Final Project

Due Date (Part 1): April 8 (30% of the project grade)
Due Date (Part 2): May 3 (70% of the project grade)

Purpose

This project shows how a type of context-free grammar (CFG) can be used to generate virtual plants. This is how many plants and other items with similar patterns are created in computer-generated animated movies. Your program will use the L-System method for generating the plants. It will then create and display new strings generated by the grammar; the individual symbols in the strings represent different parts of the plant. From these strings, your program will produce a file that represents a three-dimensional plant that can be viewed in a suitable application program and printed on a 3D printer.

This project can be completed individually or in a group of no more than three students.

Problem

Pixart is really good at generating animated movies. Now, they would like to create a new ride at DishneyWhorld. To do this, they need many props to create a virtual world for visitors. You have been hired, for many thousands of dollars, to write a program that will generate 3D plants suitable for printing. Furthermore, they want you to create and print a small version of such a plant as a prototype for the park.
To create such a plant, the program should use a Bracketed L-System. The program will read a file containing a plant's definition, including its grammar. The program will ultimately produce a file suitable for viewing and 3D printing.

Input

The program should prompt for an input file name. The input files will be called `input`i`.in`, where i = the number of the input file. For example, if you have four input files, the last one will be called `input4.in`. You must supply **at least** three input files defining three *very* different plants.

The file will contain data for creating a plant. At minimum, the parameters that should be defined include:

- the angle θ used for branching
- the length of a branch
- the width of a branch
- an L-System grammar, including **at least** $+$, $-$, and [ ], as well as a start symbol, terminal symbols, and non-terminal symbols

Many other parameters are possible, and are up to you. The more you have that can create better plants, the better your grade.

For Part 1, all that is needed in the file is the grammar.

Output

Part 1: The program should first display what each symbol in a string means (e.g., "B means 'branch'"). The program should then display the string produced by the grammar, **one iteration at a time.** Each time the user presses the `Enter` key, the program should display a new string

with created with an additional iteration. Thus, the plant should "grow" with each press of the key. This process should stop when a "q" is input.

This part must work properly for the entire project to be successful!

Part 2: In addition to generating the strings as in Part 1, the program should now generate a graphical object. The object should incorporate the parameters defined in the input file. The output is a Standard Tessellation Language (STL) file. This file can then be used to view your plant in any free online viewer **and** be used to print a 3D object.

Implementation

In order to get a valid object from the 3D printer, the virtual object defined by the STL file must be "water tight;" that is, the object must be solid, with no gaps. This will be tricky to implement. To get the final STL file that will be printable, follow these steps:

1. Create a 2D model of a plant, following the string produced by one of your grammars. This should be made up of simple rectangles.
2. Fill in the gaps between the rectangles with triangles, such that the vertices of each triangle are shared with the adjacent rectangles.
3. Tessellate the rectangles into triangles.
4. "Extrude" the sides of the original rectangles and the triangles to form 3D rectangles/triangles.
5. Tessellate all the new surfaces.
6. Calculate the surface normal for each triangle.[1]
7. Store all of the data in a file in STL format.
8. View your plant!
9. Print your plant!

Specifics

- The brackets [ ] in your grammar must be used for pushing and popping.
- You must supply at least three different plant grammars.
- Much of the specifications of what goes into the input file (and ultimately how your program runs) is up to you. In order for Pixart to understand your program so that they can create their own plants and install them at DishneyWhorld, you must write a several-page, nicely formatted "User's Guide." This guide must contain the following:
  - Directions on how to compile the program.
  - Directions on how to run the program.
  - How the input files are specified; i.e., how the initial values are input, how the production rules must be set up, etc.

[1] This may be optional; stay tuned.

  - Detailed descriptions of valid symbols (terminals) and what they accomplish; e.g., "B creates a branch with length l, width w, and thickness t." Of course, the items l, w, and t must then be defined somewhere as well.
  - Your three sample input files.
  - Graphical output of the three sample input files.

  When writing this guide, think about how often you've had trouble understanding an application's instructions. Make yours better! The easier it is for Pixart (me!) to understand, the easier it is for me to run your program, and the better your chances for a good grade.

- Your program should follow good OOP practices. Design good classes, etc.

Notes

Your grade is partly dependent on how nice your plants are, as well as the usability of your system; that is, how easy is it for Pixart to create new plants from the grammars your system can understand. Take time to design your system before trying to implement anything.

You can get bonus points if you create "flowers" or "leaves" on your plants. However, be sure everything else works before you attempt this!

Part 1 Deliverables: Send in your source code and your three input files via email, as usual. The source code file(s) and the input files should be zipped together, and named with a group member's last name and "Part1," such as `gousiePart1.zip`. Turn in your hard copy in class the next day. Also turn in a one-page, word-processed document that shows the names of the group members and describes the grammars in the three input files.

Part 2 Deliverables: Send in your source code and sample input files as above, except zip file name should end with "Part2.zip." In class the next day, turn in your User's Guide and **a 3D printed version of one of your plants.** You do **not** need to turn in hard copy of your code.

Don't just understand technology, but know how to use it understand your data.
– Janet Reno, 11/9/00, at Wheaton College