L-systems based 3D tree modeling

Team 13

KUNAL RATHOD and SARTHAK TYAGI

1 INTRODUCTION

The goal of our project is to model 3D trees using L-systems. Tree modeling is an important part of several virtual reality applications. But trees are difficult to represent because of their geometric complexity and wide variability. The methods used to model trees may be broadly divided into rule-based and image-based methods. Rule-based methods repeatedly build branches and leaves using a set of generative rules. Because of their highly parameterized nature, these approaches provide effective controls over the shape of trees. The goal of image-based approaches is to create very realistic models of trees by utilising photographs from the natural world. Rule-based tree modeling techniques are more appropriate and convenient for applications where the model does not need to resemble a particular image of the actual tree [4].
L-system consists of alphabet symbols, used to make a string/command, an initial string(axiom) for beginning construction and a set of rules for converting strings into geometric structures.
By using the approach discussed in the research papers, the user just needs to specify a few intuitive and descriptive parameters to control the tree's properties. Using this, different trees can be modeled by just varying the required parameters.

2 LITERATURE REVIEW

L-systems were initially introduced as a mathematical model of multicellular organisms. It was used to describe the development of breaching structures using a parallel string rewriting mechanism. Now parametric L-systems extend this original idea of the L-system using numerical parameters to represent plant components. Here each plant module is represented by a letter and different letters represent different types or different states. And a sequence of letters forms a word which represents the entire plant. This extension of parametric

L-systems can be incorporated with the features of programming languages and can provide techniques for creating hierarchical models.

In plant modeling research, L-systems have proven to be successful from a practical perspective. The programs using these systems do not have to be recompiled each time a new plant is simulated. The concept of turtle representation provides a simple method to represent the geometrical information in the form of a string. This parametric L-systems with turtle interpretation as a tool for scientific visualization of development in modular branching organisms.

0L-system is the simplest class of L-systems. Here only lineage mechanisms are used to control the development, this class is context-free, interaction less or zero sided L-systems. The definitions of this systems are as follows:

The 0L system is ordered by the triplet G = <V, ω, P> where V denotes the alphabet, it is a set of symbols containing elements that can be replaced (variables) and elements that can't (constants); ω is a non-empty word called the axiom, it's the initial string from which the construction of the system will begin and P is a finite set of productions, this consists of two strings one predecessor and one successor [7].

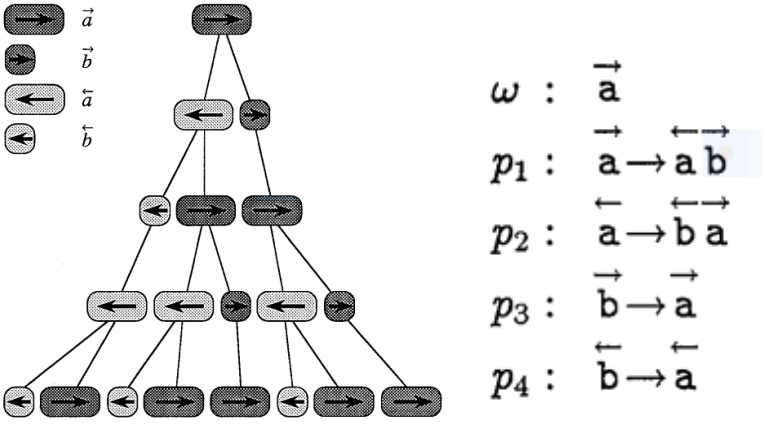


Figure 1

This above diagram shows the development represented. Where p shows the rules for the development.

Another widely used method is Stochastic L-systems. All the plants generated by the above L-system will look the same which is useful when we want a forest or a garden. Stochastic method adds randomness to geometry using a probability factor. Which changes the topology and geometry of the plant.

Stochastic L-system is an ordered quadruple G = <V, ω, P, π>. Where V, w and P are the same as above, π is a probability factor.

There is a notion of bracketed (left “[” and right “]”) strings in L-systems. Which is used in modeling of branching structures. This brackets works the same way parentheses works in arithmetic expressions. Here bracketed strings are considered as delimiters of branches, which can be interpreted as branching structures. In the context free L-systems such as 0L these brackets have no effect but the stochastic system imposes a branching topology on the strings. An example of such branching can be seen in figure 2.

There are other signals passed into these bracketed strings as well like branching angles with other geometric attribute symbols.

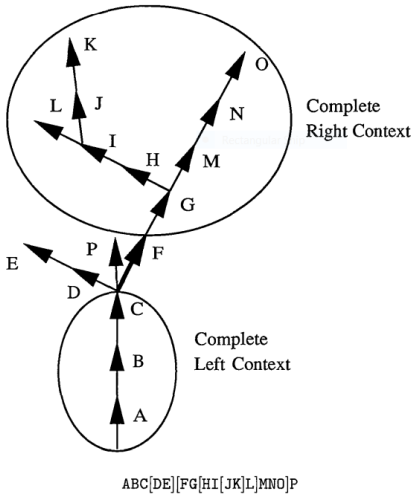


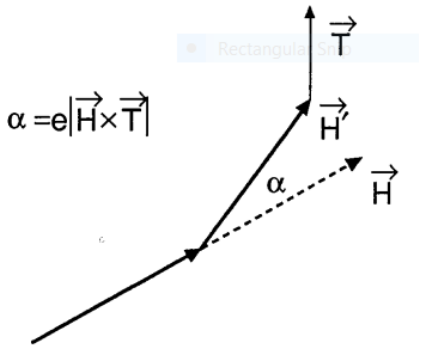
Figure 2

Now with this method, to model and visualize complex branching plants, a more graphical interpretation of the L-system is needed, so we have turtle interpretation. A. R. Smith demonstrated the potential of L-systems for realistic image synthesis using state-of-the-art computer graphics techniques[1, 2]. Prusinkeiwicz incorporated the geometric commands directly within the L-system,

and extended this turtle interpretation to include bracketed L-systems and 3D models[3].

The turtle interpreting a string is represented by a set of attributes that constitutes its state. The major components of the turtle's state are its position, represented by three cartesian coordinates x, y and z, and its orientation in space, represented by three vectors indicating the turtle's heading (H vector), Left (L vector), and Up (U vector). These vectors are unit vectors perpendicular to each other.

$$\left[\begin{array}{ccc}\vec{H}' & \vec{L}' & \vec{U}'\end{array}\right] = \left[\begin{array}{ccc}\vec{H} & \vec{L} & \vec{U}\end{array}\right] \mathbf{R}$$



$$\mathbf{R}_{\mathrm{U}}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_{\mathrm{L}}(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$\mathbf{R}_{\mathrm{H}}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Now to implement these concepts, we will have two classes here Turtle and L-system. Here the Lsystem class will implement a L-system, which will be a set of rules to generate a huge string to draw a tree. And this string will be used to give commands to the turtle. This consists of the axiom, a tree to start with, the number of recursions for generating rules from the axiom, a fixed number of rotations, turns, thickness of branches etc. From a given axiom, this class will generate a string using defined production rules.

The Turtle class will implement a turtle graphics system. We can move it forward, backward, turn, rotate. It can save its states, and return those states. Which will be used to implement trees, where we can turn left and right, draw a branch, etc. In our turtle class we will have methods such as drawTrunk, drawLeaf, rotate, translate, SetThickness, setAngle, etc. We will have current states of the turtle which will be its coordinates, angle of rotation and thickness when drawing the trunk of the tree and it will be following a set of production rules set by L-system.

A leaf can consist of circles and triangles. And we can draw lines on the viewport using starting and ending points for leaves.

3 MILESTONES

S.No.	Milestone	Submission	Member	Status
1	Create a class for generating branches. Set the production rules for this.	Mid evaluation	Kunal Rathod	Done (implemented its functioning in lsys:: visualisepattern)
2	Make a class for the L-system, which will implement a set of rules. And will give a string from an axiom using the set of production rules. Draw pattern from generated L-system string.	Mid evaluation	Kunal Rathod	Done (class lsys)
3	Make shader programs. Need to do shading for leaves and for the branches. It should be according to the shapes of generated leaves which can be made of circles and triangles.	Final evaluation	Kunal Rathod	
4	Create a color class. Make a class for drawing a pixel on the screen from given parameters of coordinates.	Final evaluation	Kunal Rathod	
5	Create basic template code for OpenGL. Create a class for generating leaves.	Mid evaluation	Sarthak Tyagi	Done (main)
6	Make a Turtle class which will work according to the string made by the L-system class. This class will have attributes like states, position, orientation, direction, angle and functions like moving the turtle to a	Mid evaluation	Sarthak Tyagi	Done (class turtle)

	specific position, changing direction, restoring the state of the turtle, etc.			
7	A class to generate the tree using L-system and Turtle graphics. A cylinder function to create a cylinder and return its vertices. Similar function for leaves.	Final evaluation	Sarthak Tyagi	
8	Render shapes like circles/triangles for the leaves and cylinder for the trunk of the trees. Adding texture to the trees and leaves.	Final evaluation	Sarthak Tyagi	

3 RESULTS

We currently have two main working classes: lsys (lsys.h, lsys.cpp) and turtle (turtle.h, turtle.cpp).

Lsys class:

This class is generating a string to feed to turtle using the L-system production rules.

function: `void createSequence(string current, int depth, int current_depth);`
Here the sentence is our axiom, the initial string we upload, and depth is the depth of the tree we want, current_depth is for calculation purposes.
This is a recursive function where according to the sentence given, it uses the production rules defined to make a tree and calls a createSequence() function with the updated string.

Here production rules used are:

```
if(s == 'F')
 return "FF";
if(s == 'X')
 return "F+[-F-XF-X][+FF][--XF[+X]][++F-X]";
```

X is a dummy character used in L-systems and F means moving forward in L-systems. We also have an initial angle provided (25.0) to the L-system which will be used by turtles. These production rules are different for different kinds of trees. We have taken these production rules from this site: click here. To make a tree like below(image is from the same site).

Axiom X; F → FF and X → F+[-F-XF-X][+FF][--XF[+X]][++F-X]

So generating a string will make a whole string for a tree which then we will provide to the turtle.

There is another function in Lsys class which is explained after turtle because it uses an object of turtle class.

Turtle Class:

This is the turtle class implemented using the concept of turtle in L-system based tree generation.

Attributes:

```
   int len;
   State current_state;
   stack<State> state_stk;
```

Main attribute of this turtle class is current_state and a stack of states. State is basically the current x, y and current rotation of the turtle.

State class is implemented like below:

```
class State{
   public:
       int x;
       int y;
       float rotation;
};
```

Functions:

```
void translate(int new_x, int new_y)
void setTheta(float theta)
void RotateTurtle(float angle)
```

```
void save_Pos()
void retrieve_Pos()
void draw()
```

These functions basically update the current state of the turtle; translate and update the current_state by new coordinates, setTheta changes angle of current_state, RotateTurtle rotates the current_state.rotation.

We have a stack of states, which saves and restores the states of the turtle, these are used when we draw the patterns using this turtle class to make branches.

We have a draw function in the turtle which draws a line between two points using glf functions.

```
draw()
   {
       int a = current_state.x + (int)(len * cos(current_state.rotation));
       int b = current_state.y + (int)(len * sin(current_state.rotation));

       glBegin(GL_LINES);
           glVertex3f(current_state.x, current_state.y, 0.0f);
           glVertex3f(a, b, 0.0f);
       glEnd();

       current_state.x = a;
       current_state.y = b;
   }
```

Function: `void lsys::visualise_sequence(string food)`

In Lsys class we have an object of turtle class. This visualise_sequence uses the string generated by create_sequence function and feeds it to the turtle (call the functions of turtle).

So our new generated string will look like something like this:
XF[+X]][++F-X]XFFF[+F+[-F-XF-X][+FF][--XF[+X]][++F-X]X]][++FFF-F+[-F-XF-X][+FF][--XF[+X]][++F-X]X]F+[-F-XF-X][+FF][--XF[+X]][++F-X]XFFFFFFFFF[+FFF+[-FFF-F+[-F-XF-X][+FF][--XF[+X]][++F-X]XFFF-F+[-F-XF-X][+FF][--XF[+X]][++F-X]X][+FFFFFF][--F+[-F-XF-X][+FF]

So according to L-system rules, we move the turtle according to cases below:

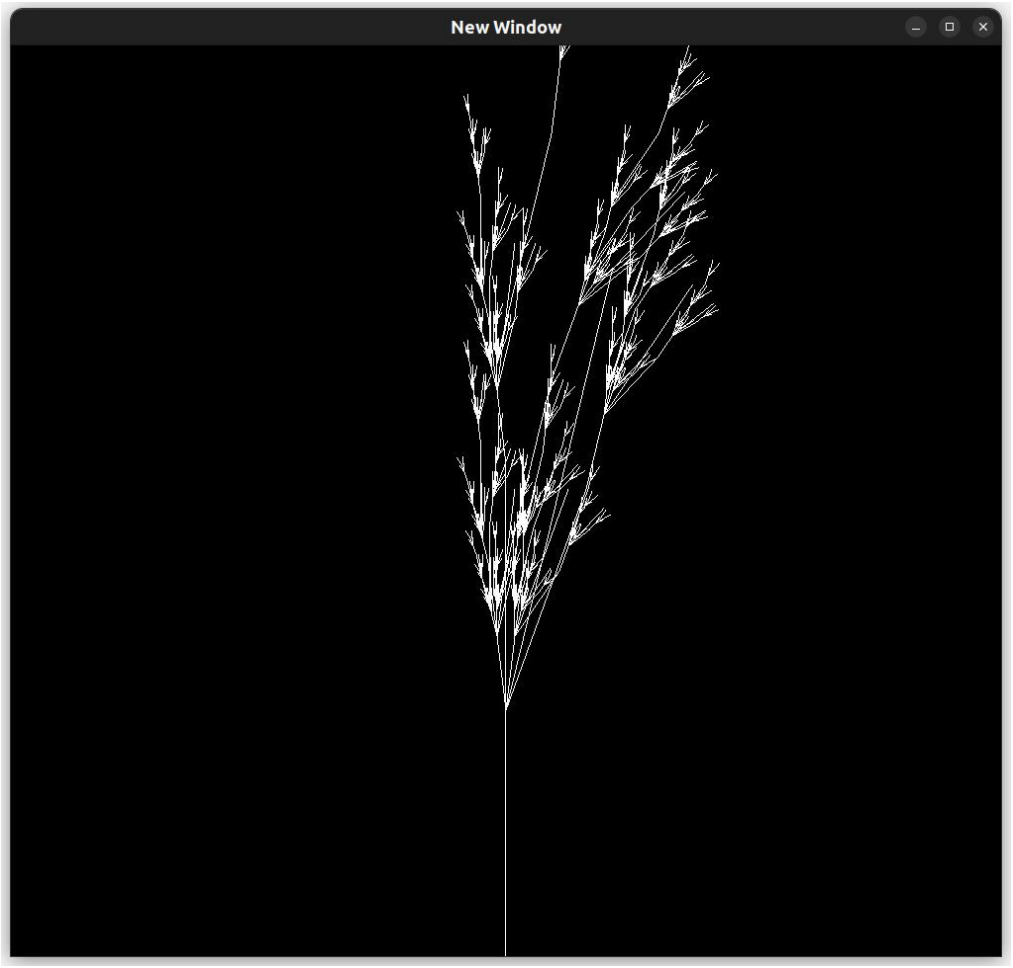
```
            case 'F':
            turt.draw();

            case '+':
            turt.RotateTurtle(angle);
```

```
            case '-':
            turt.RotateTurtle(-angle);

            case '[':
            turt.save_Pos();

            case ']':
            turt.retrieve_Pos();
```

This is the standard L-system rules that we have to move forward when we have ‘F’, or to save and restore states when we have brackets like ‘[’ and ‘]’.
And visualise_sequence iterated to the whole generated string and for each character in the string it moved the turtle according to above rules. And the draw function draws the lines on the screen.

The logic behind using save and restore state:
Here when we have ‘[’ this bracket (let’s say at point A), in L-system it means that there will be a branch starting. So now we will save the current state of the turtle in the stack(state of the point A) and make that branch using the turtle. At the end of the bracket ‘]’, the turtle will be at the end of the branch. Now since it was a branch there will be multiple branches coming from that point A, so after drawing the point we need to move the turtle to point A so we will restore it when we have ‘]’ this bracket (when the branch is over). So after restoring to state A, the turtle will be again at point A, And we can draw another branch or move forward with our tree. (Please note that this logic of saving and restoring state is also a part of L-system rules).

Output:



This tree is similar to the one in the site from where we took the production rules. By just changing the productionrules function in lsys we can make all the different types of tree or we can even make our own rule to make our own trees.

Currently this is only the trunk of our tree, we are making it using the turtle.

Link to the demo video is on the github
Link to the github repo:
https://github.com/CSE-333-Computer-Graphics-2022/L-System-tree-generation/

REFERENCES

[1] Smith. Plants, fractals, and formal languages. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 22-27, 1984) in Computer Graphics, 18, 3 (July 1984), pages 1-10, ACM SIGGRAPH, New York, 1984.

[2] A. R. Smith. About the cover: Reconfigurable machines. Computer, 11(7):3-4, 1978.
[3] P. Prusinkiewicz. Applications of L-systems to computer imagery. In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, editors, Graph grammars and their application to computer science; Third International Workshop, pages 534-548. Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science 291.
[4] Intelligent Tree Modeling Based on L-system
[5] PARAMETRIC L-SYSTEMS AND THEIR APPLICATION TO THE MODELLING AND VISUALIZATION OF PLANTS, BY James Scott Hanan Regina, Saskatchewan June, 1992
[6] https://iopscience.iop.org/article/10.1088/1757-899X/322/6/062005/pdf
[7] L-system