Randall Smith (agrippa@cs.utexas.edu) is in charge of this project.

Introduction

In this assignment, you will design and implement an L-system that generates a tree.

1 L-system

For a definition and description of L-systems, refer to the class notes from September 22nd.

Your primary task in this project is to design and implement your own L-system grammar that generates a realistic-looking tree. The exact form this can take is wide open; the type of tree you generate and how you do it is up to you. The minimum requirements are as follows:

- Your tree must be in full 3D. The user should be able to view all sides of it, and it should still look like a tree.
- Your tree must be generated using a grammar of your own design. In a separate text file, document this grammar and describe what the rules do. Use notation like that used in the class notes.
- Your grammar must contain at least the following actions: branching, rotation, scaling, push / pop, drawing leaves, and drawing branches.
- Branch and leaf geometry must be 3D. Branches should at least be tapered cylinders, and leaves should have some depth and shape to them.
- You must do all your own modelview matrix math. You are allowed to load the modelview matrix into OpenGL using glLoadMatrix(), but calls to glRotate(), glTranslate(), glScale(), glPush/PopMatrix(), gluLookAt(), and so forth are prohibited. You are allowed to use an external matrix library for this, and we’ve included glm for you. You are allowed to use gluPerspective() to set up your projection matrix, however.
A chunk of your grade will be based on the visual quality of your tree. If your tree looks terrible, so will your grade. To make your tree look good, try looking up different types of trees online, and see if you can design your grammar to generate similar results. Play with the parameters and mess with the rules until it looks decent! Adding some randomization to the parameters as the tree is generated can make things look much better with very little work.

Note that you do not need to actually implement an L-system parser. Just implement your system directly in code by creating a function for each one of your grammar’s rules and having them call each other recursively. Be sure to cap the recursion at some maximum depth.

2 Camera control

Adapt your orbit camera from the previous project. You will need to rewrite it to replace the OpenGL matrix functions with your matrix library, but its operation should stay the same. Like last time, use perspective projection. Make sure that the default camera shows a good view of the tree.

3 Rendering

Use the same rendering setup as was used in the lighted mode of the last assignment. This will generate some diffuse shading and give your trees a sense of depth. Note that you must generate and submit normals along with the rest of your geometry. Also, pick some reasonable colors for your geometry; OpenGL’s default flat grey is too boring.

4 Screenshot

5% of your grade in this project is a screenshot that you submit along with your code. Generate the best tree that you can and we’ll put the results up in a gallery after the projects have been submitted.

5 Bells & whistles

If you made it through all that, you’ve finished the project. No additional features are required, however, there are many ways to make it more awesome if you’re so inclined. If you impress the grader, it’s possible to earn a good amount of extra credit. Here are some suggestions:

- **Fruit / flowers**: Most trees sprout more than just leaves. Add some fruit, flowers, seed pods, pinecones, Spanish moss, roots... get creative! You can potentially combine this with your model loader code from the last assignment to get interesting effects.

- **Background**: A tree floating in a plain white background just looks odd. Add some surroundings for it, like the ground, a skybox, a background image, etc. If you want to get really fancy, you can add heightfield-based terrain.
• **Textures:** Solid-colored leaves and branches don’t look very realistic. Add texture mapping to make them look better! The starter code includes a PPM image loading function `readPPMfile()` for this purpose, although you’ll have to do the rest yourself.

• **Forest:** Why stop at just a single tree? Generate and render a bunch of different trees to get a forest. You’ll quickly end up with too much geometry, so make sure to do some caching and culling to only render the things you can actually see.

• **Multiple types:** Different grammars generate different looking plants. See how many types you can create. Pines, poplars, willows, shrubs, cacti, and many other species are possible.

• **Animation:** Adjusting the parameters used to generate the tree over time will result in animation. Experiment with this to see if you can achieve different effects such as growing, blowing in the wind, etc.

**Skeleton code**

Starter OpenGL code is provided on the class webpage, which should run under Linux, Mac OS X, and Windows. The project sets up a simple GLUT project, a basic perspective projection, and a rough framework that you can fill in for your tree generator. Substantial modifications will be necessary to complete your project, so don’t be afraid to rearrange the functions, change the parameters being passed around, and so forth.

**Logistics**

Like all projects in this class, you may work in a group of up to 2 people.

Any clarifications and revisions to the assignment will be posted on the class web page and announced in class.

In case you needed a reminder: **START EARLY.** This project will take a lot of tweaking to get right, even once you have all the basic code in place.

**Turnin and grading**

You can develop your application on whatever operating system you like, but before you submit it, you **must** make sure that it builds and runs properly on the department Linux machines! All grading will take place on these machines, so if your code doesn’t work on them, you’re in trouble.

To submit your code, use the department’s turnin script, like so:

```
turnin --submit agrippa proj2 treegen/
```
Replace `treegen/` with whatever your code directory is named. Make sure that *all* the necessary code is submitted, as projects that don’t build are worth nothing! On the other hand, make sure that you’re including just the files for this project, and not several hundred megs of extra junk.

Make sure you have included your (and your partner’s) name and UTCS ID in a comment at the top of each of your files. Also, include a readme explaining the usage of your program, including any menu options or keyboard commands. If you use any slip days, be sure to put that in your readme and email the TA as well.

To get a grade on this project, you’ll need to sign up for demo session with the TA after you’ve submitted your code. This is when you’ll run through all the functionality in your project, and the TA will be able to verify that everything is working as it should. This is also your chance to show off your extra features and slick interface!