Priyadharshni Krishnan

(Student ID: 33681281)

MSc in Virtual and Augmented Reality

Goldsmiths, University of London

IS71021C : Mathematics and Graphics for Computer Games 1

(2020 – 2021)

Coursework 1 : L-Systems

27-Nov-2020

Links to project code and executables

1. GitHub Link to Project Source Code : https://github.com/priyacodes/Lsystem
2. WebGL Link to run app : https://priya13.itch.io/l-systems?password=goldsmiths
3. Youtube Video : https://youtu.be/71rb6dDor6o
4. Links to download output files : https://github.com/priyacodes/Lsystem/releases

Contents

Introduction to L-Systems

L-Systems are a category of fractals, which was introduced by a biologist named 'Aristid Lindenmayer' in the year 1968 to showcase the growth of cells in plant structures. It was later expanded to create complex structures of trees including branches, leaves, etc.

The primary concept of L-Systems is rewriting. The L-System is governed by a set of grammar which is responsible for generating the model, by the principle of successively replacing a complex object by another simple object using a set of rules. The concept of L-Systems is similar to that of Chomsky's grammar, but they mainly differ by the method of rewriting. L-system uses parallel rewriting where the characters are replaced in parallel when a rule is applied.

The following are the components of a L-System grammar:

i. **Alphabets** – A set of characters or letters for which rules are defined, and is part of any valid sentence of the L-System.
ii. **Axiom** – A character or set of characters that form the initial state of an L-System.
iii. **Production Rules** – Rules are definitions for successive replacements of the characters to generate a sentence. At first, they are applied to the axiom to generate a string of characters, and later on they are recursively applied to the string to generate the final sentence.
iv. **Sentence** – The final string after recursively applying the rules.
v. **Graphical Transformation Mapping** – Graphical rule definitions for each character or symbol in the sentence.

Sample L-System Grammar:

Alphabets : F

Axiom : F

Production Rules:

F $\longrightarrow$ F[-F]F[+F][F]

Graphical Instructions:

F :	Draw a line
+ :	Turn left by 20 degrees
- :	Turn right by 20 degrees
[:	Push position in stack
] :	Pop position from stack

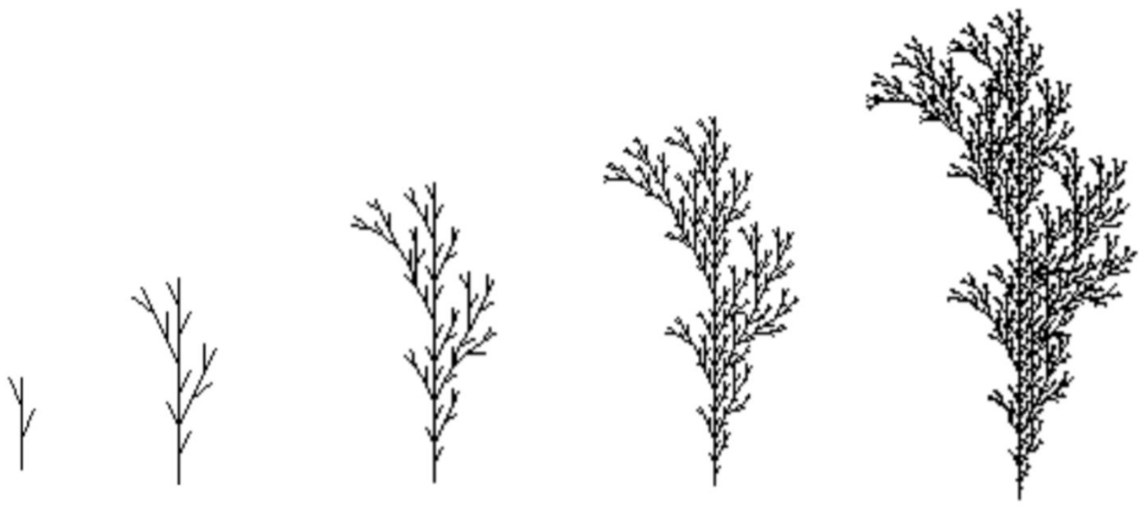


Fig (1) L-System tree generated by the rules for 5 iterations

Objectives

The main aim of the project is to read a list of configuration files consisting of L-System grammars, and generate the L-System trees using line renderers in Unity 3D. To implement a graphical user interface serving as a hotkey for selecting the choice of L-System and also consisting of options to modify the L-System parameters such as angle, iterations (number of generations), length of the tree. An option to view the generated L-Systems in grayscale or colored mode.

Features of the Project

Softwares and Language used for the project:

- Unity 3D – For technical development
- Paint 3D – For creating UI assets
- Scripting using C# language

A simplistic user interface is created for the user to interact with the application and visualize the L-System trees generated. The UI consists of buttons as hotkeys for the tree selection. The thumbnails of the buttons are embedded with sprites of the generated L-Systems (Fig. 3.2), which acts as a map legend and gives the user a visual cue of the output. The UI also consists of text labels which are used for displaying the details such as the axiom, characters, production rules, etc., from the configuration file of the generated L-System. There are options to modify the parameters (such as angle, iterations) of the L-systems, in the form of the UI sliders, such that the user can see the generated L-system being dynamically modified based on the interaction with the slider.

The L-System trees are generated in two modes: Colored and Grayscale. The colored mode showcases the trees with dual colors namely brown and green, denoting branches and leaves respectively. Whereas, the grayscale shows the trees in monotone with emissive lighting. The user can select the mode of viewing in the UI using a toggle button.

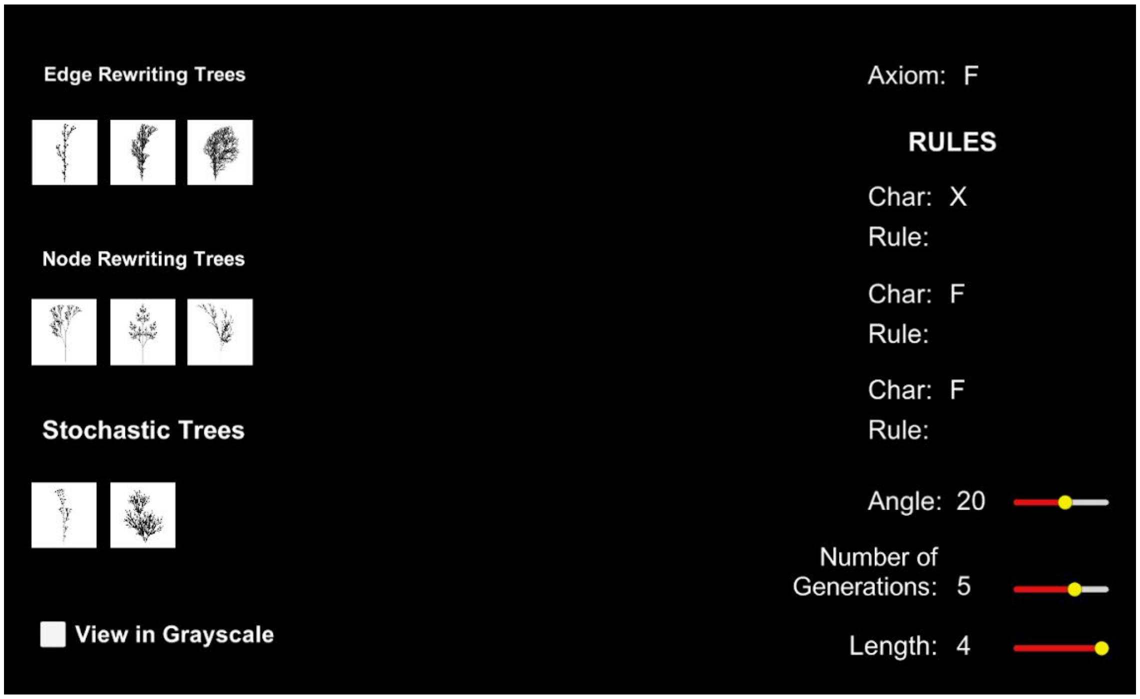


Fig. (3.1) User interface of the application

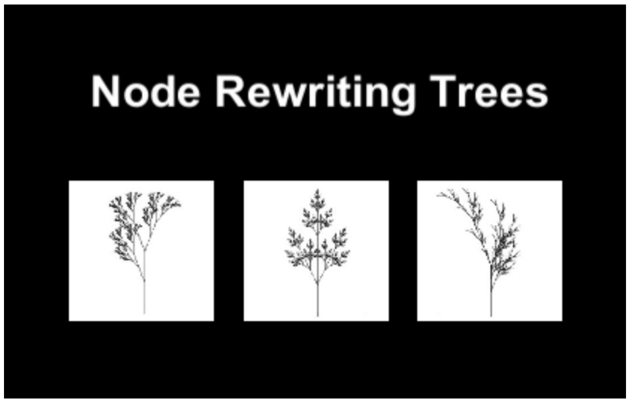


Fig. (3.2) Buttons with thumbnail images of the L-Systems

Types of L-Systems:

The L-Systems generated in this application fall under the category of bracketed OL-Systems. Deterministic context-free L-systems (DOL) and Stochastic L-Systems are visualized in the application. **Deterministic** L-Systems are the simplest class of L-Systems where there is **exactly one production** rule for each symbol, whereas the **stochastic** L-Systems contains **several production** rules for a symbol and each rule has an associated probability per iteration.

The application has the option to generate eight L-Systems classified into the following:

1. Edge Rewriting Trees	- Production replaces figures for the edges
2. Node Rewriting Trees	- Production replaces figures for the vertices
3. Stochastic Trees	- Multiple rules for realistic generations

The number of iterations and the length of the L-System trees, vary for each of the configurations so as to match the size of all the trees with the available screen resolution. However, the user is free to interact with the sliders in the UI to alter these values and see the changes in the L-System. For example, if the preset value of generations in a configuration file is 5, then a tree is generated with five generations and the user can decrement the value using the slider to see the preceding generations of the L-System.

The details of the configuration file including the axiom and productions are displayed along with the generated tree. By default the tree is generated in a colored mode and the toggle button can be used to view the tree in grayscale.

Technical Implementation

The application is developed using C# language. The key technical concepts used in the generation of the L-Systems are recursion, dictionaries and vectors. Dictionaries are used to store the production rules of the configuration files. The concept of recursion is used in the generation of the sentence string which is responsible for creating the model of the L-System. Line renderers are used to model the components. The vectors are then used to define the values for these line renderers in terms of transform, angle and length.

Classes Used :-

```
public class PresetTrees

{

    public int treeID;

    public char axiom;

    public float angle;

    public int generations;

    public float length;

    public Dictionary<char, string>

                 rules;

}
```

```
public class StochasticTrees

{

    public int treeID;

    public char axiom;

    public float angle;

    public int generations;

    public float length;

    public Dictionary<char,

        StochasticRules> Stocrules;

}


public class StochasticRules
{

public string subRule1,subRule2;

}
```

The script consists of two different classes: ‘*PresetTrees*’ for the deterministic L-Systems and the ‘*StochasticTrees*’ for the stochastic L-Systems. Both the classes are very much similar in terms of its variables, but differ in the dictionary types used for storing the production rules.

The ‘PresetTrees’ class has the dictionary type set as <char,string> where the ‘char’ key is the variable in the production and the ‘string’ type is the value which replaces the variable when the L-System string is generated.

Below is an example of how the configuration file is defined using the PresetTrees class:

```
PresetTrees presetTree;  // declare a variable

presetTree.treeID = 0;
presetTree.axiom = 'F';
presetTree.angle = 25.7f;
presetTree.generations = 5;
presetTree.length = 1;
presetTree.rules = new Dictionary<char, string>
{
   { 'F', "F[+F]F[-F]F" }    // Production Rule
};
```

Production Rule: F → F[+F]F[-F]F

The above rule is incorporated in the dictionary in the form of character and string where every time the character ‘F’ is encountered it is replaced with the string expression ‘F[+F]F[-F]F’. Notice that there is only one character in the grammar and there is only one production rule pertaining to it, hence, this falls under the category of deterministic L-systems (edge writing).

Below is an example of how the configuration file is defined using the StochasticTrees class:

```
  StochasticTrees stochasticTree; // declare a variable

  stochasticTree.treeID = 6;
  stochasticTree.axiom = 'X';
  stochasticTree.angle = 20f;
  stochasticTree.generations = 5;
  stochasticTree.length = 4;
  // Using a class of two strings for multiple productions
  stochasticTree.Stocrules = new Dictionary<char,StochasticRules>
   {
          {'X', new StochasticRules{ subRule1 = "F[+F]F[-F]F",
subRule2 = "F[+F]F[-F]F" } },

          {'F', new StochasticRules { subRule1 = "F[+F]F", subRule2
= "F[-F]F" } }

    };
```

Production Rule: X → F[+F]F[-F]F

F → F[+F]F

F → F[-F]F

The above production is of a stochastic L-System tree. It consists of two characters ‘X’ and ‘F’, and there are two productions for the character ‘F’. Hence, the dictionary type uses another class named *StochasticRules* consisting of two strings namely subRule1 and subRule2, which holds the productions of F. During the sentence generation, there is a 0.5 probability for each of the sub rules to be chosen. Since the dictionary uses the concept of multi string, the same production rule is keyed into both subRule1 and subRule2 corresponding to the character ‘X’. The rules are picked using Unity’s randomisation function - Random.Range(0, 2) where 0 picks subRule1 and 1 picks subRule2.

String builder is used to generate the sentence string of the L-System. Depending upon the number of generations in the configuration file, the axiom is replaced and appended with the production recursively to generate the final sentence. This sentence is then used to model the L-system based on the graphical instructions.

The graphical execution is done using instantiation of prefabs of line renderers and the angle of rotations are calculated using vectors. Each character from the sentence is fetched to compare with the list of graphical instructions. For example when the character ‘F’ is encountered in the sentence it denotes a line to be drawn. Hence, a line renderer is instantiated. Based on the characters preceding and succeeding the current read character, it is decided whether the current prefab to be instantiated is a branch or a leaf.

```
newLine = currentString[i % currentString.Length] == 'F' &&
currentString[(i + 1) % currentString.Length] == ']' &&
(currentString[(i - 1) % currentString.Length] == '+' ||
currentString[(i - 1) % currentString.Length] == '-') ?
Instantiate(leaf) : Instantiate(branch);
```

If the read character is ‘F’, succeeded by ‘]’ and preceded by ‘+’ or ‘-’ then a leaf is created, else a branch is drawn.

Similarly a condition is applied to the sentence for the node writing trees with more than one variable.

Class used for stack operation:

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

The class consists of a 3D vector to store position and quaternion to store the rotation. When the character encountered in the sentence is a ‘[‘ the transform datas are pushed to the stack and when a ‘]’ is encountered, the stack is popped. The angular changes in the L-System are executed using transform.Rotate function, where the Vector3.forward and Vector3.back is used to indicate positive and negative angular deviations respectively.

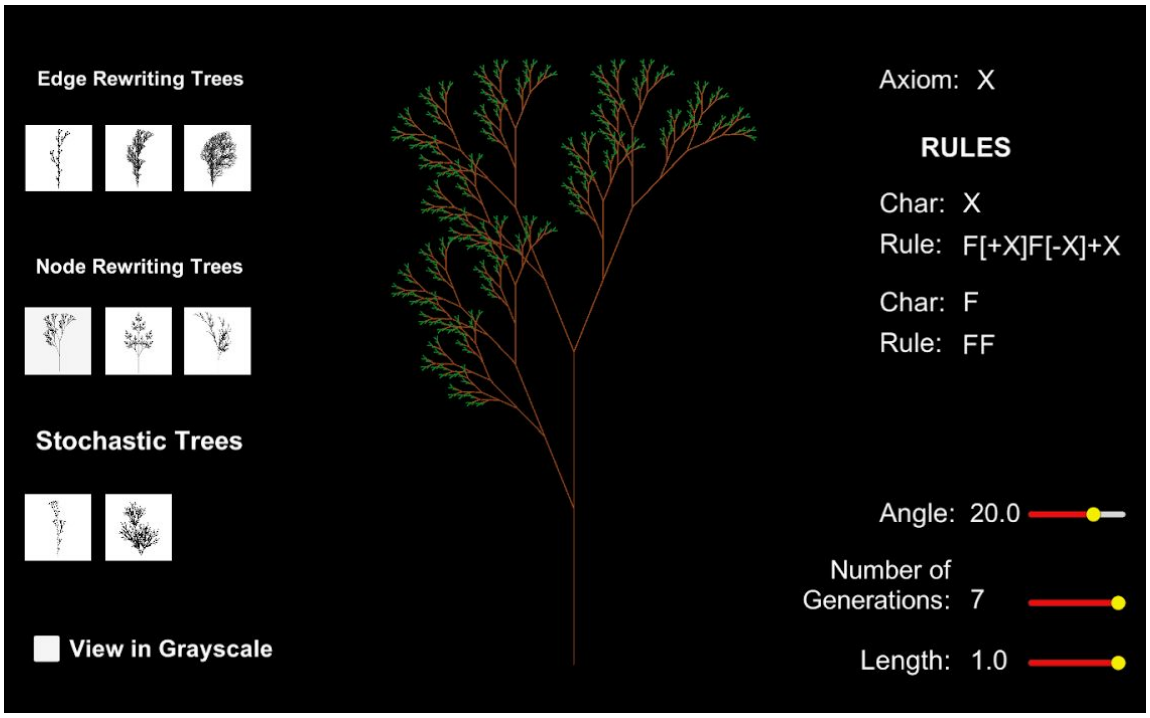
Vector3.forward denotes Vector3(0,0,1) and Vector3.back denotes Vector3(0,0,-1). Since the application is in 2D, the angles are applied along the z-axis only. The angle is multiplied to the vector and is applied to the transform.

The color mode of the L-System is changed in the Grayscale.cs script. Based on the toggle value in the UI, the line renderer is assigned the corresponding material.

Every time a new L-System is created, the current existing L-system tree is destroyed and the transforms are reset to ensure that all the generations are consistent and the performance is not compromised.

Results

The L-Systems are successfully generated with UI buttons acting as hotkeys to select the trees. The generated trees can be altered using the interactive elements in the interface i.e., the sliders, and the outcome can be visualized dynamically. The parameters in the configuration files are mapped to the interactive sliders and the values are assigned as the maximum values of the slider, whereby the user can decrement the values to see the preceding generations of the model. Each time the stochastic tree buttons are clicked, the generated model varies due to randomization of the rules, unlike the other trees (deterministic) which generate the same model every time the hotkey is pressed. The color modes of the L-System trees are successfully executed as well, where the updated tree can also be visualized dynamically.



Axiom: X

RULES

Char: X
Rule: F[+X]F[-X]+X

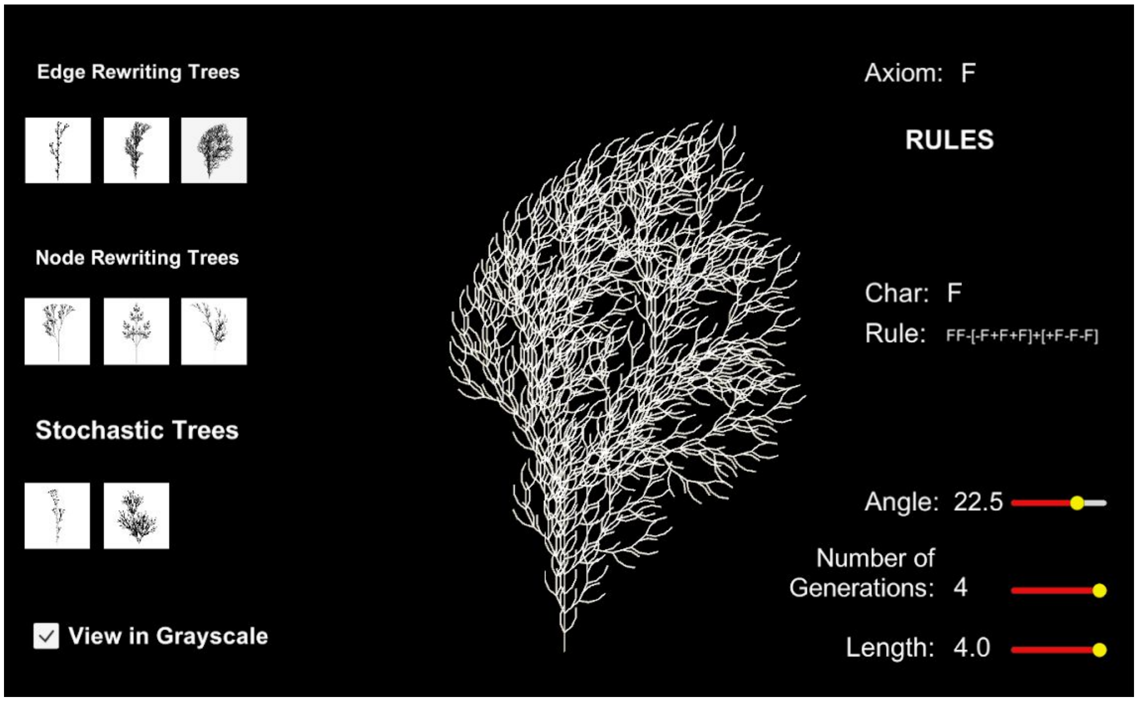
Char: F
Rule: FF

Angle: 20.0

Number of Generations: 7

Length: 1.0

Fig. (5.1) L-System generated in color mode



Axiom: F

Char: F
Rule: FF-[-F+F+F]+[+F-F-F]

Angle: 22.5
Number of Generations: 4
Length: 4.0

✓ View in Grayscale

Fig. (5.2) L-System generated in grayscale mode

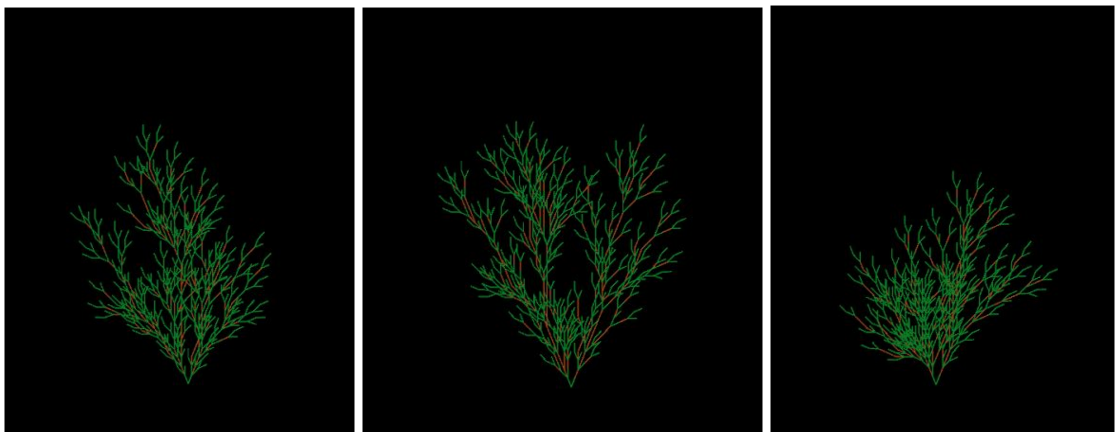


Fig. (5.3) Stochastic L-Systems generated with randomised rules

Conclusion

L-Systems are a great way of implementing fractals, vegetation and anything that can be produced in a recursive fashion. It helps to visualize the objects generated using the concept of recursion. Drawing or modelling the objects manually using recursive rules can be tedious. With the help of the softwares and math techniques, these object generations can be executed in the digital world, thereby reducing the work of manual drawings and helping in visualising the objects generated over multiple recursions. The concept of procedural generation of 3D objects in games and art industry is a clear example of how useful this concept is.

References

Several research papers and blogs have helped me in executing this project successfully. I would like to highlight a few of them here along with the knowledge inculcated.

1. Recognizing Plants Using Stochastic L-Systems [1994] - This research paper has clearly detailed about stochastic L-Systems and their implementations.
2. L-Systems - Simulation of development and growth - A presentation which explains the differences in node and edge rewriting methods.
3. Unity 3D Procedural Generation - Dilmer Valecillos - A great tutorial on implementing L-Systems in Unity 3D
4. L-systems: Draw a stochastic plant (II) - A simple yet clear blogpost on L-System fractals and plants using animated illustrative arts to explain the concepts of stochastic plants.
5. L-System User Notes - Written by Paul Bourke [Version 2.5 - 1995] - A complete collection of L-System examples from various categories such as trees, Koch curves, snowflakes, Kolam (South Indian patterns drawn in front of doorways).

Bibliography

1. Prusinkiewicz, P., Lindenmayer, A. 1990. The Algorithmic Beauty of Plants.
2. Unity L-System Tutorial by Peter Philips
3. http://web.cse.ohio-state.edu/~wang.3602/courses/cse3541-2016-spring/17-tree.pdf
4. Modeling Plants with Lindenmayer Systems - Allen Pike [2007]
5. L-systems : Draw nice fractals and plants (part I)