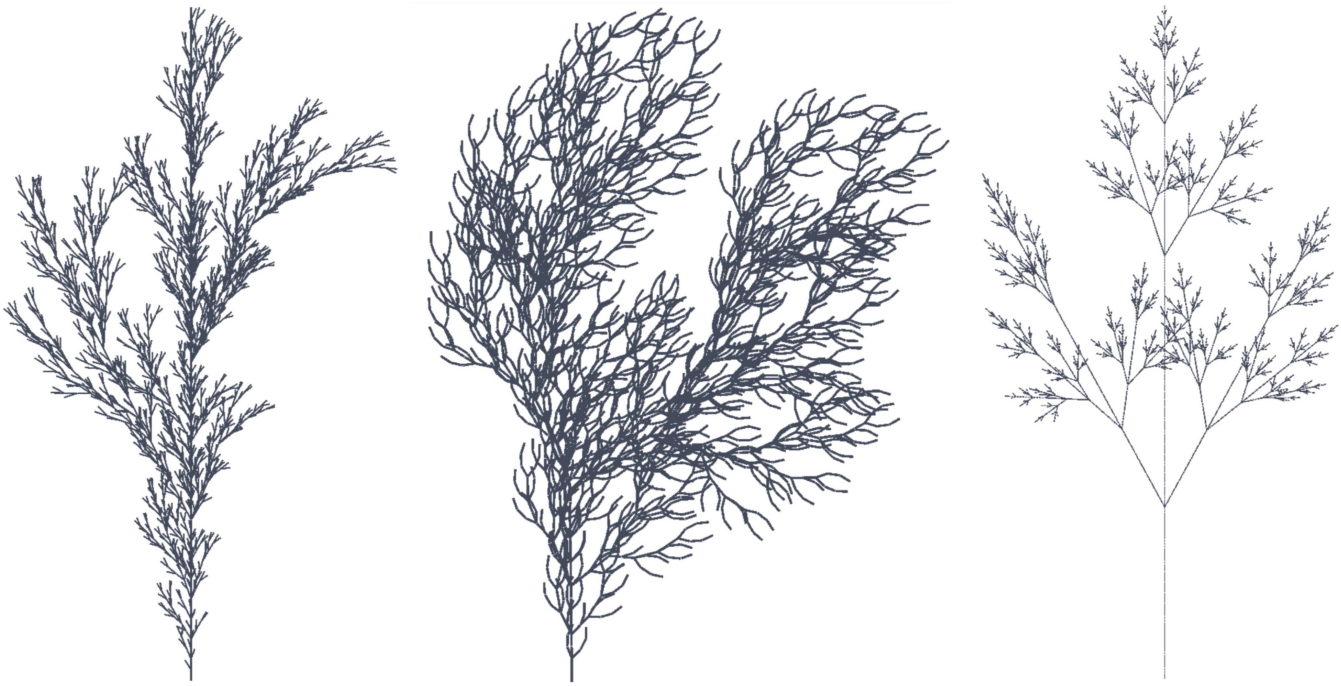
Mathematics and Graphics for Computer Games 1

Assignment 1
5 December 2016

L-Systems



Contents

Implementation Description

Code repository and video walkthrough

github.com/MatthewDuddington/MG1_01_LSystems

youtube.com/watch?v=FISyxHFgSno

High level overview

main.cpp

This is essentially a duplicate of the *Octet* basic app launching boilerplate code used within the earlier *Invaiderers* example. It simply serves to set up *Octet* specific support elements, before loading the L-System application itself, i.e. l_system_app.h

#includes are ordered to prevent declaration dependency problems.

Class: ***LSystemApp***

Handles the core setup and looping of the application. Hotkey input is listened for and resolved with calls to relevant classes within demarcated sections of the main loop. Calculates and sets the camera's position based on feedback from the tree. Manages the resetting of the application. Delivers user feedback via printed messages to the console. Initiates the draw calls, clearing of screen between frames and general OpenGL hooks.

Originally the main app utilised program states to differentiate the act of setting up or modifying parameters of the tree from the actual drawn state of the tree. However, after some later testing demonstrating that the overhead of re-drawing the tree was minimal at high enough orders, the code was refactored to enable ‘live’ editing of the tree design via the hotkeys.

It also stores the Tree class instance that is being drawn from.

Class: ***Tree***

Contains many properties that relate to a specific instance of a tree in the application, including the Recipe class that the Tree receives its draw seed from and the vector of Branches that define the tree visually.

Also defines the conceptual 'turtle' structure, which pushes and pops its position and rotation into either a matrix or Euler style record (depending on mode setting). Its main function GrowTree() contains the functionality which interprets the recipe’s seed, to define how the turtle is manipulated in order to decide where branches are to be laid out in order to construct the tree.

Alternate paths through the main function allow for process behaviour alterations for different modes, such as the rotation or turtle step.

Class: ***Branch***

Similar to the *sprite* class in *Invaiderers*, the Branch class is a renderable object within the application canvas. When creating a new branch, it self-pushes to the branch vector. It contains the render function to talk with OpenGL.

Class: ***Recipe***

Responsible for processing the design parameters and building the recipe seed string or evolving it according to the given rules.

In the application's current state, this arguably overlaps with the function of RecipeBook as most of the unique properties of the 'current' recipe were stored in RecipeBook's Designs vector. However, were the application to need to draw multiple trees at once, then those properties would be better relocated within the Recipe instance itself and, thus, it has remained within my codebase.

Class: ***RecipeBook***

Defines the structure of a TreeDesign and Rule.

Rules cover both Constants and Variables in the same container. Constants replace themselves with an identical character during processing, whereas Variables are replaced by a string of other characters.

Stores the vector of prepared designs that is read from to set up new or refreshed recipe parameters.

Handles the reading of external design files. The main processing of the ImportDesigns() function has been observed to work correctly (see segments marked 'simulate input file'), however, an issue has been encountered with Visual Studio referencing the root project folder and thus it is unable to load the specified files. Instead, this is being emulated via loading of those same files from directly within the lower half of the RecipeBook class.

Designs[0] contains the attributes being used by the live tree.

Process and Learning

High level objectives

Initial objectives

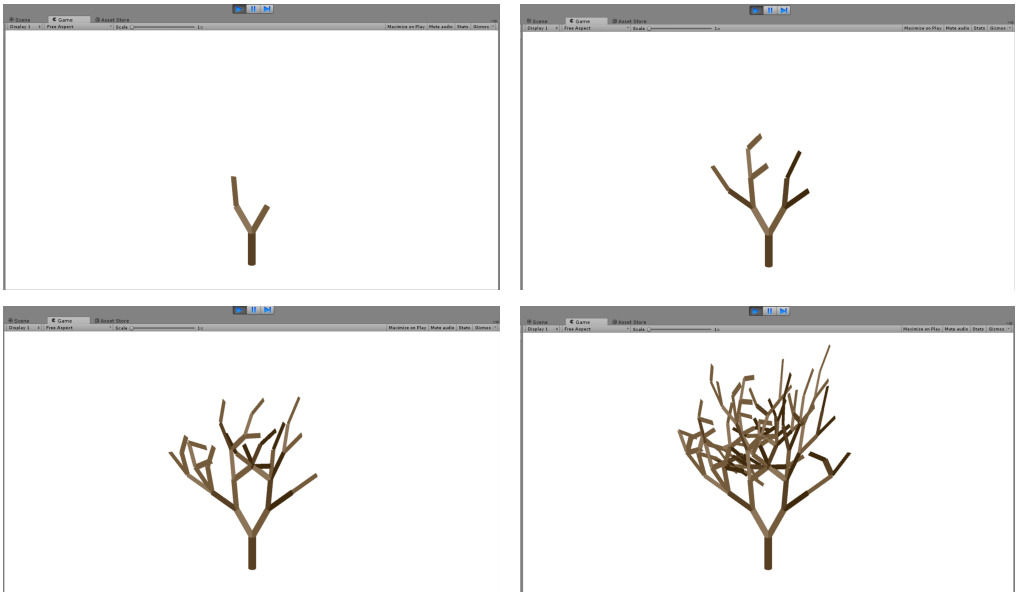
- Prototype initial functionality using Unity (as I am more familiar with it).
- Produce main application using C++ and OpenGL within Visual Studio.
  - Utilise learning from *Introduction to Programming* module and *Octet* to enable this.
  - Render content to the screen dynamically and receive / act upon input from the user.
- Separate code into intuitive concepts via different classes of functionality.

Progress objectives

- Draw the entire tree as a single mesh rather than spawning many objects.
- Reduce reliance on accessing functionality through static methods (with a view to anticipating my approach needs when dealing with multithreading in later work).
- Include some elements of stochastic behaviour.
  - Branch angle and length.
  - Leaves / fruit spawning.
- Include setting to display the stages of drawing so the way in which the tree seed string is processed by the turtle can be more easily visualised.
- Experiment with alternative ways of interpreting the seed string commands.

Process Record

Early in the semester I was able to get to work with prototyping a very basic implementation using Unity. Through this, I wanted to establish an understanding of the general process I would need to take in order to deal with the l-system seed strings and how I would adapt the 'turtle graphics' concept. Using the basic example recipe at the beginning of *The Algorithmic Beauty of Plants* (Prusinkiewicz & Lindenmayer 2004), the prototype stepped through the recipe using the spacebar and could handle around 5 or 6 orders before Unity became overloaded with my inefficient mesh spawning.



First 4 orders of the Unity prototype tree. *Axiom:* A_1 $A_1 \rightarrow A_2 B_1$ $A_2 \rightarrow B_2 A_1$ $B_1 \rightarrow A_1$ $B_2 \rightarrow A_2$

Despite an inaccuracy in how it interpreted the string (there was an extra step taken on some branches due to how Unity handled the event calls), the general process of stepping through the seed worked and thus I was confident I could adapt this once I began in the C++ version.

Following this early progress, I was able to build an initial version of the seed evolution and interpretation part of my C++ application. While in the Unity prototype the behaviour of the growth had been hard coded, in contrast, this new code was much more adaptive. The framework used a rule structure to understand how to recursively evolve the seeds it was given and could pass this back into the early stages of the tree class, which was planned to eventually handle the actual drawing behaviour.

At this stage I hit the first of my main challenges. As this was still early in the semester, I had not yet learned how to achieve the instantiation or rendering of elements to the screen through OpenGL or Octet. So, for some time, my attention was taken up with the *Introduction to Programming* assignment, that would gradually teach me those skills.

Initially, I attempted to approach the rendering by adapting the Octet simple triangle drawing example, but later chose to make an adaptation of the renderable object concept from the *Invaiderers* project. This was largely due to my second challenge, that creating a single mesh for the entire tree proved to be much more complex than I had anticipated. Ordering the vertices such that the ends of branches would not draw back on themselves when returning to earlier branch points was not something I was able to understand within the time I was comfortable in spending considering the problem. Therefore, I chose to return to treating each forward-drawing command as an individually oriented object - which, because I was no longer using Unity, was exponentially more lightweight and thus a viable solution.

Progress with this approach was good and I was able to take an input recipe of rules, axiom and additional properties and process it through the drawing interpreter into a rendered image. The need for adaptive positioning of the camera became an issue, though one which was relatively easy to solve sufficiently for a first stage, by offsetting the size of the tree with a fixed increment. Later, this would be replaced by the much more robust approach of recording the outer limits of X and Y in each direction during drawing and using this to scale and position the camera immediately before rendering to the screen.

Around this time, the first of the three significant issues I encountered appeared. For reasons I have, to date, been unable to identify, the application was experiencing an intermittent crash from an out of bounds call being made on a vector as a part of the Octet mat4t class (a 4x4 matrix functionality object). What was unusual and difficult about this issue was the intermittent nature of its manifestation. This issue persisted for quite some time, until I was able to work around the issue by disabling some functionality of the class at the stage where a problem was being caused.

Concurrently with attempting to diagnose the intermittent crash, I encountered the second significant issue of the project. All seemed well with the approach I had taken to processing and keeping track of the locations and rotations of the conceptual ‘turtle’ during the drawing process, as the first of the example trees was drawing correctly. However, once I attempted to display trees C and F I discovered that these were being resolved incorrectly.

By hand iterating their seeds, I was able to confirm that my string builder was working accurately and producing the right recipe (which later would be confirmed by my second drawing approach outputting the correct versions of the trees). This suggested that the way in which I was utilising the Octet mat4t class meant that these specific trees’ seeds were not followed in the expected way by the turtle, even though the seeds for trees A, B, D and E were. It was unrelated to the disabled code as part of resolving issue one.

Trees C (top pair) and F (bottom pair), in incorrectly calculated forms (left), and the correct forms (right) which were achieved later.

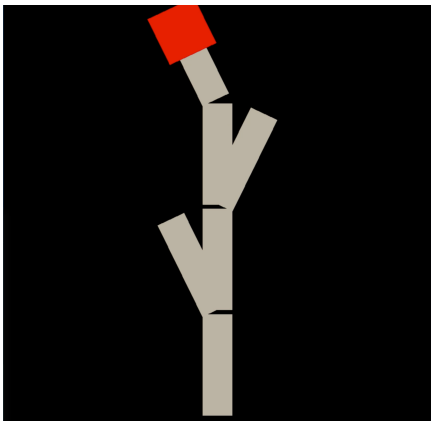
In order to resolve this second issue I developed an alternative method for storing and making use of the turtle's position and rotation. Rather than manipulating matrix objects, I broke down the transformations into an Euler [XYZ] position vector and a single Euler [Z] angle. From this it was possible to calculate, using Polar to Euler conversion, the changing position of the turtle.

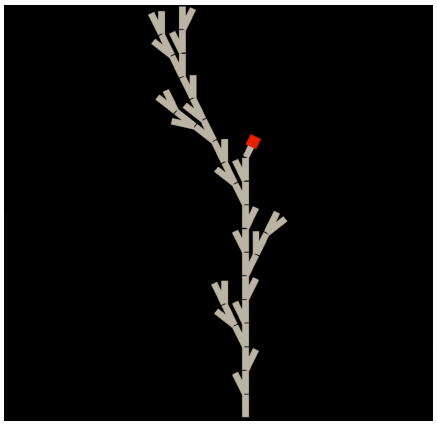
As a part of the process of diagnosing the miscalculation I developed a couple of extensions to the expected functions of the application. The first of these was a pair of alternative methods of resolving the rotation of the turtle following a 'load' step.

The normal pattern is to reload the rotation that was present at the moment of the 'save' step. My first alternative was to load the saved position but zero out the rotation (i.e. return the turtle to facing North), the second alternative was to load the saved position but preserve the rotation held before the 'load' step. Both of these yielded some interesting alternative renders of each tree. Some of these loop around on themselves, others become a single long string and, in some cases, the tree barely changes at all.

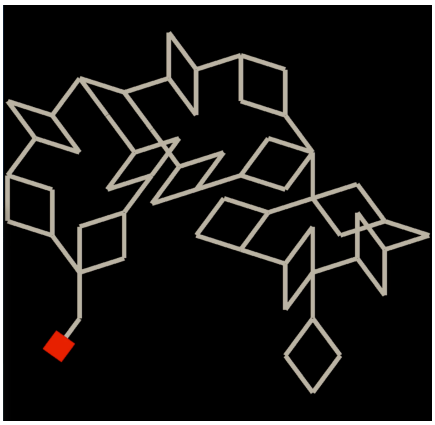
Trees A, B, D and F using the three rotation modes: Normal, Zero out, Preserve.
Several of the effects are reminiscent of the effect of a genetic mutation, causing an unstable cell formation.

As well as the different rotation modes, the second extension was to implement a step-by-step visualisation of the processing of the seed string by the turtle. In 'turtle step mode' the turtle's position and rotation are visualised with a red square, whilst each branch, rotation and load step is drawn over time. This greatly helped me in understanding exactly where in each string the different parts of the tree are drawn and how the different patterns of recursive-like behaviour played out.





Turtle step mode on tree A early in the drawing process and later on.



Part way through design 9, Penrose Tiling.

The third significant issue that I encountered was in addressing external files. This was a process I successfully carried out for the *Introduction to Programming* project, however, this particular Visual Studio project did not seem to be able to load either the font files or design files in the assets or root folder. To date I have been unable to solve this issue which meant that I have had to rely on displaying instructions and feedback for the user to the console rather than printing into the application canvas.

```
INSTRUCTIONS:
  Hold 'L' and press number from 1 to 9 to load the prarameters of a
  predesigned LSystem

  Hold 'P' (increment) or 'O' (decrement) and press the numbers '1' - '5' to
  change the following recipe parameters:
      1 -> Order      +/- 1                       Currently: 4
      2 -> Rotation Left  +/- 0.5 degrees         Currently: 36.0 deg
      3 -> Rotation Right +/- 0.5 degrees         Currently: 36.0 deg
      4 -> Branch Circumference  +/- 0.05 meters  Currently: 0.10 m
      5 -> Branch Thinning Ratio +/- 0.05 percent Currently: 1.00

  Hold 'I' and press the corosponding parameter number to make it stochastic:
      3 -> Branch Rotation
      5 -> Branch Length
      7 -> Leaves (experimental!)

  Press 'T' to toggle 'turtle step mode', which draws the branches one at a time
  Press 'M' to toggle matrix mode (experimental!) and polar mode (stable)

  Hold 'R' when in matrix mode and press 'A', 'S' or 'D' to switch how rotation
  is handled during 'load' drawing steps:
      A -> Rotation from saved position will be loaded upon load position
      S -> Rotation will be zeroed out upon load position
      D -> Current rotation will be preserved upon load position

  Press 'H' to print these instructions to the console again
********************************************************************************
```

Console based instructions and feedback.

It also meant that my file interpreter function LoadDesigns() in the RecipeBook class, whilst functioning perfectly with a simulated input file directly in code, could not be used with the external files, as it was being passed an empty ifstream. Instead my designs are loaded via the LoadDesignsManually() function. However, the method underlying the way in which the external file loading code works is otherwise correct. My intuitive sense is that there is a mismatch in how Visual Studio is processing the root directory.

I had intended to include a variety of stochastic elements in the final application, from which I successfully achieved randomising branch angles and lengths, within fixed ranges, that are independent to each design. Unfortunately, I did not mange to include leaves / fruit spawning or colour variation before the deadline. However, I remain curious to add these in the future.

Stochastic branch angle and length on tree C, creating a much more organic look.

Bibliography

10K L-Systems Blog. *Examples*. (accessed November 2016) Available online: 10klsystems.wordpress.com/examples/

Dickheiser, M. J. (2007) *C++ For Game Programmers*. 2nd ed. Charles River Media.

Dunn, F. and Parberry, I. (2011) *3D Math Primer for Graphics and Game Development*. 2nd ed. CRC Press.

Meyers, S. (2015) *Effective Modern C++*. 1st ed, 4th release. O'Reilly Media.

Prusinkiewicz P. & Lindenmayer A. (2004) *The Algorithmic Beauty of Plants*. Electronic version, reproduced from print: (1996) Springer-Verlag, New York. Available online: algorithmicbotany.org/papers/abop/abop.pdf

Thomason, A. (2012) *Octet*. C++ development library. Available online: github.com/andy-thomason/octet

Wikipedia. *L-system*. (accessed September 2016) Available online: en.wikipedia.org/wiki/L-system