L-System Tree

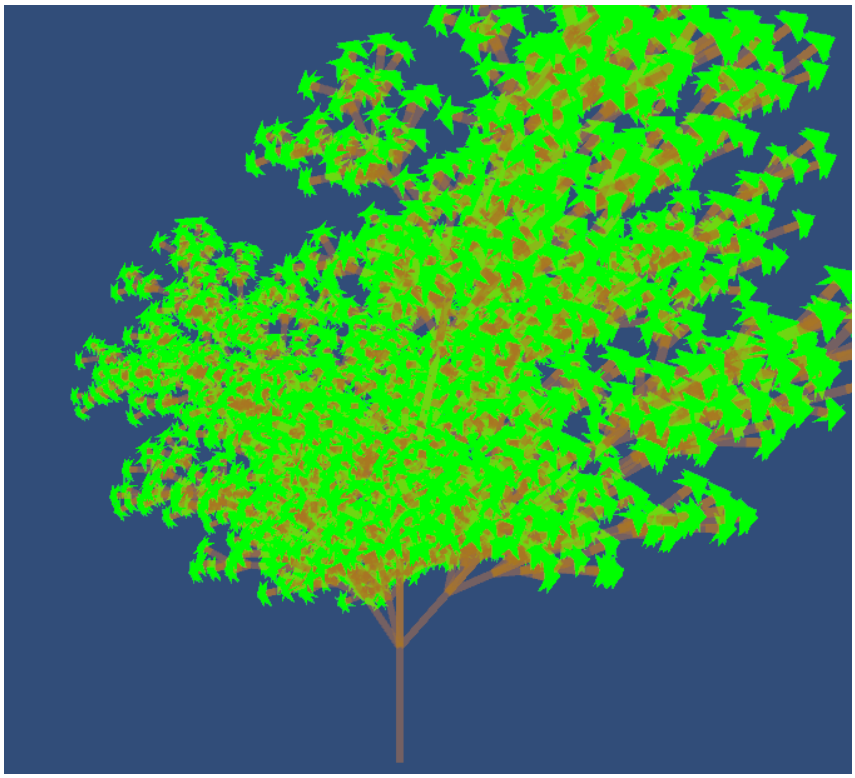
Sang Wu

Msc Computer games programming

IS71021C: Basics of Mathematics for Games and VR/AR

2022/11/24

https://youtu.be/o_s-4PO61Vw



Abstract

In this assignment, I try to make an L-system renderer using line renderer. Users can edit some part of L-system tree such as the width, length and angle of the branches and leaves, and make rules by themselves. Uses can also use rules that I already set to generate trees. Both 2D and 3D trees can be generated successfully. This assignment is made by C# in Unity 3D.

Keywords: Line Renderer, unity 3D, L-system

Features

This system has two features, one is DIY trees by users own, one is that of loading preset models. Figure 1 and figure 3 are my UI in this program.

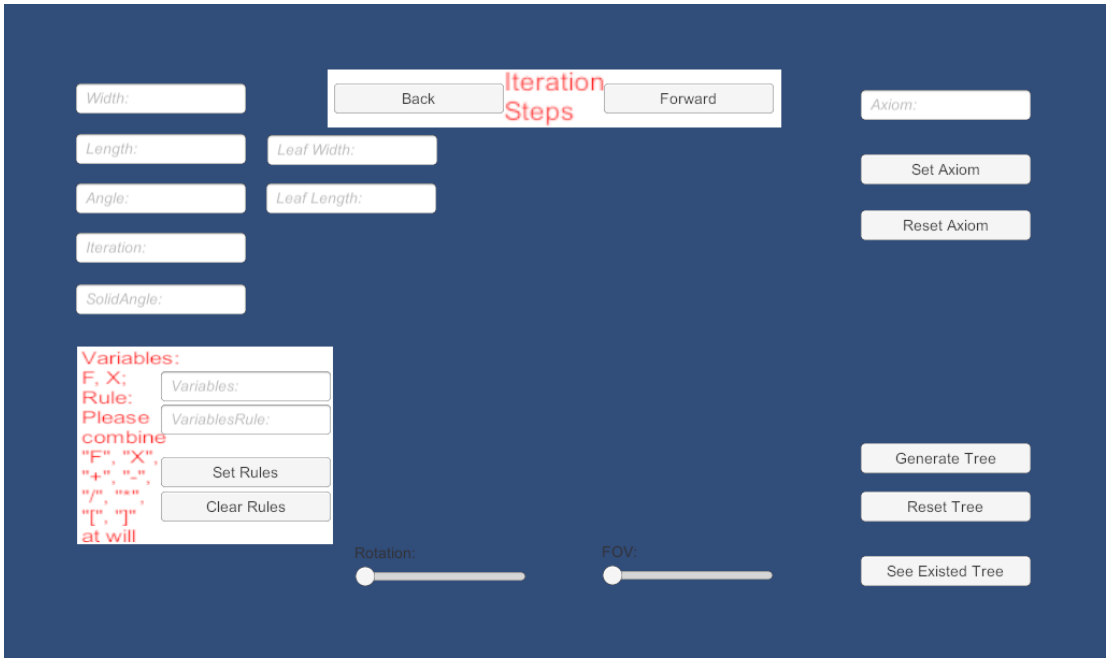


Figure 1 The menus of DIY trees

First, this scene (Figure 1) allows players to make L-system trees by themselves. Players should enter some parameters in the text box. These parameters include the width and length of branches and leaves, the angle and the solid angle that can let the branches and leaves grow in a different direction in world space, the rules and axiom that define the growing way, and the iterations that

can be manually stepped (or clicked) through. Players can also slide the slider below and press the middle mouse button to control the camera lens. The rules can only include those characters that are qualified on the screen, and it has a certain format: in variables text box, players can input “F” and “X” (you can also input other variables, and the number of variables can be more than two , but please make sure that you set the reasonable rules), and in variables rules text box, players can input different combinations of “+”, “-“, “*”, “/”, “[“, “]” and variables that they already set. The program will generate different trees according what players input. Figure 2 are some examples using different rules:

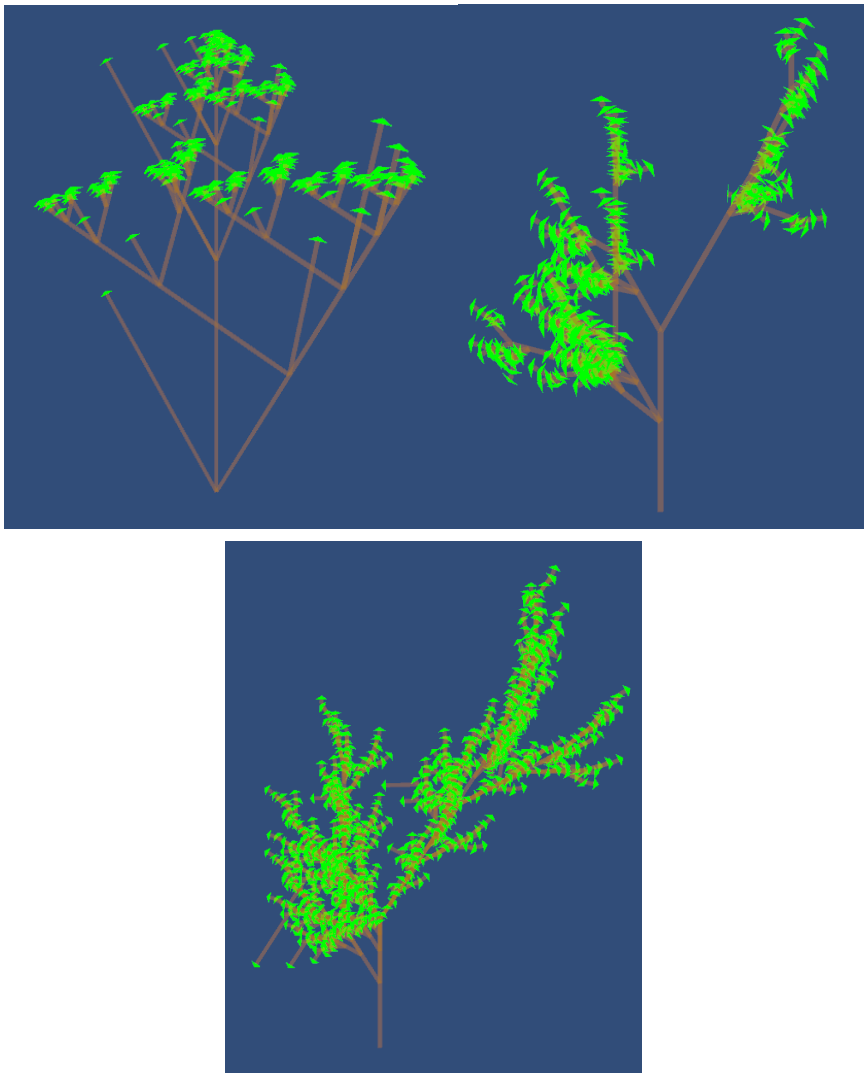


Figure 2 Trees using different rules

As it shows, finding a correct rule to generate a regular and beautiful tree is difficult. There may be

only a few rules that can define regular trees.

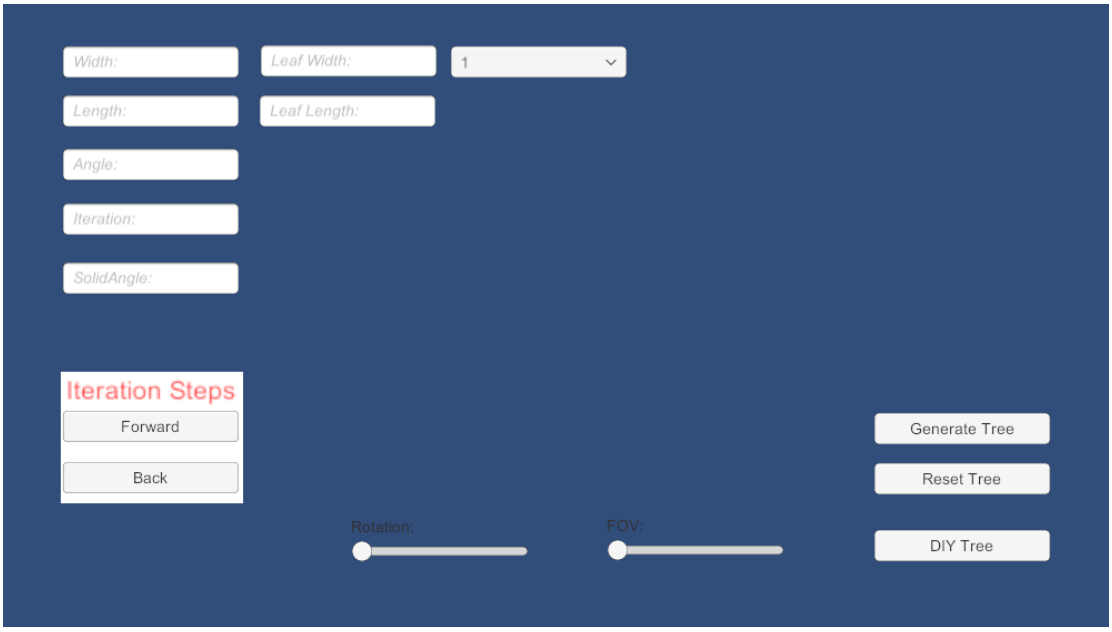


Figure 3 The menus of setting trees using existed rules

Second, if players do not want to set rules, they can click the “See Existed Tree” button to see 9 different trees I already set (Figure 3). The trees are shown at figure 8 and below:

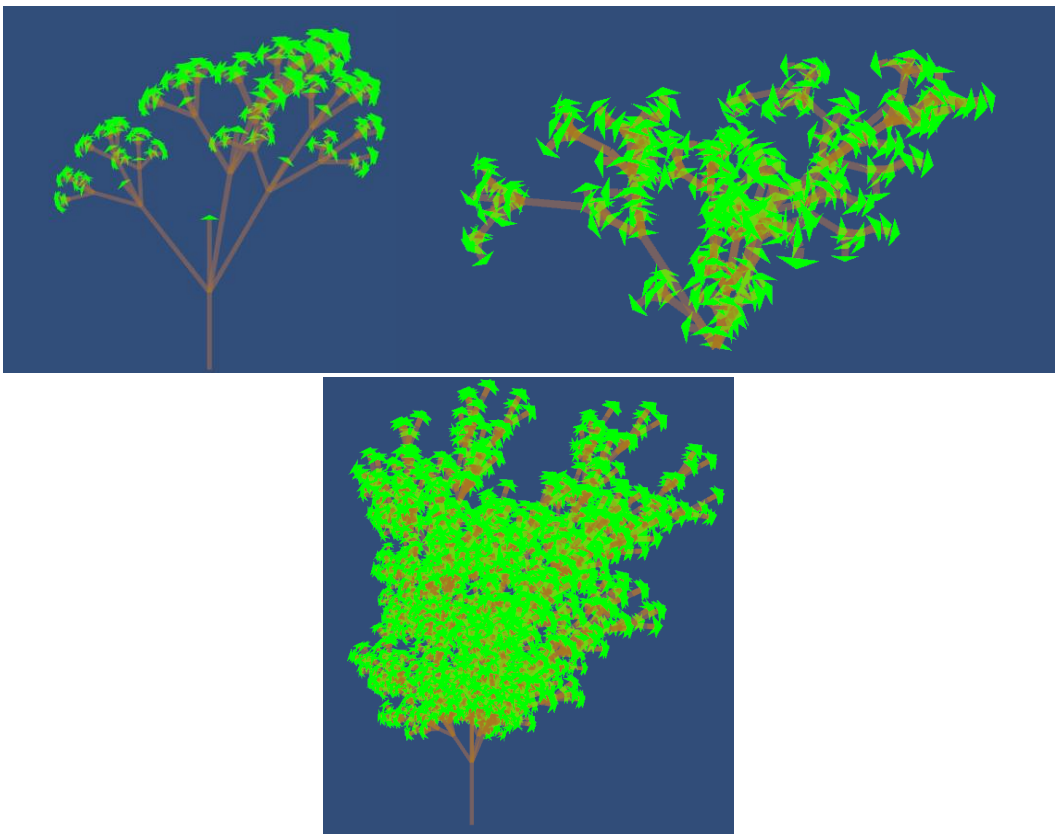


Figure 4 Trees using rules 7, 8 and 9

In this part, players can also set and change some parameters relating to the shape of trees.

Data Classes

I use three custom data classes in my scripting, including a data class to store positions and rotations, a data class to control the camera and a date class for the user-generated variables and constant data.

```
public class GenerationLTree : MonoBehaviour
{
    public GameObject TreeParent;
    public GameObject Branch;
    public GameObject Leaf;
    private GameObject Tree;
    private GameObject TreeSegment;
    private GameObject Leaves;

    private string Axiom;
    private string CurrentString = string.Empty;
    private string Rules_Key;
    private string Rules_Value;

    private float width;
    private float length;
    private float leafwidth;
    private float leaflength;
    private float angle;
    private float solidangle;
    private int iteration;
    private Color StartColor = new Color(169.0f / 255.0f, 118.0f / 255.0f, 32.0f / 255.0f, 100.0f / 255.0f);
    private Color EndColor = Color.green;

    List<GameObject> TreeSegments = new List<GameObject>();
    List<GameObject> LeavesList = new List<GameObject>();

    private Dictionary<char, string> rules;
    private Stack<TransformInfo> transformStack;

    [SerializeField] private InputField IAxiom;
    [SerializeField] private InputField IWidth;
    [SerializeField] private InputField ILength;
    [SerializeField] private InputField ILeafWidth;
    [SerializeField] private InputField ILeafLength;
```

```
[SerializeField] private InputField IAxiom;
[SerializeField] private InputField IWidth;
[SerializeField] private InputField ILength;
[SerializeField] private InputField ILeafWidth;
[SerializeField] private InputField ILeafLength;
[SerializeField] private InputField IAngle;
[SerializeField] private InputField ISolidAngle;
[SerializeField] private InputField IIteration;
[SerializeField] private InputField IRules_Key;
[SerializeField] private InputField IRules_Value;

public Button GenerateTree;
public Button ResetTree;
public Button GenerateAxiom;
public Button ResetAxiom;
public Button GenerateRules;
public Button ResetRules;
public Button StepBack;
public Button StepForward;
public Button Come2EightTree;
public Slider FOVS;
```

Figure 5　The data class that stores variables and content data

Firstly, this data class (Figure 5) will monitor all the data players entered and pass them into the script. The variables contained here will be added to a C# list. Then a variable will be set as beginning string in current string. System will check this variable to see whether it matches the key in the dictionary. If so, this variable will pass the dictionary value information into current string. If not, this variable will be stored as content data class or just be skipped directly. Every iteration the system will check the variables in current string to see whether them match the key in dictionary and do the same thing above. In this script, it also contains preset data class. Users can input some strings and numbers into script through UI, and it expands the variables and constant data class by storing all necessary data. So the constant data come from users' input.

```
public class CameraController : MonoBehaviour
{
    4 个引用
    public enum MouseState
    {
        None,
        MidMouseBtn,
        LeftMouseBtn
    }
    private MouseState mMouseState = MouseState.None;
    float _mouseX = 0;
    float _mouseY = 0;
    public float moveSpeed = 1;

    public Slider rotationSlider;
    public float lastValue;
```

Figure 6 The data class that controls the main camera

Secondly, this data class (Figure 6) controls the movement and rotation and FOV of main camera.

```
public class TransformInfo
{
    public Vector3 position;
    public Quaternion rotation;
}
```

Figure 7 The data class that stores position and Quaternion

Finally, this date class (Figure 7) contains a Vector3 to store the position and a Quaternion to store the rotation. In my main script, a new stack of this data class is defined and pushed into it to be popped when it comes to letter “]”. A stack allows data to return to the latest entry in the list each time.

Main Structure

The main structure of this program’s working way is:

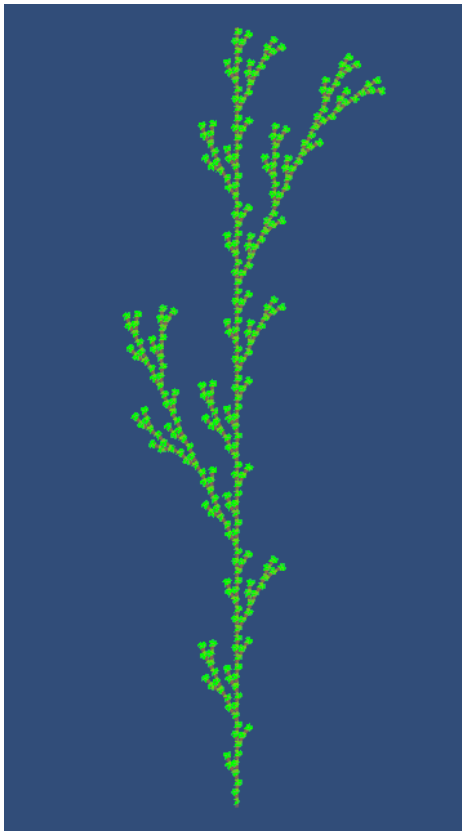
1. The user defines the angle and iterations and any other parameters and settings under main screen. The user fills the rules fields and axiom fields as desired using letters and characters defined in the variables and constants panel. There are some hints shown in screen that tell the user which he or she should input.
2. The user clicks all buttons related to generate something and then the system will pass these texts into script as a dictionary’s content or variables. Dictionary enables a rule field for each of the defined variables.
3. The user clicks the “Generate” button. Then the system uses the defined rules in the dictionary.
4. For each iteration, the system will process the string (starting with the axiom string) according to the rules in the dictionary. The system will traverse the characters in the string and compared them with keys in dictionary. If the system encounters a key of dictionary, the system will parse the key into the dictionary and replace it with the retrieved string. Otherwise, the system will convert this character into string type and store it directly.
5. Then the function converts the string supplied to it into an array of characters. The system will go down the array step by step and check if the character here is a variable of dictionary. If it is, the system will proceed to the fourth step. If it is not, but can match the cases in switch part of the script, the system will handle it differently according to the cases. If it is not, and also cannot match the case, the system will skip this character. The way handling it in 3D system is totally the same with it in 2D system. By the way, system will define leaves or branches in this step by finding the closest “F” near the left of char “]”.
6. Then the system enables all the lines at once, rendering the L system model. The users now can

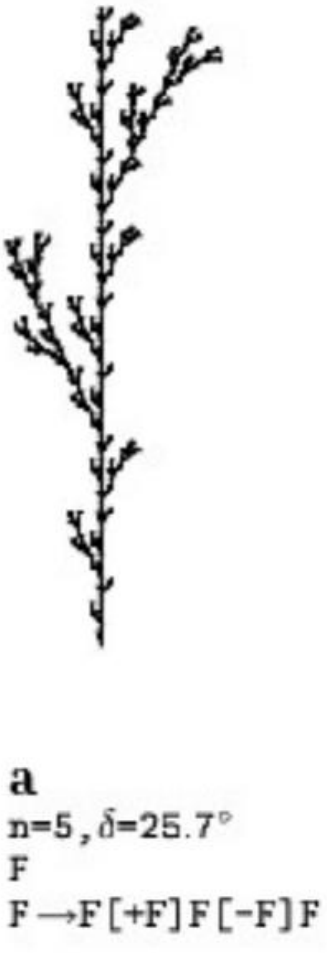
acquire a rendered model defined all by themselves through the user interface. They can continue to affect the model by changing the rules, variables and constants at will.

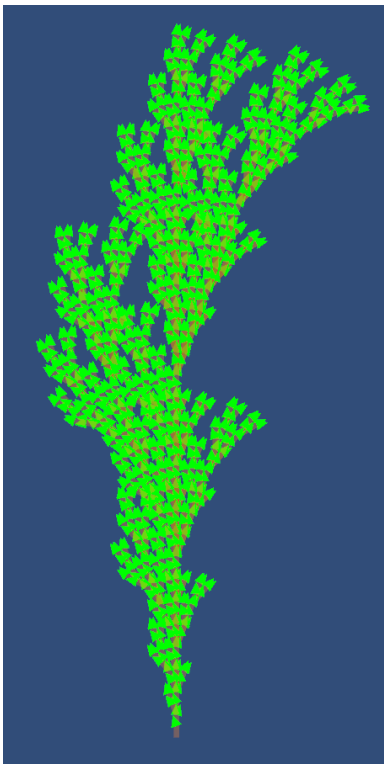
Results

This program allows users to generate a tree which is totally rendered by lines. Without any model, users can also get regular graphics at will. Users do not have to know the principles behind the system, because all operations can be done through the user interface.

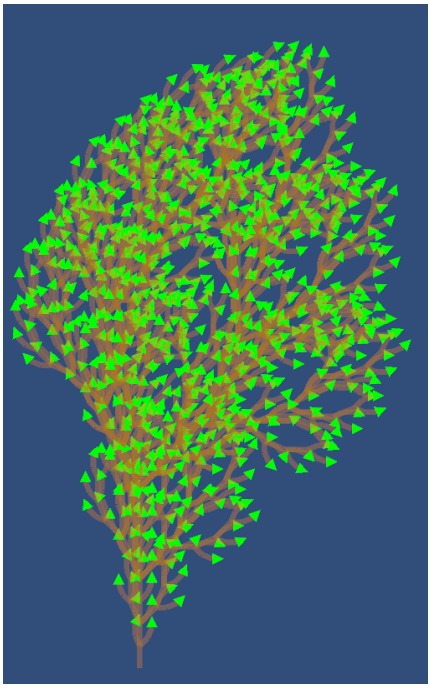
To ensure the accuracy, I put my results (on the left) there compared with reference trees (on the right):

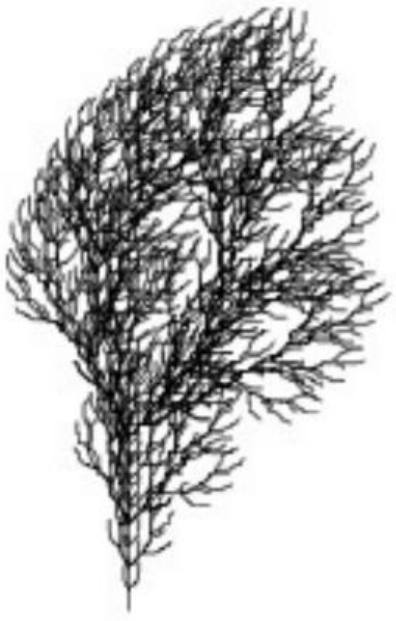




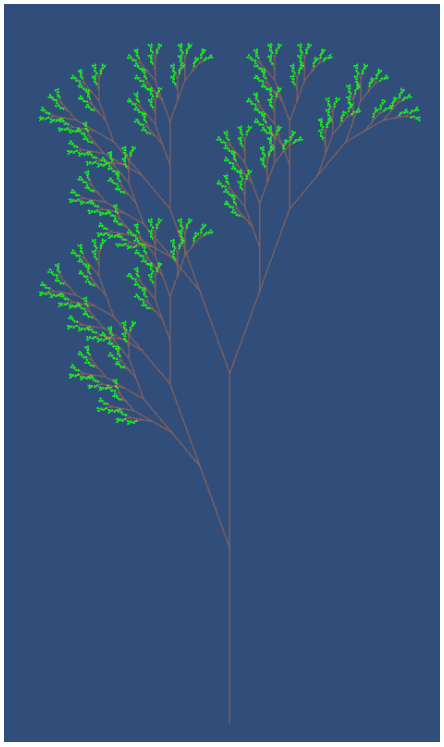


b
n=5, δ=20°
F
F → F[+F]F[-F][F]





c
n=4, δ=22.5°
F
F → FF-[-F+F+F]+
[+F-F-F]



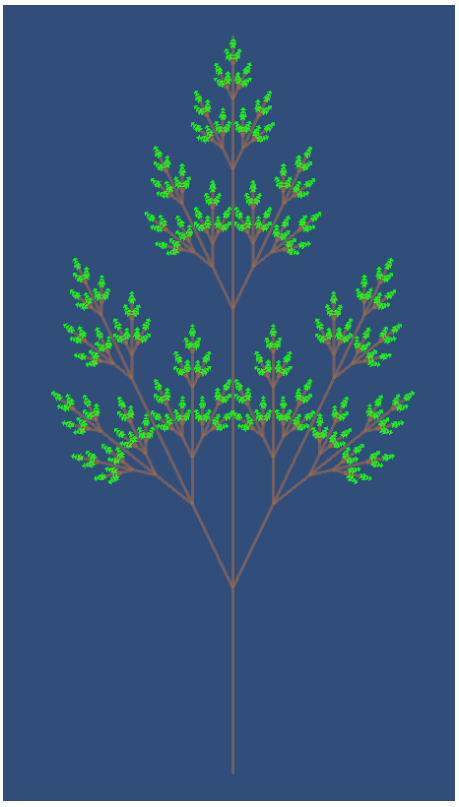
d

n=7, δ=20°

X

X → F[+X]F[-X]+X

F → FF



e

n=7, δ=25.7°

X

X → F[+X][-X]FX

F → FF

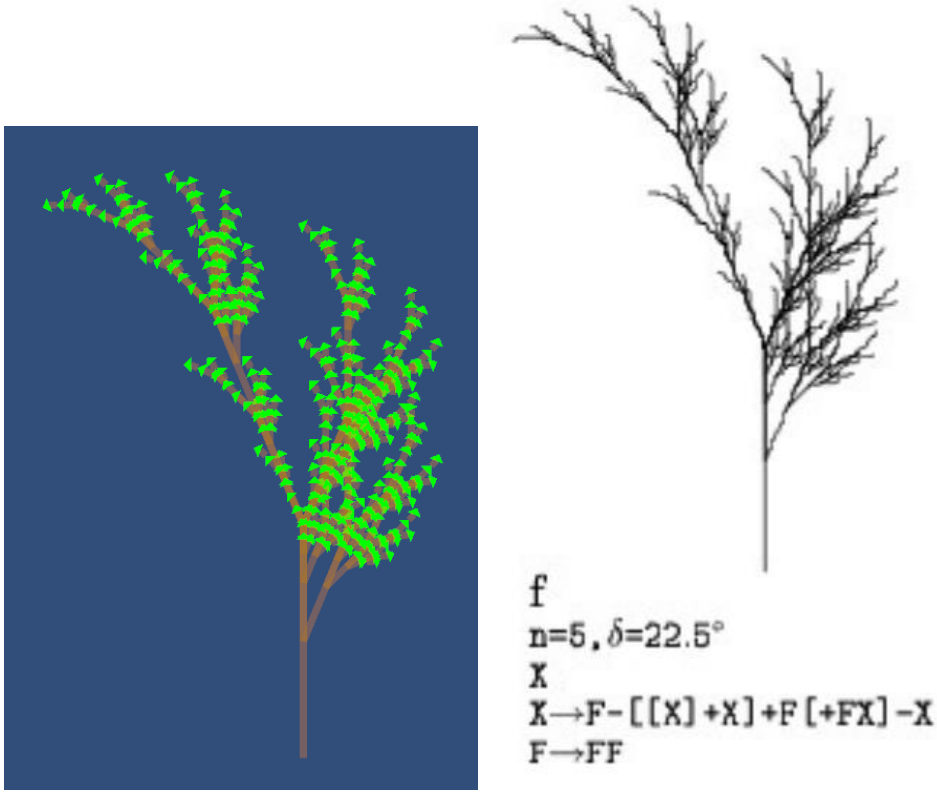


Figure 8 My works compared with reference works

As it shows, this system does a good job of performing the most basic functions. What's more, it shows the position of leaves clearly in every tree.

Conclusion

In conclusion, I have made a rough L system, and there are still some aspects that could be improved. First, I can write shader and material to display different type of trees, like cherry blossom trees or pine trees, and these settings can also be edited by users. Second, the line renderer component allows coder to set shapes of leaves and branches, but I just set end width and start width simply. Third, the way to generate leaves should be more complex, and I should add more random point inside system. Fourth, the UI can be more user-friendly, such as replace the input fields with sliders. Last but not least, a more realistic system should consider much more detailed control of the model, such as some individual random rotations and movements.

In fact, I already wrote the material code in my script. Next time I just need to write more variables that can be input in script and pass the material into it directly. I will continue to study making more

realistic trees.

References List

P. Prusinkiewicz, A. Lindenmayer. (1990) *The Algorithmic Beauty of Plants:* Available at: http://algorithmicbotany.org/papers/abop/abop.pdf (Accessed on: 1990).

Bourke, P. (1991) *L-System User Notes:* Available at: http://paulbourke.net/fractals/lsys/ (Accessed on: July 1991).

Santell, J. (2019) *L-Systems:* Available at: https://jsantell.com/l-systems/ (Accessed on: 09 December 2019).

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

Pete, P. (2018) *L-Systems Unity Tutorial [2018.1]:* Available at: https://github.com/pejoph/L-Systems (Accessed on: 24 Jan 2018).

Hiestaa. (2014) *3D-Lsystem:* Available at: https://github.com/Hiestaa/3D-Lsystem/tree/master/lsystem (Accessed on: 30 Jan 2014).