CSC 205 - SPRING 2016
2D GRAPHICS AND IMAGE PROCESSING
ASSIGNMENT 3
UNIVERSITY OF VICTORIA

Due: Tuesday, March 1st, 2016 by noon.

1 Assignment Overview

A *Lindenmayer system* (or *L-System*) is a formal grammar which can be used to model certain types of recursively generated systems. The formal aspects of L-systems are more relevant to CSC 320 (since L-systems are similar to context-free grammars) than a graphics course, but L-Systems are often used in a graphics context as a way to model detailed natural systems like plants and trees. The goal of this assignment is to use L-Systems to study coordinate transformations. Although some of the technical aspects are beyond the scope of this course, the examples on the Wikipedia page for L-Systems (`https://en.wikipedia.org/wiki/L-system`) may be useful. The definition of an L-System presented here is simplified significantly from the version presented in the Wikipedia article.

The objective of this assignment is to implement a renderer for L-Systems. The basic functionality will render a simple tree-like structure (with branches and leaves) using geometry primitives and three basic transformations: translation, rotation and scaling. To recieve full marks, you will be expected to extend the basic functionality in some way (for example, by adding extra transformations, new parsing rules or different geometric primitives, or by designing an interesting new L-System).

You may implement your solution in any language that can be run (using non-proprietary software tools) in ECS 354, ECS 242 or ECS 266. Starter code has been provided in C++, Java and Python. All three versions of the starter code are based on the samples provided for Assignment 2, so the C++ and Python versions use the SDL for rendering. The starter code contains a parser to perform the L-System productions (see Section 4). It should not be necessary for you to modify the parser (although you may do so). If you want to use a language other than C++, Java or Python, you will have to adapt the parser yourself. However, unlike in Assignment 1, the parser is not very complicated or difficult to re-implement. As with the previous assignments, you are not required to use any of the starter code in your final submission.

If you use a language which relies on platform-specific libraries, you may find that it is easier to complete the entire assignment on a lab machine, instead of working on it on your own machine, since it can be difficult to resolve library version conflicts in some cases. The sample code for C++ and Python has been tested on the ECS 354 machines and a recent Debian-based linux distribution. Talk to your instructor if you need help compiling or running the sample code on your own machine. The compile/build/run process for the starter code will be covered in the lectures and labs.

2 Externally-Sourced Code

You are permitted to use appropriately licensed code from an external source, if full attribution is given (including the name of the original author, the source from which the code was obtained and an indication of the terms of the license). Note that using copyrighted material without an appropriate license (such as the GPL, LGPL, BSD License or MIT License) is not permitted. Short fragments of code found on public websites (such as StackOverflow) may be used without an explicit license (but with the usual attribution). If you embed externally sourced code in your own code files, add a citation in your code file and include a README file detailing the sources of all code used. If you use entire, unmodified source files from an external source, a README file is not necessary as long as the file in question contains complete authorship and licensing information. If you submit the work of others without proper attribution, it will be considered plagiarism. Additionally, for assignments in CSC 205, the following two caveats apply to externally sourced code.

- You will not receive any marks for the work of others. That is, if you use externally sourced code to implement the core objectives of the assignment, you will only be marked on the parts of the code that you personally wrote. Therefore, you should only use externally sourced code in a supplementary capacity (such as including a third-party hash table implementation which your code uses to store data, or a third party Gaussian elimination package to solve matrix equations). If you have any doubts about what is considered supplementary, contact your instructor.
- You may not use externally sourced code from another CSC 205 student, or share your code with another CSC 205 student (whether they intend to use it or not), without explicit permission from the instructor.

3 Coordinate Transformations

A coordinate transformation in 2d can be represented by a 3×3 matrix using homogeneous coordinates, and transformations can be composed using matrix multiplication. Transformation matrices are a powerful way to manipulate coordinates during rendering, since, with the appropriate transformation, complicated objects can often be drawn with relatively simple logic. For example, programs which render 2d graphics may draw onto a canvas of an arbitrary size $w \times h$, which may have idiosyncratic characteristics that complicate rendering (such as the y values increasing from top to bottom, instead of from bottom to top). The coordinate system of a drawing window of dimensions $w \times h$ would normally have the point $(0, 0)$ at the top left and the point $(w-1, h-1)$ at the bottom right. In many graphics contexts, it is more useful to assume that the canvas is centered at the point $(0, 0)$ and that y coordinates increase from bottom to top. Using transformation matrices, it is possible to achieve this coordinate system. Consider the matrix M below.

$$M = \begin{bmatrix} 1 & 0 & w/2 \\ 0 & 1 & h/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w/200 & 0 & 0 \\ 0 & h/200 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix M is the product of a translation by $(w/2, h/2)$, a reflection across the x-axis (that is, a scale by -1 in the y-direction, which reverses the order of y coordinates) and a scale by $(w/200, h/200)$. If $v = \langle x, y \rangle$, where $x, y \in [-100, 100]$, then the vector $u = Mv$ will have coordinates

$u_x \in [0, w]$ and $u_y \in [0, h]$. If M is used to transform all coordinates before rendering, then the canvas can be treated as a cartesian plane from -100 to 100 in both dimensions. Such a transformation is often called the *viewport transformation*, since it adapts the coordinate system of the program to the coordinate system of the screen.

In the provided starter code, a similar viewport transformation has been applied already, except with the origin $0, 0$ at the bottom of the screen instead of the center. A special renderer has been implemented in the C++ and Python code to allow objects to be transformed with the active transformation matrix before rendering. The Java API contains the necessary functionality already,

4 L-Systems

For the purposes of this assignment, an L-System consists of two parts: an **axiom** and a list of **rules**. The axiom defines the set of initial conditions of the system and the rules define how the axiom will be expanded as the system iterates. The system must have exactly one axiom, but may have any number of rules (or no rules). The axiom and the rules are simply strings of text, which may contain any characters (except spaces and equals signs). The meanings of the characters are irrelevant to the system itself (although the renderer ascribes meaning to each character). Rules are formatted as equality statements, with a single character on the left hand side.

Consider the system below.

Axiom: A
Rules: A = ABA
B = C
C = xYx

A **production** is a string of characters which results from taking the axiom of the system and applying the rules as substitutions for a specific number of iterations. During each iteration of a production, the result of the previous iteration is scanned, from left to right, and for each character in the string, if there is a rule with that character on the left hand side, the right hand side of the rule is substituted for the character. The result of iteration 0 is assumed to be the axiom itself. The table below shows the result of iterations 0 - 3 using the system above.

Iteration	**Production**
0	A
1	ABA
2	ABACABA
3	ABACABAxYxABACABA

L-Systems are examples of context-free grammars, and the semantics of L-System productions are more appropriately discussed in a theory course like CSC 320. The starter code for the assignment contains a parser which reads an L-System from an input file, then provides methods to obtain a production for a given number of iterations (so you do not need to worry about the mechanics of L-System productions, although the code itself is not difficult). The interface allows the user to change the number of iterations by pressing the Up and Down arrows. The format of the input files is covered by Section 5.

You are expected to write code which takes the produced string and, by reading it from left to right, performs transformation and rendering operations. Your code must support the following characters and operations.

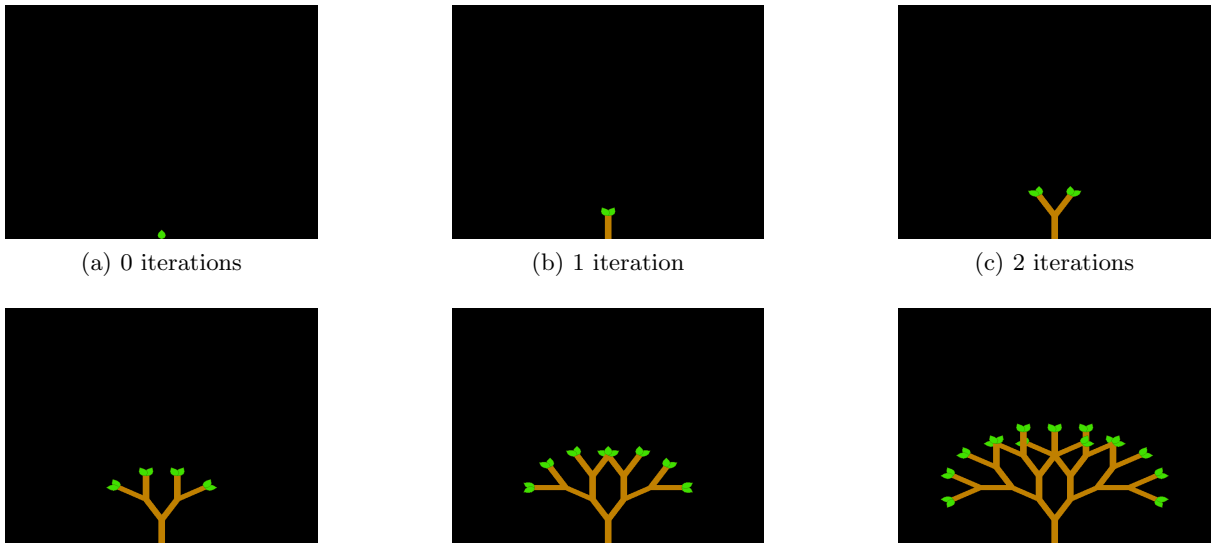
Character	Effect
`L`	Draw a 'leaf' at position $(0, 0)$ of the local coordinate system (code for drawing leaves has been provided).
`T`	Draw a vertical 'stem' or 'branch' at position $(0, 0)$ and multiply the local coordinate transform (on the right) by a translation by $(0, h)$, where h is the height of the stem.
`+`	Multiply the local coordinate transform on the right by a counter-clockwise rotation of 30 degrees.
`-`	Multiply the local coordinate transform on the right by a clockwise rotation of 30 degrees.
`s`	Multiply the local coordinate transform on the right by a scale of $(0.9, 0.9)$.
`S`	Multiply the local coordinate transform on the right by a scale of $(1/0.9, 1/0.9)$.
`h`	Multiply the local coordinate transform on the right by a scale of $(0.9, 1)$.
`H`	Multiply the local coordinate transform on the right by a scale of $(1/0.9, 1)$.
`v`	Multiply the local coordinate transform on the right by a scale of $(1, 0.9)$.
`V`	Multiply the local coordinate transform on the right by a scale of $(1, 1/0.9)$.
`[`	Save the current local coordinate transform.
`]`	Restore the last coordinate transform that was saved. You must be able to save any number of transform matrices (and each restore command should retrieve a saved transform in a last-in-first-out order).

You may add extra commands as needed. Your code should ignore any characters in the production which are not valid commands. Some L-Systems will deliberately use undefined characters as placeholders in substitution rules, and it is not an error for a production to contain undefined characters.

Figure 1 shows the result of rendering the L-System below (posted as `sample_tree1.txt`) with various numbers of iterations.

Axiom: L
Rules: L = T[+L][-L]

Note that the exact appearance of leaves and branches is up to you (as long as they are clearly visible, you do not have to make them look like the example).



(a) 0 iterations (b) 1 iteration (c) 2 iterations

(d) 3 iterations (e) 4 iterations (f) 5 iterations

Figure 1: Various renderings of an L-System.

5 L-System Input Files

The L-System input files contain the axiom on the first line and the rules on the subsequent lines. For example, the L-System displayed in Figure 1 would be encoded as follows.

```
L
L = T[+L][-L]
```

Any number of rules can be specified, and the substitution can contain any characters except the equals sign. Spaces are ignored. The parser provided with the starter code allows some extended notation to be used to define a finite lifespan for a rule, or to apply a rule only on certain iterations. The lifetime of a rule is specified by an integer at the beginning of the line. If the lifetime is n, then the rule will be applied up to iteration n, and then ignored. If the lifetime is a negative value $-n$, then the rule will be applied until iteration $t - n$, where t is the total number of iterations. A 'parity flag' can also be specified before the rule with the `%` and `^` characters. If the flag `%` is used, then the rule will only be applied on even numbered iterations, and if the flag `^` is used, the rule will only be applied on odd numbered iterations.

6 Evaluation

Submit all of your code electronically through the Assignments tab on conneX. Your submission should include a complete set of files needed to compile, run and demonstrate your implementation's functionality for marking. Your implementation will be marked out of 60 during an interactive demo with an instructor. You may be expected to explain some aspects of your code to the evaluator. Demos must be scheduled in advance (through an electronic system available on conneX). If you do not schedule a demo time, or if you do not attend your scheduled demo, you will receive a mark of zero.

You may receive up to 70 marks on this assignment, but the final assignment mark will be taken out of 60 (so any marks you receive beyond 60 will be treated as bonus).

The marks are distributed among the components of the assignment as follows. The term 'playing field' is used below to refer to the active drawing area; this should not be interpreted to mean that you are required to implement a game. If you have any questions about how the specifications below would apply to your intended implementation, ask your instructor.

Marks	**Feature**
25	Basic rendering: The `T` and `L` specifiers work as required, and L-Systems without the save/load specifiers `[` and `]` render correctly for any number of iterations. All of the required transformations work as required.
10	The save/load specifiers `[` and `]` work for any number of iterations and any depth of nesting.
5	Improved rendering: Gradient or textured rendering is used instead of flat colours.
5	Interesting inputs: New L-Systems with complicated behavior (and which are your own work) are included in your submission.
4	Extra geometric primitives (besides the branch and leaf) are implemented. Demonstration files must be included to show these in use.
5	The L-System is rendered more than once in different positions and orientations (for example, to produce a forest instead of a single tree). Transformations are also used to produce perspective or shadow effects.
8	The implementation can render L-Systems beyond the simple branch/tree models used in this specification (for example, you could receive full marks in this section by making your program render all of the examples in the Wikipedia article).
8	The implementation allows the window to be resized at any time by the user, and the drawing perspective and aspect ratio are scaled proportionally to fit the new window size.

Additionally, you may receive extra marks (with the total overall mark capped at 70) for other features of your own design, if you have received permission from the instructor before the due date.

Ensure that all code files needed to compile and run your code in an ECS lab are submitted. Only the files that you submit through conneX will be marked. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.