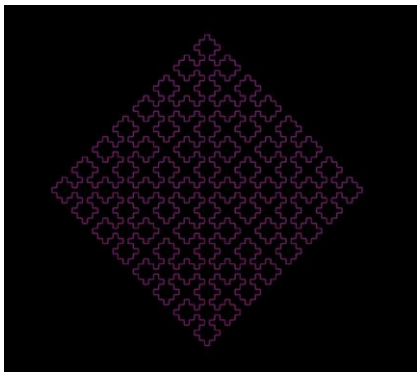
For the last 2 rendering, I am taking referencing from **L-System User Notes, written by Paul Brouke**





L-System Generation and
Modification





Conclusion

In conclusion, I have created a stable L-system modification program with 3 parameters. All my l-systems have been rendered using Unity's Line Renderer. An alternative to this could have been implementing 3D shapes in places of lines, perhaps ones that adapt to their circumstances to create a more complex model. One approach to this could have been to use cuboids that are scaled down the further they get from the axiom point.

Overall, I have created a program that allows the users to modify the parameters of the L-System to get a better understanding of the L-system.

References

Bourke, P. (1991, July). *L-System User Notes*. Retrieved from paulbourke.net:
http://paulbourke.net/fractals/lsys/

L-Systems Unity Tutorial [2018.1]

https://www.youtube.com/watch?v=tUbTGWl-qus

Jennings, C. G. (2017, July 28). *Lindenmayer Systems*. Retrieved from cgjennings.com:
https://cgjennings.ca/articles/l-systems/

McInerny, A. (2015). *Generating a Forest with L-Systems in 3D*. Retrieved from csh.rit.edu:
https://www.csh.rit.edu/~aidan/portfolio/3DLSystems.shtml

Santell, J. (2019, December 09). *L-Systems*. Retrieved from jsantell.com:
https://jsantell.com/l-systems/