CS148 Final Project Report: L-system Plant Visualizer with Wind Animation

Shalom Rottman-Yang SUNet ID: jaronry

August 13, 2016

1 Problem Statement

For my final project, I wanted to visualize plants created parametrically with L-systems in real time, and have them be animated as if in wind. I wanted to allow user control over the parameters used to create the plants. I was inspired by the videos *Real-Time Rendering and Animation of Trees* by Ralf Habel and *L-System Visualiser with C++ and OpenGL* by George Rassovsky.

2 Division of Labor

As this was an individual project, I did all the work.

3 Sources

I used the following papers as sources to learn about L-systems and wind animation for trees:

- Real-time 3D Plant Structure Modeling by L-System with Actual Measurement Parameters by Rawin Viruchpintu and Noppadon Khiripet
- L-System Plant Geometry Generator by Hung-Wen Chen
- L-systems: from the Theory to Visual Models of Plants by Przemyslaw Prusinkiewicz, Jim Hanan, Mark Hammel and Radomir Mech
- Chapter 1: Graphical modeling using L-systems from *The Algorithmic Beauty of Plants* by Przemyslaw Prusinkiewicz and Aristid Lindenmayer
- Chapter 6: GPU-Generated Procedural Wind Animations for Trees from *GPU Gems 3* by Renaldas Zioma

4 Background

4.1 L-systems

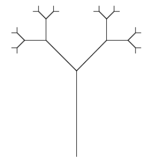
L-systems, or Lindenmayer systems, are formal grammars for generating geometric structures. They are composed of an alphabet of symbols and constants, an axiom string to begin generation, production rules that map each symbol to a string of symbols, and constants which map to specific actions for geometric generation. Constants may include, for example, rotation about a specific axis, saving and reverting state, or change in generation parameters. Oftentimes, the specific angle (or set of angles) to be used in generation are also included in the L-system. One or more of the symbols may also map to an action in geometric generation. To generate structures, the production rules are used to replace symbols per the rules for a given number of iterations. At each iteration, every symbol that has a rule is replaced.
For example, the L-system defined as follows:

- Symbols: F
- Constants: none
- Production rules: F → FF
- Action: F → Draw line segment

would simply draw a straight line. The L-system defined as follows:

- Symbols: F
- Constants: [,], +, -, !
- Production rules: F → F[-!F][+!F]
- Action: F → Draw line segment
- Action: [→ Push current state to stack
- Action:] → Set current state to state popped off stack
- Action: + → Rotate left by angle
- Action: - → Rotate right by angle
- Action: ! → Set line length to half line length
- Angle: 45 deg
- Line length: 1

would draw a simple 2D tree that looks like this:

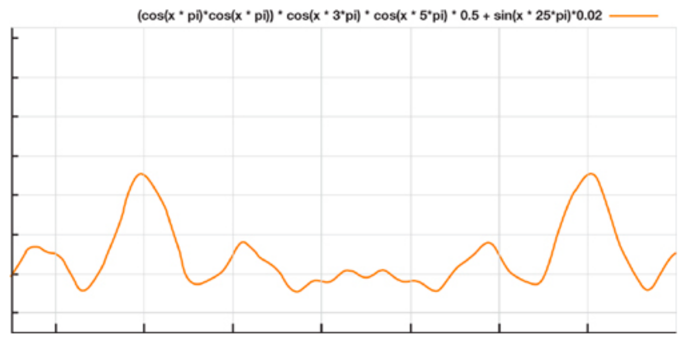


They can be generalized to 3D. L-systems were originally used to describe algae growth, and have since been used extensively to describe the growth of plants and fractals.

4.2 Wind Animation for Trees

The principles of wind animation I used for my project came from the chapter of *GPU Gems 3* that I consulted. There were a few key principles that I used to guide the physical model that I constructed:

- Given the complexity of forces acting on trees in wind, we can model tree movements as essentially random.
- Therefore, branch movement can be modeled as a sufficiently noisy function, e.g. the following function taken from the chapter:



- Wind can be modeled as a vector in 3D space representing the direction of a 2D force field.
- Branches can be treated as rigid segments connected by ball joints.
- Branches will move in the direction of the wind. They will also move in the direction perpendicular to the wind due to the uneven distribution of branches, and will rotate around their own axis due to lift.
- Turbulence and the effect of other branches can be modeled as higher frequency noise on top of the existing function.
- The amplitude of the noise function modeling the movement of a branch can be adjusted based on branch characteristics such as width, length, and elasticity.
- Branches can be on the side of the tree facing the wind or facing away from the wind. Branches on the side of the tree facing into the wind will have reduced amplitude, and vice versa.

5 Implementation

L-systems are inherently hierarchical; the current state is influenced by previous steps. Trees are inherently (somewhat) cylindrical; each branch can be simply modeled as a cylinder of diminishing radius. For these reasons, and because OpenGL 1.1 makes it easy to hierarchically draw primitives such as cylinders, I used OpenGL 1.1 to code my project. I used Homework 2 as the code foundation for my project, since it sets up movement and lighting. Animation could be easily accomplished with `glutIdleFunc()`.

5.1 L-system Visualization

For the L-system visualizer, I allowed 7 production rules: F, L, H, G, X, Y, and Z. F draws a branch, and L draws a leaf. Branches are cylinders, with a parameter for the widening or narrowing of a branch along its length. The ends of branch cylinders are capped with domes, to look a bit more natural. As the local coordinate system follows the drawing matrix, I was able to simplify the drawing of a branch to be along the local z-axis, to help make calculations for wind animation easier. Drawing is done recursively, taking advantage of the current drawing matrix to store state. However, branch radius and length are not stored in the matrix. A true L-system would nevertheless store these in the state. I could accomplish this using a stack parallel to the matrix; however, for performance purposes, I did not want to do this, so I added constants to undo changes to radius and length.

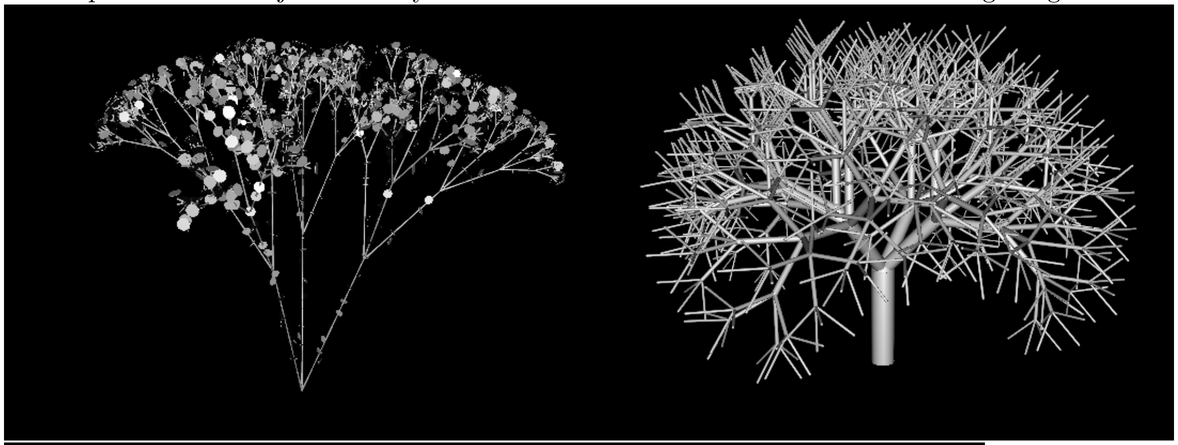
My visualizer allows flexibility in the production rules and axiom, but the constants are set in stone, and only F can specify drawing a branch (and L a leaf). Only one rotation angle is allowed. The constants are as follows:

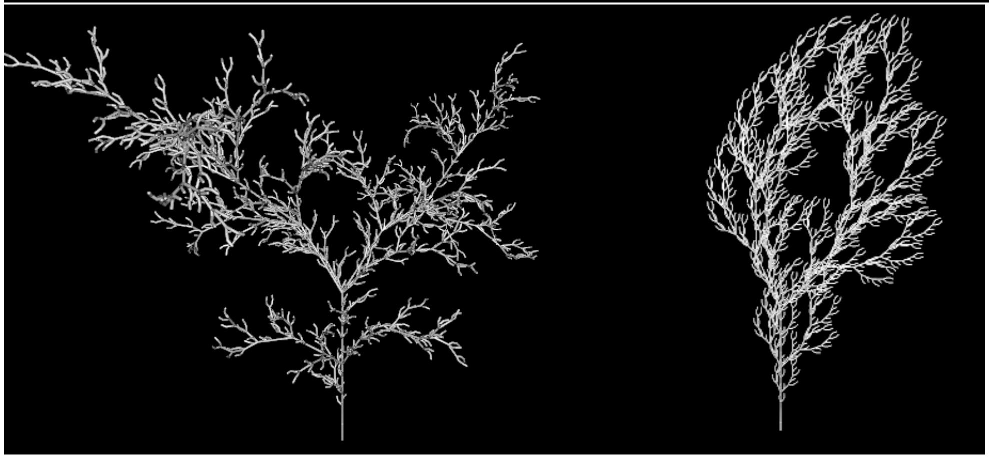
- ! multiplies current radius by the first diameter ratio
- ? divides current radius by the first diameter ratio
- : multiplies current radius by the second diameter ratio
- ; divides current radius by the second diameter ratio
- $ multiplies current length by the first length ratio
- * divides current length by the first length ratio
- # multiplies current length by the second length ratio
- @ divides current length by the second length ratio
- + rotates by the rotation angle around the x-axis
- - rotates by the negative rotation angle around the x-axis

- [pushes state
-] pops state
- & rotates by the rotation angle around the y-axis
- ˆrotates by the negative rotation angle around the y-axis
- / rotates by the rotation angle around the z-axis
- \rotates by the negative rotation angle around the z-axis
- — reverses direction (rotates by 180 around x-axis)

I included two types of leaves. The first type represents leaves as spheres, which gives trees a simpler, cartoon-y look. I wanted to include leaves that could move with wind as well, so the second type represents leaves as flat circles at the end of thin stems. This type does not look realistic, but does show leaf movement in wind. I included an option to draw a leaf when the end of recursion is reached. This can be used to add leaves to a plant that does not include leaves in the L-system definition.

The implementation of just the L-system visualizer allowed me to make the following images:





5.2 Wind Animation

For the wind animation, I moved each branch in the direction of the wind by a slight amount based on a noise function, as well as rotating it about its own axis. For the noise function in the direction of wind, I used

$$0.5\cos(\pi x)\cos(\pi x)\cos(3\pi x)\cos(5\pi x) + 0.02\sin(25\pi x) + 0.1$$

For the noise function perpendicular to the direction of wind, I used

$$0.6(0.5\cos(\pi x)\cos(\pi x)\cos(3\pi x)\cos(5\pi x) + 0.02\sin(25\pi x))$$

For the noise function for rotation, I used

$$0.1\cos(\pi x)9\sin(3\pi x + 2)$$

I arrived at these functions by using the one provided by *GPU Gems 3* as a basis, and tuning them until they looked best. I included a couple other functions as comments in the source code in `windFunc()` and `perpFunc()`; the above functions look best, I think.

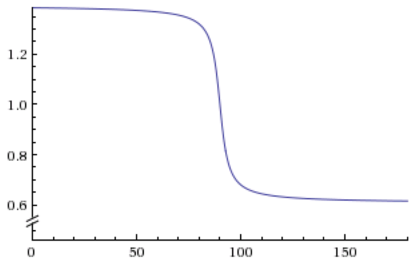
In order to move in the direction of the wind, the wind vector needed to be rotated in the opposite direction of any rotation done while drawing the L-system. I kept track of the current wind vector in a stack, to accompany the hierarchical drawing. Rotating around the axes was simple; however, during my algorithm for individual branch movement with the wind, I needed to rotate around an arbitrary axis. To do so, I used code by Bibek Subedi from programming-techniques.com, which is also cited in my source code.

The noise functions given above (functions of time) were used to get the amount of branch movement in the appropriate directions. I used a phase shift that was changed by a different amount for each type of step taken in the L-system drawing in order to get a different values for each branch, so that movement does not look synchronous.
I modified the values returned by the noise functions (effectively changing the amplitudes of the noise functions) in the following ways:

- I multiplied the movement values in the wind and perpendicular wind directions by $\frac{length}{E*width}$, where E is an elasticity value that can be controlled by the user, to simulate longer branches moving more and wider branches moving less. I multiplied the movement value in the rotational direction by half that amount, since wood is harder to rotate than bend.
- I obtained the angle between the wind vector and the branch vector. I multiplied the movement values in the wind and perpendicular wind directions by

  $$1 - \frac{\arctan((\theta - 90)/3)}{4}$$

  This function looks like this:

  In this way I approximated the principle that branches on the side of the tree facing into the wind will move less than branches on the other side. This method of implementation fails when a branch on the side of the tree facing into the wind has an acute angle with the direction of the wind due to tree movement, but this is uncommon enough (and only happens to branches near the sides of that tree face) that it does not significantly affect the visuals.

I applied the algorithm for branch movement to each branch individually. My algorithm for branch movement is as follows:

1 Add the wind vector times its movement scalar and the perpendicular wind vector times its movement scalar to get the direction of movement.

2 Find the vector that results from adding the direction of movement to the vector in the direction of the branch (which is (0, 0, 1)).

3 Find the angle of rotation between the vector computed in part 2 and the branch, then use the cross product to find the axis of rotation.

4 Rotate around the axis of rotation by the angle of rotation.

5 Rotate around the z-axis by the rotation amplitude scalar times a specified branch rotation angle.

Animating the leaves was done differently to speed up computation. Spherical type leaves did not need to be animated. For the stem-and-flat-circle leaves, I rotated the leaf around each of the three axes using a function of the wind vector, the time, the phase shift, and `rotFunc()`.

To see the wind animation in action, simply make and run the source code.

5.3 User Controls

User control was done through the keyboard. The following are the keys and their actions:

- `>` Increases the depth of the L-system
- `<` Decreases the depth of the L-system
- `b` Increases the initial radius
- `s` Decreases the initial radius
- `l` Increases the initial length
- `m` Decreases the initial length
- `r` Increases the rotation angle of the L-system
- `e` Decreases the rotation angle of the L-system
- `p` Increases the leaf radius
- `o` Decreases the leaf radius
- `k` Increases the leaf thickness (for flat leaves)
- `j` Decreases the leaf thickness
- `t` Toggles whether the L-system places leaves at the end of a recursive draw
- `a` Toggles whether the leaves are spheres or stem-and-flat-circle
- `w` Toggles whether the wind field is drawn
- `1` Increases the wind vector's x value
- `2` Decreases the wind vector's x value
- `3` Increases the wind vector's y value
- `4` Decreases the wind vector's y value
- `!` Increases the wind vector's z value

- `@` Decreases the wind vector's z value
- `5` Increases the speed of animation
- `6` Decreases the speed of animation
- `7` Increases the narrowing ratio (the amount a branch narrows along its length)
- `8` Decreases the narrowing ratio
- `9` Increases the amount branches rotate along their own axis in the wind
- `0` Decreases the amount branches rotate along their own axis in the wind
- `c` Moves the camera up
- `v` Moves the camera down
- `;` Decreases the elasticity of the plant
- `:` Increases the elasticity of the plant
- `y` Shows a geometric pattern to demonstrate an L-systems' ability to draw fractals
- `u` Shows a standard tree
- `i` Shows a standard bush-like plant
- `h` Shows another plant

Camera movement is available through the mouse. To write an L-system of their own, the user must edit the source code.

6 Presentation

For a better presentation in the visualizer, I went for a cartoon-y style. I made the background sky blue, and added a grass-colored sphere on top of which the plant sits. I colored the branches brown and the leaves leaf green. I would have liked to add a toon shader, but could not do so in OpenGL 1.1. The final visualizer produces images like so (with animation):