WPI Worcester Polytechnic Institute

Computer Science

CS 543, Fall 2013 Homework 3: Hierarchical 3D Modeling of L-System Trees

Due: Tue Oct 29, submitted by class time

Objective: In this assignment, you will learn how to generate a forest of trees using an iterated function system (IFS) called Lindenmayer Systems (a.k.a. L-Systems), and place those trees on a "ground plane." You shall also insert PLY models of a car and other PLY objects and place it on the ground plane to model a park. This assignment consists of two parts: a "Preparation" part and a "Submitted Stuff" part.

NOTE: Please submit ONLY the work described below in the section "submitted stuff" by the due date (Oct 29, 2013). Do NOT submit the work you do in the "prep coding" section

Preparation: The aim of this preparation part is for you to create the IFS for generating the strings that will define each tree for the forest. In addition, you will create a PolyCylinder routine to better understand transformations (*e.g.*, translations, rotations) in OpenGL.

A drawing pattern is defined by a *turtle string* made up of command characters that control how the turtle moves, as well as its state. The commands include:

Character	Meaning
F	Move forward a step of length *len*, drawing a line (or cylinder) to the new point.
f	Move forward a step of length *len* without drawing.
+	Apply a positive rotatation about the X-axis of *xrot* degrees.
-	Apply a negative rotatation about the X-axis of *xrot* degrees.
&	Apply a positive rotatation about the Y-axis of *yrot* degrees.
^	Apply a negative rotatation about the Y-axis of *yrot* degrees.
\	Apply a positive rotatation about the Z-axis of *zrot* degrees.
/	Apply a negative rotatation about the Z-axis of *zrot* degrees.
|	Turn around 180 degrees.
[	Push the current state of the turtle onto a pushdown stack.
]	Pop the state from the top of the turtle stack, and make it the current turtle stack.

L-Systems are used to generate a turtle string by iteratively expanding a *start token* by applying production (or re-writing) rules. Each L-System consists of a grammar that defines re-writing rules. Each rule in the grammar consists of a left-hand side (LHS) and a right-hand side (RHS), separated by a colon.

A sample grammar looks like this:

```
start: F-F-F-F
F: F-F+F+FF-F-F+F
```

In addition to specifying a grammar, we also need to specify the values for *len*, *xrot*, *yrot*, and *zrot*, as well as a value denoting how many times we want to iterate (*i.e.*, apply the production rule(s)).

Similar to the way the Koch curve is created by replacing each segment with a predefined pattern, the turtle string is rewritten by substituting every instance of a LHS by its corresponding RHS. For any token in the string for which there is no matching LHS, the token is simply copied into the new string. For example,

given the grammar:

```
start: F+F
F: F-F+F-F
```

after one iteration, the resulting turtle string would be:

```
F-F+F-F+F-F+F-F
```

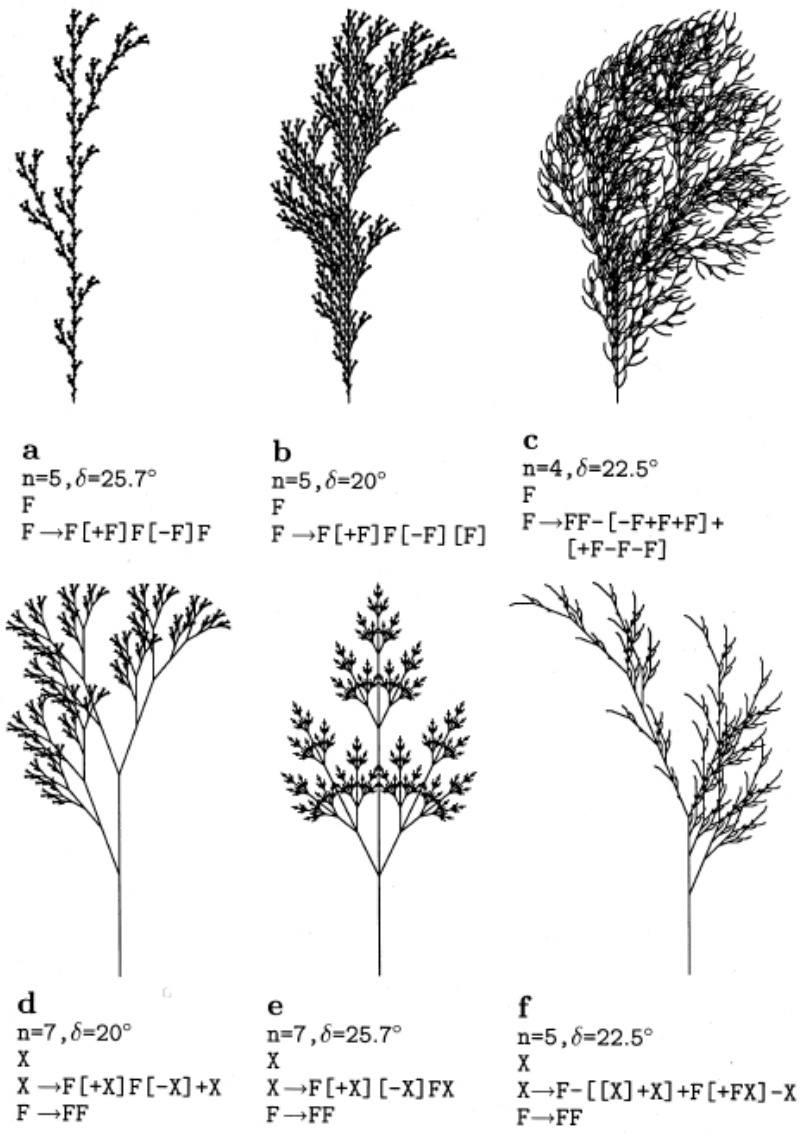
and after two iterations, the resulting turtle string would be:

```
F-F+F-F-F-F+F-F+F-F+F-F-F-F+F-F+F-F+F-F-F-F+F-F+F-F+F-F-F-F+F-F
```

Your system should be able to handle multiple production rules, each having a unique LHS, for example:

```
start: X
X: F-[[X]+X]+F[+FX]-X
F: FF
```

Here is an example (in 2D) of some grammars and their corresponding trees. Can you extend these to 3D for use in this project? (The value of *n* is the number of iterations, and the angle specified is the rotation for the "+" and "-" characters.)



Setup: Using the starter code (same one as you got for homework 2) [here] or your project 2 code as a starting point, add in the other modules you create in this project to render L-systems (*e.g.*, grammar.h grammar.cpp, *etc.*). Be sure to use the right type of projection matrix and setup parameters to show proper perspective (foreshortening).

Prep Coding:

1. **L-Systems:** Write a program that implements the L-System re-writing scheme described above. Your program should be able to read several grammars from grammar files, store them in appropriate data structures, and then generate the correct turtle strings by applying the rewriting rules as specified. Here is a sample L-System file. A description of the file format is contained in the file itself, and you may assume that all files have the same format.

   Compile and run your program!

2. **The PolyCylinder:** We define a *PolyCylinder* as a polyline in three dimensions. Given a sequence of points in 3-space { (x1, y1, z1), (x2, y2, z2), (x3, y3, z3) }, a polycylinder draws a cylinder from (x1, y1, z1) to (x2, y2, z2), and another one from (x2, y2, z2) to (x3, y3, z3). To avoid "gaps" in the joints of the polycylinder, a sphere of the same diameter as the cylinders is drawn at each point. You can use the following cylinder and sphere ply files ([Here]) for drawing your PolyCylinder.

   You need to write some code that:

   1. Moves (translates) to the first point,
   2. Draws a sphere,
   3. Points (rotates) towards the next point,
   4. Draws a cylinder to the next point,
   5. Moves (translates) to the next point,
   6. Draws a sphere,
   7. Points (rotates) towards the next point,
   8. Draws a cylinder to the next point,
   9. Moves (translates) to the next point,
   10. And so on, for all the points in the list.

   HINT 1: This part is an exercise in translating and rotating your coordinate system before drawing your geometry. After you accomplish this, you will understand how transformations are accumulated in OpenGL.

   HINT 2: You can simplify your life by always drawing your cylinder along the Z axis. To do this, apply your rotations such that you are always "moving" along the Z axis by first rotating yourself to point the current Z direction towards the next point.

   Set up a scene that you will use to draw a park with a forest of trees. Set up a view of your scene, draw a ground plane, and draw some PolyCylinders in the scene. You can just come up with some (x, y, z) points on your own for this. Be creative! Write your name, or something.

   Compile and run your program!

Submitted Now we need to put it all together.

Stuff:

1. Using the code you wrote in the preparation section, your program should read in five L-System files, "lsys1.txt," "lsys2.txt," "lsys3.txt," "lsys4.txt," and ONE OF YOUR OWN, and store them in instances of your *grammar* class. You should then apply the re-writing rules for each grammar according to the values specified for this in each grammar file. Your five L-System files should be drawn on the (a,b,c,d,e) keystrokes respectively. For instance, hitting keystroke a draws lsys1.txt, etc.
2. Choose a random location on the ground plane to start drawing one of the randomly selected trees, using a random color, and draw it.
   HINT: You should apply a translation and a rotation to move to the correct start location.
3. Repeat Step 2 (at least) 5 times in order to draw your forest. For inspiration, you can look at the gallery of forests created by students in a previous CS 4731 class HERE Some of the students in that class used a checkerboard floor. You can just make a plain floor.
4. Make your park look more realistic by adding in more PLY mesh models such as a car and any other objects you think would make your park look realistic. Use some of the PLY files from homework 2. You can get the PLY files [here] Be creative. You don't have to add 3D effects like texturing yet since we haven't covered them in class, although people who decide to add any effects won't have points taken off.
5. Keystroke f draws park including your forest of trees, ground plane and car.

(If you **REALLY** want to test your program, try this input file. This is from p. 20 of the reference book listed at the bottom of this page and is PURELY OPTIONAL!)

Attacking the Problem:

For the L-System part, start out by creating several classes that will help you manage the different things you will need to keep straight. For example, you might want to have a *turtle* class consisting of a *position*, *orientation*, *length* when drawing, and a *string* representing the turtle string.

You might want to have a *rule* class that has strings for the *lhs* and *rhs*.

A *grammar* class would consist of a list of *rules*, along with a method (*addRule*) to add a new rule to the grammar. The main method for the grammar class might be something like a *rewrite* method that takes in a *turtle* and a number of *iterations*, and returns a new string after applying the rules to the turtle string for the desired number of iterations.

The functionality for implementing '[' and ']' should be achieved by implementing a matrix stack and **PushMatrix** and **PopMatrix** functions. This will save and return you to the proper state.

Another thing that will help you greatly is to use the Standard Template Library (STL) that is available with C++. There are a number of classes that you will find very useful, such as "string," "hash_map," "multimap," and/or "list." If you have never used these before, this is a good opportunity for you to "get out of your comfort zone," as these tools will serve you well in most future endeavors, both in this class and well beyond.

Some good links to C++ help include the SGI site and also CPP Reference.

Submitting your Work:

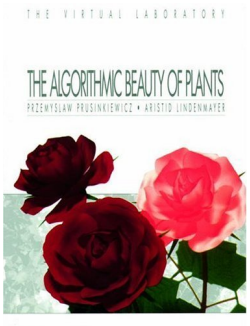
Make sure to double-check that everything works before submitting. Submit all your executable and source files. Put all your work files (Visual Studio solution, OpenGL program, shaders, executable and input files into a folder and zip it. Essentially, after your project is complete, just zip the project directory created by Visual Studio. Submit your project using web-based turnin.

Create documentation for your program and submit it along with the project inside the zip file. Your documentation can be either a pure ASCII text or Microsoft Word file. The documentation does not have to be long. Briefly describe the structure of your program, what each file turned in contains. Explain briefly what each module does and tie in your filenames. Most importantly, give clear instructions on how to compile and run your program. **MAKE SURE IT RUNS** before submission. Name your zip file according to the convention *FirstName_lastName_hw3.zip*

Academic Honesty:

Remember the policy on Academic Honesty: You may discuss the assignment with others, but you are to do your own work. The official WPI statement for Academic Honesty can be accessed HERE.

References:



Most of the motivation for my interest in this work comes from the book "The Algorithmic Beauty of Plants," by Prezmyslaw Prusinkiewicz and Arstid Lindenmayer, Springer Verlag, New York, 1990, ISBN 0-387-97297-8, ISBN 3-540-97297-8.

The whole book has been put online by the author. In addition, there are links to other interesting works on this page.

You can also find it on Amazon.

mailto:emmanuel@cs.wpi.edu