Adam Joyce

Coursework 1 – L-Systems

Maths and Graphical Effects

MSc Computer Games and Entertainment

18/11/2015

**Coursework 1: L-Systems**

This document is designed to give the reader a description of my first Maths and Graphics L-systems coursework on the MSc Computer Games and Entertainment at Goldsmiths.

**Structure Overview**

There are a number of different files included in my submission, the key three are:

* l\_systems\_generator.h
* l\_systems.h
* data1.csv – data8.csv

The l\_systems\_generator.h file contains the code that I use to read in the l-system parameters and generate the axiom for that stage of the L-system’s iteration.

The l\_systems.h file contains the main portion of code and is responsible for drawing and displaying the L-systems at each iteration as well as responding to input from the user and altering the L-systems appropriately.

The data1.csv file through to the data8.csv file are used to store the initial parameters for the eight different L-systems.

**Project Objectives**

My four main project objectives were as follows:

* Read configuration files containing L-System parameters
* Display resulting L-systems allowing for iterative steps to be displayed using hotkeys
* Hotkey each of the eight configuration files in order to easily load and display each L-system
* Hotkey the increment and decrement of three parameters that effect the structure of each L-system

**Reading the Configuration Files**

Each CSV configuration file is made up of five different L-system parameter categories:

* Variables
* Axiom
* Rules
* Angle
* Iterations

The reading in of each data CSV configuration file is managed by my ‘l\_systems\_generator’ class in the ‘l\_systems\_generator.h’ file. The class uses its ‘read\_data()’ function to parse the CSV files and store each of the parameters of the categories mentioned above. I included identity headings in the CSV files to make it easier to identify the rows that contain each parameter category.

**Displaying the L-System**

Once the initial axiom has been read from the a CSV file each character in that axiom is parsed by the ‘parse\_axiom()’ function in my ‘l\_systems’ class. ‘parse\_axiom()’ loops through each character in the axiom interpreting its meaning using a switch statement. Each case in the switch statement corresponds to a constant in the L-system, with the default clause skipping over any unevaluated progression values meant only for axiom expansion.

To allow the position and angle of each line drawn in the l-system to vary I use a stack to store both the position and angle variables of the line (or ‘node’). For the stack I use a simple ‘dynarray’ containing instances of my data class ‘tree\_node’.

As mentioned, an instance of ‘tree\_node’ contains only two variables - a vec3 position, and an angle. These variables are meant to signify the position and angle of the node at which the next line is to be drawn.

This ‘tree\_node’ stack allows for the branching that is present in the majority of the L-systems rendered. It does this by creating, and pushing, a new ‘tree\_node’ instance onto the stack whenever a ‘[‘ character is encountered in the axiom. The parser is then free to continue drawing along this new branch without worry that the branching position or angle will be lost. Once the parser then reaches the end of the branch, signified with a ‘]’ character, it pops the last tree\_node instance of the stack and sets the current position and angle equal to the previous node’s.

The remaining constants in ‘parse\_axiom()’ function are ‘+’ and ‘-‘, which alter the angle of the node and thus line, and ‘F’, which determines the colour of the line and calls ‘draw\_line()’ to actually draw it.

**Iterating**

As my ‘parse\_axiom()’ function only evaluates and draws a single string of characters, for iteration, I took the approach of generating the next stage of the system, refreshing the display, then passing this new axiom in to be parsed.

The expansion of the existing axiom is handled by the ‘next\_iteration(…)’ function in the ‘l\_systems\_generator’ class.

I wanted to not only be able to iterate forward through the steps in the L-system but also backwards through the steps. As such during each iteration I add the previous iteration string on the end of an array to be stored. It is then a simple case of setting the current axiom to equal the last string in the list.