Generating Trees From L-Systems

CS175: Computer Graphics
Serena Booth

December 16, 2015

1 Intro

I created a tree-generating system. This system uses Lindenmayer, or L-systems, in which a formal grammar describes a tree. The grammar consists of an axiom and a set of rules; the axiom describes the smallest possible tree for a given grammar, and the rules are then applied in order to substitute characters from the axiom to generate a string. My grammars are interpreted stochastically. I parse such a generated string into a scenegraph SgRbtNode. When the tree branches, I add a transform node, and when the tree continues along a branch, I add a shape node representing the branch.

2 Hot Keys

On running my system – a make file is provided – a tree is generated. In order to better interact with the system, I provide hot keys. These processes are detailed here.

- **0: animate the current tree growing!**

  How it works: I modified the Drawer class to create a TreeDrawer. This TreeDrawer iteratively draws the tree to a larger depth, starting from a depth of 2 and ending at the depth of the tree as represented in the scenegraph.

- **1: increase the recursion of the L-System.**

  How it works: I wrote an L-System parser which takes an axiom and a set of rules and iteratively applies the rules to the axiom or the generated string. On startup, my system applies the rule set to the axiom 6 times. I keep track of a global variable indicating the depth to which I wish to generate. On pressing '1', this variable is incremented, and the tree is removed from the scenegraph, regenerated, and re-added to the scenegraph.

- **2: decrease the recursion of the L-System.**

  How it works: see '1'. In this instance, the global variable is decremented, but the approach is the same.

- **3: grow a different L-System.**

  How it works: I provide 6 L-System grammars to demonstrate the trees these systems can produce. On pressing '3', I read the next L-System, apply its rules to its axiom some number of times, remove the existing tree from the scenegraph, grow the new tree, and re-add this new tree to the scenegraph.

- **4: increase tree branch thickness.**

  How it works: tree branches are represented by cylinder geometry. I initialize such a cylinder to have a height of 1 unit and a radius of 1 unit. When I add this cylinder to the tree, I stretch it in x, y, and z dimensions. Thus I am able to track an overall branch thickness global variable, and increase this variable in order to increase branch thickness. This thickness variable is multiplied in to the stretch factor. As above, I then remove the existing tree from the scenegraph, generate a new tree, and add this new tree into the scenegraph.

- **5: decrease tree branch thickness.**

  How it works: see '4'. In this instance, the global variable is decremented.

- **6: increase branch length.**

  How it works: see '4'. In this instance, I stretch the cylinder's height when before I was stretching its radius.

- **7: decrease branch length.**

  How it works: see '4'. In this instance, I shrink the cylinder's height.

- **8: turn off leaves.**

  How it works: a global variable toggles the presence of leaves on the tree on and off. On pressing this hot key, the existing tree is removed from the scenegraph, a new tree without (or with, respectively) leaves is generated, and this resultant tree is re-added to the scenegraph.

- **9: simulate leaves moving.**

  How it works: in order to achieve a partial wind effect, I allow the user to press the hot key '9'. This removes the existing tree from the scenegraph, generates a new tree, and adds this tree to the scenegraph. This new tree differs in one major way: it includes transform nodes between all the branches and all the leaves. From this point, I call the function leafAnimateTimerCallback, which adjusts the rotation of

the transform nodes. All of the leaves move in parallel, and the effect is designed to be reminiscent of leaves blowing in the wind. This simulation ends when the user presses '9' again.

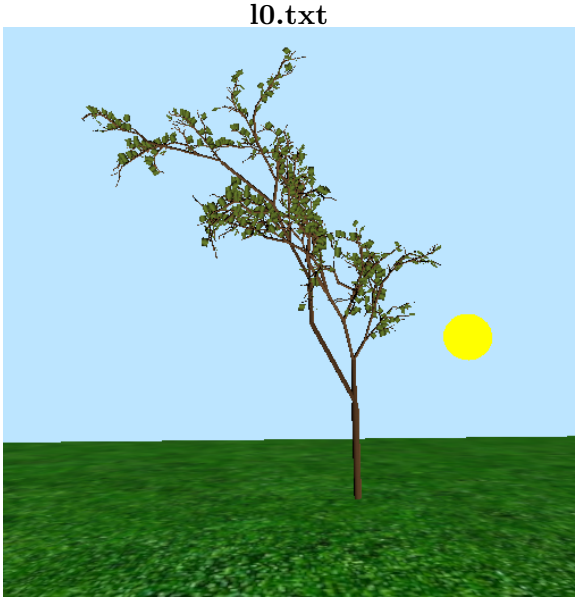
3 L-Systems, Interpretation, and Results

3.1 Interpretation

I use a uniform set of variables in my L-Systems. The character 'X' is used as a placeholder for generating future iterations. The character 'F' means move forward in this direction - e.g. place a cylinder at this location. The character '-' means move -30°, plus or minus 0 to 5° randomly in one of the X, Y, or Z dimensions as viewed through the cylinder's frame of reference; the character '+' means the same, but 30°, plus or minus 0 to 5°, instead. The character '[' indicates the start of a branch; the character ']' indicates the end of a branch. On seeing one of these start branching characters, I push the tree's current state onto a stack, and return to that state on seeing the branch's end.

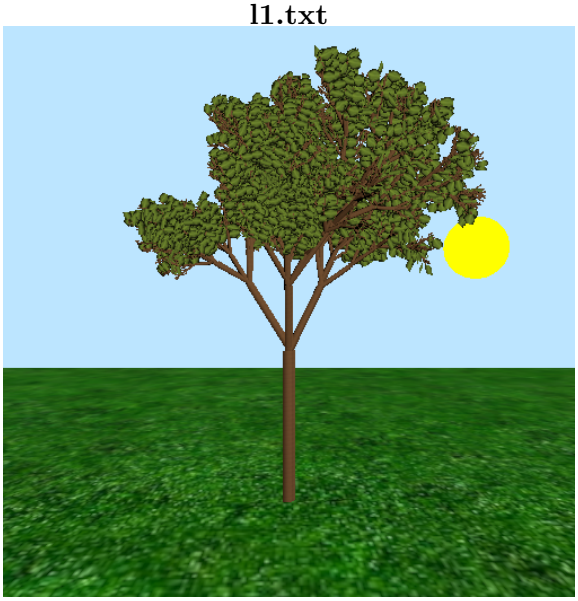
3.2 Trees generated from L-Systems

l0.txt



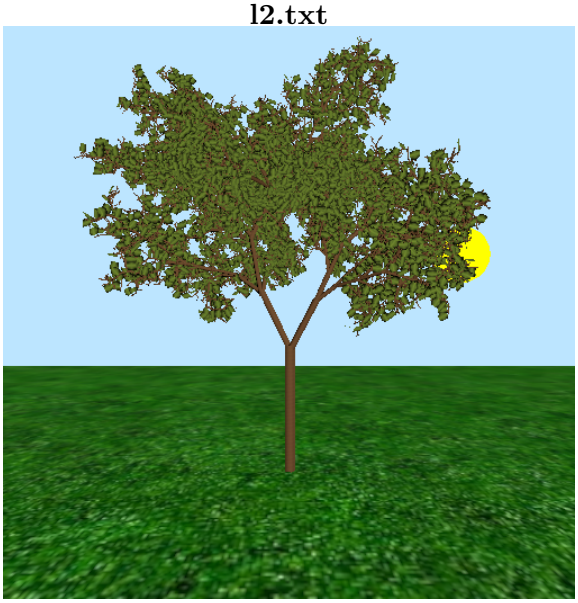
Axiom: X
X → -F[-X+X]+F[-X+X]
F → FF

l1.txt



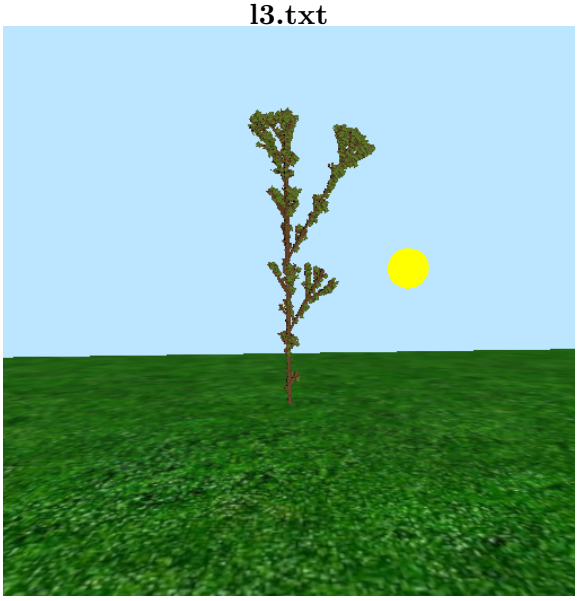
Axiom: X
X → F[-X+X][+X-X][-X+X][+X-X]
F → FF

l2.txt



Axiom: X
X → F-[-X-X+X+X]+[+X+X-X-X]
F → FF

l3.txt

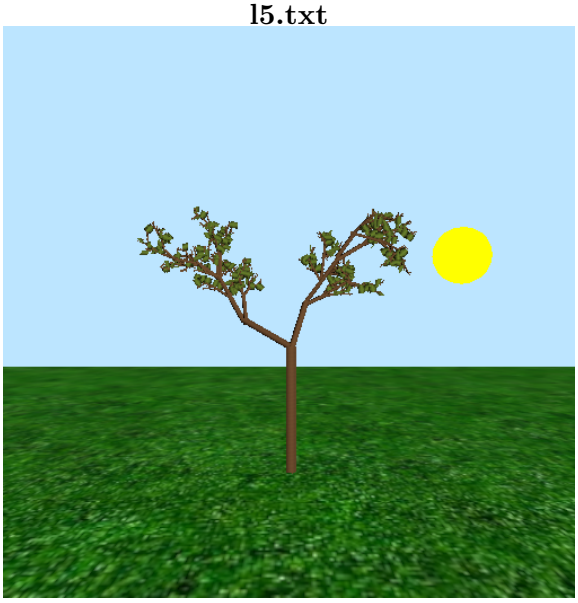


Axiom: F
F → F[+F][-F]F

l4.txt

Axiom: X
X → F-[-X+X]+[+X-X]
F → FF

l5.txt



Axiom: F
F → FF[+F][-F][F]

4 Difficulties & Hacks

My main difficulties in this project have been issues of complexity. A tree generated by one of these grammars (e.g. L2.txt), for 6 iterations will generate a string of over 800,000 characters. Interpreting this string gives a tree with over 400,000 transform nodes, and more than 100,000 leaves. Because of this complexity and the depth of the generated scenegraph, rendering such a tree takes a significant amount of time on my machine. In response to this complexity I have considered simple optimizations. For example, the tree is represented by cylinders, each of which contain 12 vertices. I could simplify this tree somewhat by substituting simpler geometry, such as a rectangle, when the branch width is sufficiently small. As is perhaps expected, this optimization did not dramatically increase runtime, so I disabled it in favor of a more uniform tree appearance.

I likewise had a difficulty in rendering the leaves on the trees. The leaves consist of very flat rectangles with a textured geometry on top. The texture is represented by a .ppm file. This file contains a leaf against a black background, so in the fshader, I check the color of the fragment under consideration. If the texture at this location is black, I want this fragment to have an alpha value of 0. If the texture at this location is part of the leaf, I want to render that leaf. However, I do so in no particular order, and am not reasonably able to order the leaves from front-to-back by distance due to the scale of the operation.

Without this, however, the leaves look broken when rendered. I thus opt to disable the GL_DEPTH_TEST when rendering the leaves.

5 Personal Take Aways

I enjoyed learning about how formal grammars can represent natural phenomenon, and found the results extremely compelling, even from a simple system.

I found the representation of the tree – both in using the scenegraph for this data structure and in creating cylinders, leaves, textures, and relationships between those components – challenging. I spent a while considering different cylinder representations – from a mesh, from geometry, using GLUT's inbuilt cylinder creation tool – and found that reflecting on this gave me more perspective on graphics geometry. I ultimately chose to create the cylinder directly from geometry, with the assistance of a resource credited below.

Unexpectedly, I also found this project taught me a great deal about object oriented programming and design. I haven't done a lot of OOP coding at Harvard, and interacting with this system, and its many moving parts – from the Drawer class to my own LSystem class – was enlightening.

Inspired by this project, I made a trek out to the Museum of Science Pixar exhibit.

6 Resources

- Prusinkiwicz, P.; Lindenmayer, A. *The Algorithmic Beauty of Plants.* 2004.

  This book provided inspiration, an understanding of L-systems, and some simple L-system grammars which I used and modified in my implementation of tree growing.

- `https://en.wikipedia.org/wiki/L-system`

  A good overview of L-systems. L0.txt is a modified version of Example 7 from this resource.

- Cylinders, Spheres, and Boxes with Direct3D11 and SlimDX. `http://richardssoftware.net/Home/Post/7`

  This website provided code for drawing a cylinder into a vertex and an index buffer.