Overview of Lindenmayer Systems

Miguel Palhas (Author)

Department of Informatics

University of Minho

Braga, Portugal

pg19808@alunos.uminho.pt

**Abstract**—FALTA METER AQUI CENAS (Abstract)

Keywords-component; computer graphics; l-system; geometry

# Introduction

# Origin

L-Systems were introduced in 1968 by a Hungarian biologist, Aristid Lindenmayer. Lindenmayer studied the patterns that can be observed in the growth of several types of plants. The original idea was to provide a formal method of describing the development of simple organisms. Later on, they started to be used to for more complex structure.

# Basic Structure and Properties

A Lindenmayer system (more commonly known simply as L-system) is a formal grammatical method to proceduraly model a system. Generically, it is represented as:

Where:

* ***V*** is the alphabet, a set of symbols defining the elements that compose the language of the system we are describing. These symbols can be concatenated into strings;
* ***w*** is the axiom, or the initial string upon which the system will operate to generate the final geometry;
* ***P***  is the set of production rules that define how the strings are replaced to form new combinations.

It’s also necessary to provide a depth level (*n*), to indicate how deeply the productions should be processed, to control the level of detail of the final geometry.

These are the components of a regular grammar, from which L-Systems are derived. Aside from these attributes, the system must also define a translation mechanism which maps each of the resulting strings into a geometric structure, in order to generate the final result.

The s ystem works by applying the rules iteratively to a string whose initial value is given by *w*. As opposed to a conventional formal grammar, which applies a single rule at a time, in this case many rules can be applied simultaneously in a single iteration.

If each production refers only to an individual symbol and to its neighbors, the system is considered to be context-free. This subset of L-Systems are specified by a regular grammar. If the rules depend also on the neighbor symbols, then it is a context-sensitive L-System.

An L-System can also be non-deterministic or deterministic. If only one rule can apply to each symbol, then it is deterministic, as the final result will not change as long as neither of the initial parameters also remain the same. However, there may be several rules applying to the same symbols, and a defined method of selecting between each one. If this method is probabilistic, then the system is a stochastic L-System.

# Example

The following system describe the original example shown by Lindenmayer, which models the growth of algae

This system produces the following result on the first iterations:

The recursive nature of the system can be seen in this example, as the resultant string large with every iteration.

# Rendering

After defining the grammar for the desired model, a method is required to generate a graphical image based on the symbols acquired from iterating through the grammar. A common method to perform this is with Turtle graphics.

Turtle graphics is a computer graphics technique used for vector graphics that works a state machine using a cursor (also called the “turtle”) in a Cartesian Plane. In this method, commands are given to alter the state of the turtle, which has simple properties like position, orientation, color, or drawing state. When the drawing state is true, the turtle acts like a pen, drawing as it moves. The turtle can receive commands such as ”move forward 5 units” or ”turn left by 30 degrees”, effectively altering the current state.

From these basic building blocks, more complex algorithms can be made. Consider the following function that instructs the turtle to draw a simple square:

draw\_box() {

forward(10);

turn(90);

forward(10);

turn(90);

forward(10);

turn(90);

forward(10);

turn(90);

}

This function can be used to draw even more complex shapes, such as a group of boxes:

draw\_boxes() {

draw\_box();

turn(90);

draw\_box();

turn(90);

draw\_box();

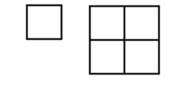
turn(90);

draw\_box();

turn(90);

}

These two functions give the following results:



Of course, the amount of instructions that can be issued to the turtle is implementation-dependent, and can be much more complex as just moving forward and rotating. Concepts like memory, stacks, probabilities and functions can be implemented to further increase the capabilities of the turtle.

# Pairing L-Systems with Turtle Graphics

In an L-System, strings are recursively expanded into other strings based on the set of production. If we interpret each Symbol of a string as a turtle command, then we can look at the whole string given by the L-System as a whole function that can be issued to render turtle geometry. A common notation for the symbols used is later explained in section 6.

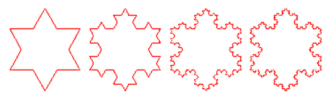
Let’s assume the following symbols, and their corresponding turtle command:

And also, the following L-System:

Without applying any productions (*n* = 0), *w* is unchanged, and according to the previously defined turtle commands, the rendered result will be:

C:\Users\Naps62\Desktop\monografia\images\1_triangle.png

With *n* = 1, the production is applied once, which means that all F symbols in *w* will be expanded. As *n* increases, so does the complexity of the result. Here are the results of this L-System for



# Adding Functionality

Let’s now add some more operators to the symbol list:

Where pushstate() is a function that instructs the turtle system to store its current state in a stack, and pop state() instructs it to reload the last saved state, removing it from the stack. With the introduction of a stack, the turtle can effectively recover previous states, allowing for branched geometries to be rendered, continued from a previous point of the geometry that is not the most recently drawn. For instance, consider now the following L-System and its corresponding result for different n values



The stack operator can be useful to allow the turtle to explicitly recover previous states, allowing complex geometries that resemble weeds and other types of vegetation.

## Parametric L-Systems

The productions of an L-System can support parameters which can be useful to use the same System to generate geometries with a certain degree of variety. For example, a generator for a tree leaf can be parameterized with the leaf’s color and length, resulting in the creation of several different leafs which, even though are sharing a common structure, have some noticeable differences between them.

Another useful capability of parameters is to allow the same productions to be used to achieve slightly different results, according to the parameters that are given to them. The previous example to render weeds can be extended with parameters to increase the complexity of the result.

## Stochastic L-Systems

A stochastic L-System is one that depends on probabilistical values, and thus allows for different results to be achieved with different executions.