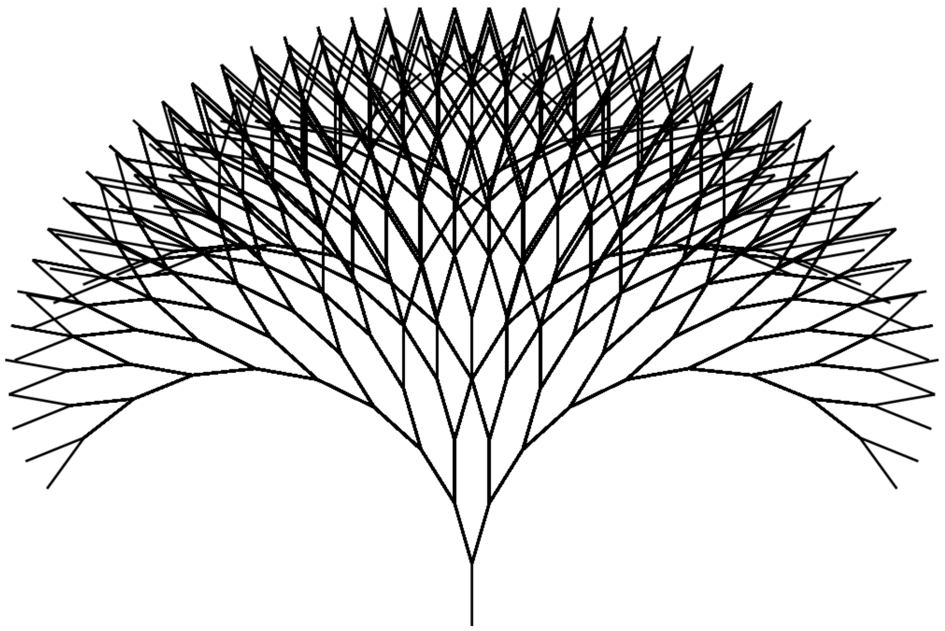
Procedural Plant Generation and Simulated Plant Growth

Massey University



Matthew Crankshaw

25 February 2019

Acknowledgements

Abstract

Contents

1 Introduction **5**

1.1 Context and Background . . . 5

2 Outline **6**

2.1 Outline . . . 6

3 Introducing Lindenmayer Systems **7**

3.1 Simple DOL-system . . . 7

3.2 Interpreting L-systems . . . 8

3.3 Branching Filaments . . . 9

3.4 Basic 2D L-systems . . . 10

3.5 The Use of L-systems in 3D applications . . . 13

3.6 Parametric L-system . . . 14

3.6.1 Formalising Parametric L-system Grammar . . . 14

4 Implementation **16**

4.1 Language and environment . . . 16

4.1.1 C/C++ Programming Language . . . 16

4.1.2 Standard Template Library (STL) . . . 17

4.1.3 Open Graphics Library (OpenGL) . . . 17

4.1.4 OpenGL Mathematics Library (GLM) . . . 17

4.1.5 Graphics Library Framework(GLFW) . . . 17

4.1.6 Git Version Control . . . 17

5 Glossary **18**

A Appendix **20**

A.1 Appendix 1 . . . 20

A.2 Bibliography . . . 20

List of Figures

3.1 Diagram showing a turtle interpreting simple L-system string. 9
3.2 Diagram showing a turtle interpreting an L-system incorporating branching. 10
3.3 Koch Curve. 11
3.4 Sierpinski Triangle. 11
3.5 Dragon Curve. 12
3.6 Fractal Plant. 12
3.7 Fractal Bush. 13

Chapter 1

Introduction

Here I will introduce the project and the thesis in general.

1.1 Context and Background

One of the most time consuming parts for digital artists and animators is creating differing variations of the same basic piece of artwork. In most games and other graphics applications environment assets such as trees, plants, grass, algae and other types of plant life make up the large majority of the assets within a game. Creating a tree asset can take a skilled digital artist more than an hour of work by hand, The artist will then have to create many variations of the same asset in order to obtain enough variation that a user of that graphics application would not notice that the asset has been duplicated. If you multiply this by the number of assets that a given artist will have to create and then modify, you are looking at an incredible number of hours that could potentially be put to use creating much more intricate and important assets.

The unique thing about plant life when compared with other types of graphics assets is that plant life is very random in the way it grows and it does not take an incredibly realistic model of a plant life to trick the human mind into believing that what it is seeing is some kind of plant. However what stands out like a sore thumb is when plants that are growing next to each other seemingly have no influence on the plant life around them.

Chapter 2

Outline

2.1 Outline

Chapter 3

Introducing Lindenmayer Systems

Aristid Lindenmayer is well known biologist who started work on what would become known as the Lindenmayer System or L-system for short. Originally L-systems were intended to be used as a grammar for describing the development of simple organisms. However over the years they have been found to be useful in describing larger organisms and even non organic structures like music. [Worth and Stepney, 2005]

In order to talk about the more complex L-systems such as the 2L-system which is used to describe structures like plants and trees. I first need to talk about how L-systems are described as well as how string rewriting works. The most simple form of L-system is a non parametric DOL-system.

3.1 Simple DOL-system

According to Prusinkiewicz and Hanan a simple type of L-systems are those known as deterministic 0L systems, where the string refers to the sequence of cellular states and '0L system' abbreviating 'Lindenmayer system with zero-sided interactions'. With 0L systems there are only three major parts. There is a set of symbols known as the (*alphabet*), the starting string (*Axiom*) and state transition rules (*rules*). The alphabet is a set of states. The starting string is a starting point containing one or more states. The transition rules are rules that dictate whether a state should remain the same or transition into a different state or even disappear. [Prusinkiewicz and Hanan, 2013].

An example of a deterministic 0L system:

We are given the *alphabet*: A, B
and the *axiom*: A
and the *rule* set:
A $\rightarrow$ AB
B $\rightarrow$ A

The symbol $\rightarrow$ can be verbalised as "replaced by". Therefor it can be said that the string 'A' is replaced by string 'AB' and string 'B' is replaced by the string 'A'.
To start we take the *axiom* which is this case is 'A' and we run through all of the states in this start string 'A' matches the rule: A $\rightarrow$ AB and is therefor replaced by 'AB'. 'AB' then becomes the new start string and we then match the rules once again. Below I have shown the resulting string

up to six generations.

If we then apply the rules to the L-system we find it creates the following generation structure.

1.) A
2.) AB
3.) ABA
4.) ABAAB
5.) ABAABABA
6.) ABAABABAABAAB

This rewriting of initial string using a set of rules is ultimately the underlying concept behind L-systems. There are a number of improvements that can be made to this type of L-system in order to accommodate for more complex and intricate structure. One of which is the inclusion of *constants*. Constants can be considered any state that does not have a rule associated with it or remains the same from generation to generation and therefore holds a consistent value or meaning. These constants are used when the L-system is interpreted and thus holds a constant value during string rewriting, I will be covering this in section 3.2.

Prusinkiewicz and Lindenmayer simulated a blue-green bacteria known as *Anabaena catenula* [Prusinkiewicz and Lindenmayer, 2012]

The DOL-system is described as follows:

$w : a_{\mathrm{r}}$
$p1 : a_{\mathrm{r}} \rightarrow a_{\mathrm{l}}b_{\mathrm{r}}$
$p2 : a_{\mathrm{l}} \rightarrow b_{\mathrm{l}}a_{\mathrm{r}}$
$p3 : b_{\mathrm{r}} \rightarrow a_{\mathrm{r}}$
$p4 : b_{\mathrm{l}} \rightarrow a_{\mathrm{l}}$

The value w is there to specify the axiom which is this case has the value of a_{r}. *p1*, *p2*, *p3*, *p4* are the names of the rules that follow after the semi-colon. In order to simulate Anabaena catenula we need four rules.
According to Prusinkiewicz and Lindenmayer "Under a microscope, the filaments appear as a sequence of cylinders of various lengths, with a-type cells longer than b-type cells. And the subscript l and r indicate cell polarity, specifying the positions in which daughter cells of type a and b will be produced. [Prusinkiewicz and Lindenmayer, 2012]

This gives us a good real world demonstration of how even a simple DOL-system can represent something in the real world.

3.2 Interpreting L-systems

Once we have generated the set of rules that allow us to create the L-system we are left with a string of characters which represent that particular L-system. As with any grammar, there is a number of ways of interpreting the string that is generated by the L-systems rules. One method proposed by Przemyslaw Prusinkiewics is "to generate a string of symbols using an L-system, and to interpret this string as a sequence of commands which control a 'turtle'". [Prusinkiewicz, 1986]

A two dimensional L-system string may hold the following commands in the form of symbols.

- F: Move forward by a specified distance whilst drawing a line
- f: Move forward by a specified distance without drawing a line
- +: Rotate to the right specified angle.
- -: Rotate to the left by a specified angle.
- [: Save the current position and angle.
-]: Load a saved position and angle.

Similarly a three dimensional L-system string may hold the following commands in the form of symbols.

- F: Move forward by a specified distance whilst drawing a line
- f: Move forward by a specified distance without drawing a line
- +: Yaw to the right specified angle.
- -: Yaw to the left by a specified angle.
- /: Pitch up by specified angle.
- \: Pitch down by a specified angle.
- ˆ: Roll to the right specified angle.
- &: Roll to the left by a specified angle.
- [: Save the current position and angle.
-]: Load a saved position and angle.

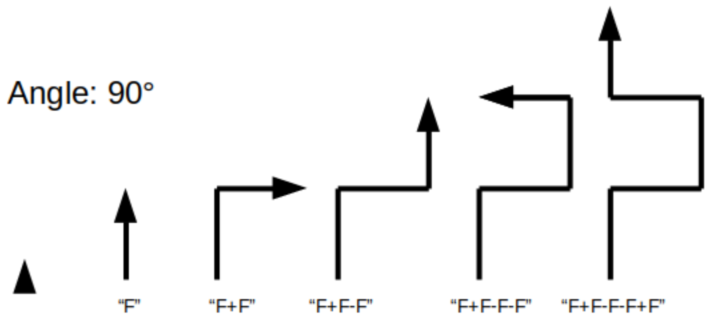


Figure 3.1: Diagram showing a turtle interpreting simple L-system string.

3.3 Branching Filaments

Explain how branching works and how it can be used for generating the L-systems

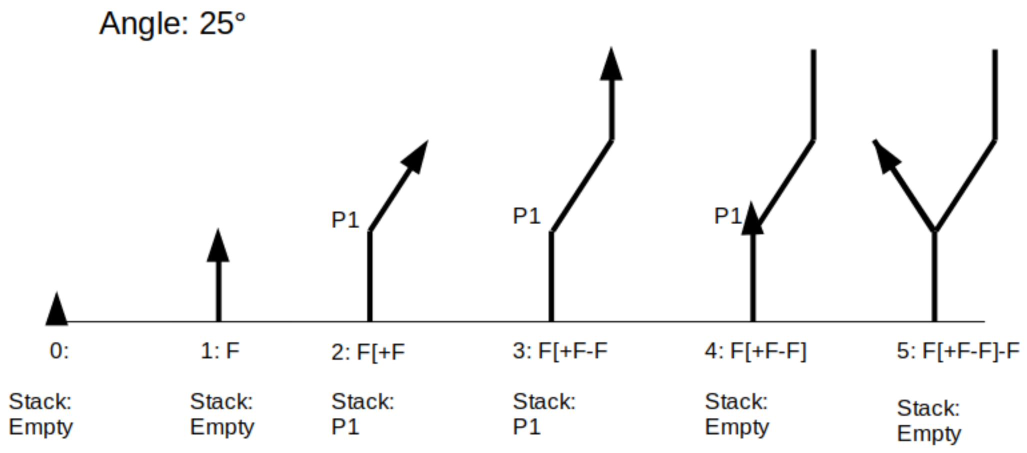


Figure 3.2: Diagram showing a turtle interpreting an L-system incorporating branching.

3.4 Basic 2D L-systems

There are a number of fractal geometry that have become well known particularly with regards to how they can seemingly imitate nature [Mandelbrot, 1982]. Particularly with the geometry such as the Koch snowflake which can be represented using the following L-system.

<u>Koch Curve:</u>
Alphabet: F
Constants: +, -
Axiom: F
Angle: 90°
Rules:
F → F+F–F+F

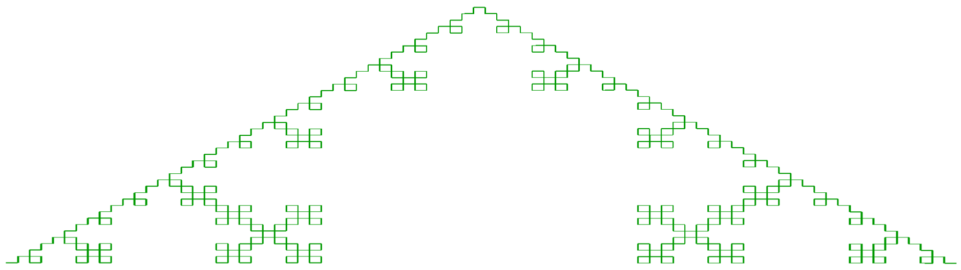


Figure 3.3: Koch Curve.

<u>Sierpinski Triangle:</u>
Alphabet: A, B
Constants: +, -
Axiom: A
Angle: 60°
Rules:
A → B-A-B
B → A+B+A

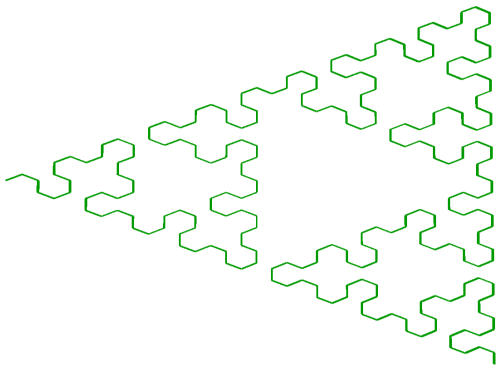


Figure 3.4: Sierpinski Triangle.

Dragon Curve:
Alphabet: F, X, Y
Constants: +, -
Axiom: FX
Angle: 90°
Rules:
X → X+YF+
Y → -FX-Y

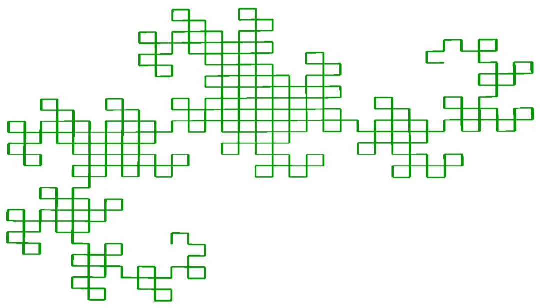


Figure 3.5: Dragon Curve.

Fractal Plant:
Alphabet: X, F
Constants: +, -, [,]
Axiom: X
Angle: 25°
Rules:
X → F-[[X]+X]+F[+FX]-X
F → FF

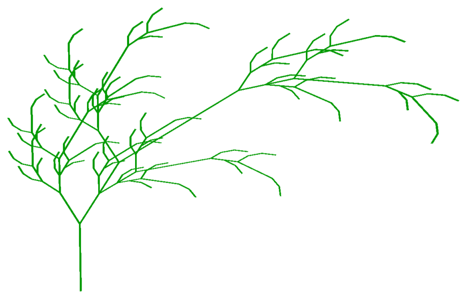


Figure 3.6: Fractal Plant.

Fractal Bush:
Alphabet: F
Constants: +, -, [,]
Axiom: F
Angle: 25°
Rules:
F → FF+[+F-F-F]-[-F+F+F]

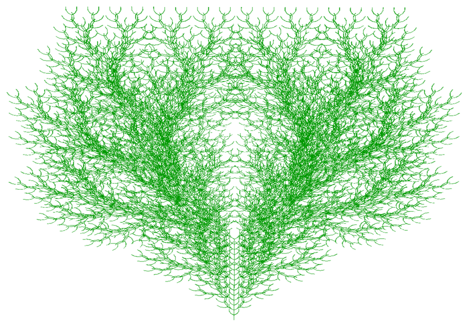


Figure 3.7: Fractal Bush.

3.5 The Use of L-systems in 3D applications

L-systems have been talked about and researched since its inception in 1968 by Aristid Lindenmayer. Over the years it's usefulness in modelling different types of plant life has been very clear, however its presence has been quite absent from any mainstream game engines for the most part, these engines relying either on digital artists skill to develop individual plants or on 3rd party software such as SpeedTree. These types of software use a multitude of different techniques however their methods are heavily rooted in Lindenmayer Systems.

3.6 Parametric L-system

With simplistic L-systems like the algae representation above, there are a number of details that are skipped over when making this simplistic representation. (talk about the representation for both parameterized and non parameterised Algae systems). When it comes to representing trees as L-systems a simplistic approach would be to just assume that the width and length of each branch section is constant and will not vary depending on where in the tree it is. We can also assume that the angles at which a branch may split is also constant, say 25 degrees. The resulting representation of this L-system is a tree like structure, however it is not a very accurate representation of a real tree in nature.
In order to more accurately model trees we need to take into account the branch width, height, branching angles. There are two different approaches to solving this added complexity. One would be to increase the complexity of the L-system grammar and the other would be to increase the complexity of the interpretation of the L-system.
For instance defining an complex L-system grammar with a less complex interpreting system can give a huge amount of flexibility to define parameters that can accurately define exactly how the L-system should be interpreted, and because the complexity is with the L-system rewriting you also have the control of being able to change the L-system rules.

3.6.1 Formalising Parametric L-system Grammar

```
<generations> ::= ”#n” ”=” <float> ”;”

<definition> ::= ”#define” <variable> <float> ”;”

<axiom> ::= ”#w” ”:” <moduleAx> ”;”
<moduleAx> ::= <variable> | ”+” | ”−” | ”/” | ”\” | ”^” | ”&” | ”!”
          <variable> ”(” <paramAx> <paramListAx> ”)”
          | ”+” ”(” <paramAx> <paramListAx> ”)”
          | ”−””(” <paramAx> <paramListAx> ”)”
          | ”/””(” <paramAx> <paramListAx> ”)”
          | ”\””(” <paramAx> <paramListAx> ”)”
          | ”^” ”(” <paramAx> <paramListAx> ”)”
          | ”&” ”(” <paramAx> <paramListAx> ”)”
<paramAxList> ::= ∈ | ”:” <paramAx> <paramAxList>
<paramAx> ::= <float>

<production> ::= ”#” <variable> ”:” <module> ”:” <condition> ”:” <successor> ”;”
<module> ::= <variable> | ”+” | ”−” | ”/” | ”\” | ”^” | ”&” | ”!”
          <variable> ”(” <param> <paramList> ”)”
          | ”+” ”(” <param> <paramList> ”)”
          | ”−””(” <param> <paramList> ”)”
          | ”/””(” <param> <paramList> ”)”
          | ”\””(” <param> <paramList> ”)”
          | ”^” ”(” <param> <paramList> ”)”
          | ”&” ”(” <param> <paramList> ”)”
<paramList> ::= ∈ | ”:” <param> <paramList>
```

```
<param> ::= <float>

<expression> ::= <expression> <symbol> <expression> |
<float> ::= [0-9]+.[0-9]+
<variable> ::= [a-zA-Z_][a-zA-Z0-9_]*
```

Chapter 4

Implementation

Introduction to the implementation section

4.1 Language and environment

To understand what programming language and environment will be best suited for this project, I first provide the technical requirements that will need to be met. The programming language will need to be relatively fast when processing the rules of the L-systems and when rewriting these strings according to those rules.

The program will also need to interpret the strings generated by the L-system rules and be able to generate a 3 dimensional representation of that L-system. This representation will need to be intuative and will have to allow me to examine it from different perspectives. In order to make the representation intuative, the 3D representation should be rendered in real time at multiple frames per second. And the user should be able to use a computer mouse and keyboard to move around the 3D world.

Due to these demands have decided to use the C and C++ programming language. Due to its and thoroughly tested in built standard template library, I can count on it to be reliable and fast enough for the purpose of this project. It will also allow me to use other very useful libraries for 3D graphics such as OpenGL which is also written in C/C++. I will be speaking in more detail about these details of these libraries in later sections.

In order to create a window and provide the environment for writing pixels to the screen I will make use of the Graphics Library Framework (GLFW). Some useful mathematics functions and facilities can be found in the OpenGL Matematics Library (GLM) and possibly the most important for rendering in 3D is the Open Graphics Library or OpenGL for short. All of these libraries together will provide me with a strong foundation of tools that I can use to approach the practical aspect of this project.

4.1.1 C/C++ Programming Language

The C programming language developed by Dennis Richie and Bell Labs in 1972 has been one of the most popular programming languages for a number of decades now [Ritchie et al., 1975]. The

C language was then extended upon by Bjarne Stroustrup in 1985 to create the C++ programming language [Stroustrup, 2000]. I have included both C and C++ as the programming languages that I will be using, as they are very closely related and C code can be compiled using the C++ compiler. For the most part I will be writing C++ code and making use of its object oriented features. However, their are instances when it will be more convenient to write C code or make use of a C library.

4.1.2 Standard Template Library (STL)

The STL in C/C++ provides a number of useful functions, data structures and algorithms that have been extensively tested for both reliability and efficiency. The most common features I will be using are strings, vectors, stacks and input and output. It is possible that in some cases it may be more efficient to use custom data structures for the most part the STL functions will be more than good enough. [Horton, 2015]

4.1.3 Open Graphics Library (OpenGL)

OpenGl is a 2D and 3D graphics API [Movania et al., 2017]

4.1.4 OpenGL Mathematics Library (GLM)

4.1.5 Graphics Library Framework(GLFW)

4.1.6 Git Version Control

Chapter 5

Glossary

Glossary

. 5

latex is something about this thing. 5

Appendix A

Appendix

A.1 Appendix 1

A.2 Bibliography

Bibliography

[Horton, 2015] Horton, I. (2015). *Using the C++ Standard Template Libraries.* Apress.

[Mandelbrot, 1982] Mandelbrot, B. B. (1982). *The fractal geometry of nature*, volume 2. WH freeman New York.

[Movania et al., 2017] Movania, M. M., Lo, W. C. Y., Wolff, D., and Lo, R. C. H. (2017). *OpenGL – Build high performance graphics.* Packt Publishing Ltd, 1 edition.

[Prusinkiewicz, 1986] Prusinkiewicz, P. (1986). Graphical applications of l-systems. In *Proceedings of graphics interface*, volume 86, pages 247–253.

[Prusinkiewicz and Hanan, 2013] Prusinkiewicz, P. and Hanan, J. (2013). *Lindenmayer systems, fractals, and plants*, volume 79. Springer Science & Business Media.

[Prusinkiewicz and Lindenmayer, 2012] Prusinkiewicz, P. and Lindenmayer, A. (2012). *The algorithmic beauty of plants.* Springer Science & Business Media.

[Ritchie et al., 1975] Ritchie, D. M., Kernighan, B. W., and Lesk, M. E. (1975). *The C programming language.* Bell Laboratories.

[Stroustrup, 2000] Stroustrup, B. (2000). *The C++ programming language.* Pearson Education India.

[Worth and Stepney, 2005] Worth, P. and Stepney, S. (2005). Growing music: musical interpretations of l-systems. In *Workshops on Applications of Evolutionary Computation*, pages 545–550. Springer.