Lindenmayer systems - a C++ implementation

Steffen Knoblauch
August 2, 2020

Lindenmayer Systems, short L-systems, are the result of **research from Lindenmayer et al.**[1] about the geometric features of plants. L-systems are a concept to mathematicaly/formal describe and model the growth processes of plant development. They are not only restricted to the plant based developments, but can also be used to generate fractals.

L-systems have an inital state and use rules, like a formal grammar, to transform or rather rewrite the current state to create the next state of the development from a plant or a fractal. It is therefore possible to successive calculate each state of the development. Such a state of a L-system can be interpreted as commands for a turtle graphic, which creates the opportunity to draw the created fractals or plant states.

Goal of this paper is to design an architecture for L-systems, which includes an implementation for L-systems, their creation and an interface for a turtle graphic. The interface should enable the polymorphic use of different turtle graphic implementations and enable drawing of the L-system state.

[1] ZITIEREN

1 Introduction

L-systems are a formal way to describe plant or fractal development and interpret the result as a graphic. In order provide a general understanding of Lsystems is this paper organised in several topics. Section 2 is a short introduction to the general idea, based around object rewriting, the grammar of L- systems and the interpretation of a L-system as graphic. After discussing the architecture and possible implementation steps, a final concept for an implementation is proposed.The code for this implementatiopn is available via my github repository.[2]

Finaly, there is a conclusion and an outlook for possible future extensions.

2 Lindenmayer systems

2.1 History

"[L-systems] were introduced in 1968 by Lindenmayer as a theoretical framwork for studying the development of simple multicellular organisms [...]"[3] and were later used in computer graphics to generate visuals of organisms and fractals.

On the beginning the focus of L-systems theory was based on larger plant parts and the graphical interpretation used chains of rectangles to display a L-system. Further research into L-system extended the interpretations, resulting in a interpretation of a L-system state with a LOGO-style Turtle. These extension make it possible to model more complex plants and fractaals and display them in a graphical way. [4]

2.2 General idea

The general idea of a L-system is the use of a rewriting system based on a formal grammar. The shape of a plant or a fractal consists of geometric pieces, for example a branch of a tree has several subbranches. "When each piece of a shape is geometrically similar to the whole, both the shape and the cascade that generate it are called self-similar."[5] The self-similartiy makes it possible to create a formal description for the plant or fractal generation as a formal grammar, further discussed in section 2.3. The rewritng uses this formal description to generate the diffferent states of the development. "In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or productions."[6].

For example this concept can be used to rewrite a inital string, called axiom, with defined rewriting rules. A simple example is the following grammar, which consits of only two nonterminals, A and B, and two production rules. The first rule is A→AB, the second rule is B→A. The arrow '→' symbols the replacement, the rewriting, of the object on the left with the object of the right of the arrow. The L-system has as axiom the value 'A' and will be expanded with these rules, creating the results in figure 1.

The first step is to use rule one, which replaces the nonterminal A with AB, resulting in the first generation. The result of the first generation ('AB') will be used to generate the second

[2]link to my github adding

[3]Zitiert ausd abop Preface - Abschnitt Modeling of Plants

[4]Zitat abop Vgl. Seite 6 Chapter 1.3

[5]Zitate The Nature of.... chapter 6 page 34

[6]Zitiert von abop Chapter 1 - 1.1 Rewriting systems

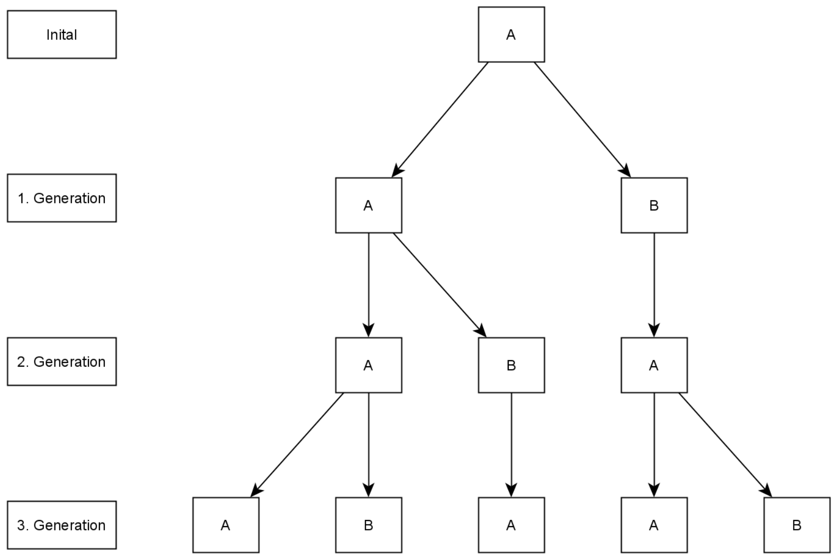

Figure 1: Simple L-system

generation. For all nonterminals the productions will be parallel applied. In this case the nonterminals A and B will be replaced, because both have existing production rules. This results in the second generation with 'ABA' as result.

This process can be successively repeated recursive for a arbitrary amount of generations and create a fractal or plant with self-similar pieces. The more generations are calculated, the more detailed is the resulting state.

2.3 Grammar

The definition of an L-system can be done similar to a Chomsky grammar, but there are some general differences. In Chomsky grammars the productions are applied sequentially, whereas in L-systems they can be applied parallel. This has some consequences for the formal properties of an L-system, for example a context-free L-system can produce a language which cannot be produced by a context-free Chomsky grammar.[7]
This paper will only focus on a class of L-systems, the DOL-systems, which can be used for string based rewriting systems. This class is deterministic (D) and context-free (O) and can be formaly described by a tuple:

$$G = (V, \omega, P)$$

V: set of symbols as alphabet of the l-system - consiting of terminals and non terminals
ω: axiom - nonempty word of the alphabet, which should contain at least one nonterminal
P: set of productions

A production conistst of a predecessor, a nonterminal symbol of the alphabet, and a successor, the replacment of the nonterminal in the next generation. To guarantee that the l-system is deterministic, there only can be one production for each nonterminal of the alphabet.The identity production is implicit a part of the set of productions.[8]

[7]Zitat abop vgl Seite 3 Abschnitt L-Systems
[8]Zitat vgl Seite 4 abop

If in the process of rewriting a terminal symbol is found, there will be no explicit production applied, but rather the identity production is applied. Terminals will therefore remain in future generations and won't extend an L-system with a production.

As mentioned in section 2.1 there are other extensions of a basic L-systems. The extensions introduce more possiblities for the generation process, like a non-deterministic behaviour. These extensions result in new grammars which can represent much complexer plants or fractals:

- Stochastic grammars: the system isn't deterministic anymore, because there can be multiple production rules for the same nonterminal with a probaility which results in a randomisation of the generation
- Context senstive grammars: production not only look for the nonterminal, they rather look for the symbols bevor and after the current symbol to process, the context.
- Parametric grammars: it is possible to set additional parameters for a symbol to influence the generation or the evaluation of the data

2.4 Interpretation as turtle commands

The L-systems introduced to this points are capable of creating a string based on a grammar. In order to create a graphic of a state of a L-system, the resulting state can be interpreted as commands of a turtle.
A turtle is a concept introduced by the language Logo as a tool for computer graphics.[9]
A state of a turtle can be described as a tuple[10]:

$$S = (x,y,\alpha)$$

x and y are Cartesian coordinates
α is the direction in which a turtle is facing, called heading

A turtle can move in the Cartesian coordinate system by altering the current state.You can think of it as a real turtle with a pen attached to it, walking on a paper. While moving in the coordinate system, or on the paper, the turtle can draw lines. Given this concept the turtle can receive different commands. The commands let the turtle walk on the paper or the coordinate system by altering the current state and drawing a line. For now I'll restrict this to a two dimensional coordinate system, but it is also possible to enhance it for a three dimensional coordinate system.

The walking path of the turtle is controlled by commands. Therefor a defined step size d and an angle θ is needed to calculate the next state of the turtle.

- Draw: moves one step in the current facing direction drawing a line
- Move: moves one step in the current facing direction without drawing a line
- Right-turn: turns to the right by the angle θ
- Left-turn: turns to the left by the angle θ

[9] Zitat VGL. Seite 179 Chappter 10 Logo book
[10] Zitat vgl abop Seite 6 ff

[11]

Additional to the state of the turtle the state there is a state for the pen. The pen state consist of the color and its width, which will result in more colorfull or different pictures.

With the given turtle concept is it possible to interpret the result of an L-system. Therefor a mapping between the symbols in the alphabet and the commands which should be called is needed. This could be done by an arbitrary mapping between a nonterminal or a terminal and a command. For this paper the following mapping will be used, but the concept for the architecture includes the possibility to use other mappings in the future.
For the mapping alphabet V is used, which is a slightly extended version of the basic version of a L-system:

$$V = (F, f, +, -, [,])$$

This alphabet will be mapped with extended turtle commands:

Symbol	Turtle interpretation
F	Draw a line in the facing direction
f	Move in the facing direction
+	Turn right
-	Turn left
[	save the current state in a stack
]	pop the last state form the stack and set it to the current state

This interpretation enables to draw the result of the L-system by iterating over every terminal and nonterminal of the result state and calling the mapped command. If no command is mapped the symbol will be skipped.

STACK ENTFERENEN UND NUR SIMPLEN WEG NUTZEN

2.5 Examples

This section will present some examples for L-system grammars which create fractals. They use the introduced mapping of section 2.4.

2.5.1 Koch curve

This fractal will be generated with the simple axiom 'F' and just one production: F→F+F–F+F
The turtle will be initalized with an angle of 60° and an arbitrary length *d* for a step.
[12]

2.5.2 Sierpinski triangle

This fractal will be created with 'F' as axiom and two productions:

- X→YF+XF+Y

[11] Zitat vlg cad book seitev 2 unteres drittel
[12] ADD A PICTURE and perhaps a source: http://mathforum.org/advanced/robertd/lsys2d.html

- Y→XF-YF-X

The turtle will be initalized with an angle of 60° and an arbitrary length *d* for a step.
[13]

[13] ADD A PICTURE and SOURCE

3 Architecture

The primary goal of this paper is creating an architecture and an implementation of a L-system as described in section 2. The focus of the architecture will be on flexibility and expandability and therefor is the following section splitted into several parts with discussions about different aspects of the final architecture.
[14]

3.1 Requirements overview

The architecture for a L-system, as introduced in section 2, has several requirements . This section will introduce the needed core requirements to get a flexible L-system and turtle.

The core of the concept is the L-system itfself, which has to guarantee a flexible usage. The L-system holds relevant data like the grammar, consisting of production rules and an axiom. The L-system should offer a way to configure a grammmar and guarantee the usage in different architectures without hardcoded grammars. The L-system should also offer to successively generate the next states of the configured grammar and access this generated result.

The turtle is another key component, which will be used to interpret the result of a L-system. The turtle has to offer the typical turtle commands as introduced in section 2.4. In order to provide a turtle that is as extensible as possible, it should be possible to implement it flexibly. Therefor an interface should be offered, that enables a fast exchange of implementations. Depending on the implementation of a turtle, it should be possible to configure the turtle and change properties, like the color or line width.

A mapping between the turtle commands and the L-system alphabet is an important part, so it is possible to call the correct turtle commands. Additional a function which is calling the mapped commands, after interpreting the L-system result, is needed to create an image of a plant or a fractal development state.

3.2 L-system

This section provides a dissussion of possible implementations of the L-system and a final proposal.

A simple and rather naive idea would be an object which holds the L-system as a string and addtional the grammar, consisting of productions and an axiom. This L-system could offer a function that calculates the next generation by iterating over every char in the string. If there is a production for the current char, the char will be replaced accodirng to the production with the predecessor. This function could be called for n generations to create the n-th generation of the L-system.[15]

This idea has several design flaws resulting in a bad performance and unclear behaviour. The first flaws exist because of the lack of seperation between datastructure and processing of the data. It is not possible to simply exchange the datastructure or the function without the seperation, which affects flexibility of the L-system. This can be solved by extracting the functionality into a seperate function and the data structure remains as "dumb" object. The seperate function can be a used to manipulate the current state in the datastrutcture itself and also creates

[14] HIER SOLLTE MAN EVT NOCH IRGENDWO DIE SEMANTSCIHE SCHNITTSTELLE ERWÄHNEN
[15] BEISPIEL ALS UML?

the opportunity to exchange the underlying datastructure, as long as the datastructure fullfills some formal properties, like the support of access to the string and grammar.
A further flaw is the bad performance because of the use of a string to save the current state/-generation. There are to reasons why the performance is bad, especially for larger generations. The first reason is the size of the string itself, which will grow very fast because of the nature of a L-system. Every nonterminal will be often replaced by severall symbols, which enormously increases the total size of the string for each generation. Additional to the large amount of space needed for the string, the replace in a string needs to be done for each nonterminal, which can result in a big overhead. Because of this problems, another more efficient design is needed.

In order to solve this problems and to create a more efficient and flexible architecture, let's recap the nature of L-systems. The generation of a L-system is based on a rewriting process. This process iterates over every object in the current state and calcualtes the resulting geneartion with the productions. Because of the restriction to DOL-systems the rewriting is not context-sensitive and can be done independent from other objects of the current state. If a object gets replaced by the successor of the production, the result of this replacment can be handled independent. The calculation of the n-th generation can be done for each object on their own and can simply be done in a recursive manner.
There is only a limited number of productions in a grammar of a L-system. Each of these productions is choosen deterministic when comparing the nonterminal. The next generation is created by rewriting the curren state, when a nonterminal is often in the current state, the same production can be used. If the same production is used muliple times, the resulting generation has a lot of similar objects or more specific the same string muliple times. These two aspects can be used to improve the current discussed idea.

-> Nächste ansätze -> Unterschiedung zwischen den verschiednen arten das zu speichern -> nature of the L-system thorey nochmal in kopf rufen

To make it possible to exchange the L-system a clear seperation between the datastructure and the functionality of a L-system is needed.

The L-system is just a container for the data and won't provide

Core der architektur

Daten halten -> zwei möglichkeiten: nur die grammatik und produktionsregeln oder eine generation

wenn zustand -> einfachste möglichkeit ist einfach einen String zu speichern und für jede generation string replacmenets machen -> sehr teuer –> Folglich muss man einen graph aufbauen, der die Daten hält damit es effizienter wird wenn on the fly dann nur grammatik notwendig -> Time memory tradeoff -> man muss jedes mal neu berechnen

-> berechnen von nächster generation notwendig -> beim graph muss man immer wieder neue Knoten ziehen und Daten anders halten -> Beim on the fly: rekursive mit direktem übergeben des wertes

-> berechnen von mehrern generationen auf einmal schwierig -> beim zustand müsste man alle zustände entsprechend markieren -> beim on the fly kann man das auch nur erschwert umsetzten

-> wie funktioniert eigentlich das aufrufen

Core of the current architecture but still excahngable

What data to hold How to generate different generations (Suksessiver aufbau des L Systems) - >why dont hold the generations - efficency On the fly generation with an Output itertaor vs Graph

Nutzung von beliebiger datenstruktur mit speziellen eigenschaften -> Semantische Schnittstelle

3.3 Turtle

Abstract interface with minimal set of needed functions

3.4 Turtle command mapping

3.5 Configuration data

convinient way of define a grammar - not hard coded - load form file

4 LSystem Datastructure

- Tree like sturcture
- save data not double only save pointers to the data
- provides access to the data with an iterator

5 Parser for the lsystem

- Parses the result of the l system
- calls the Turtle Graphic on the fly
- Problem for now -> not very flexible (perhaps for the future: provide which function to call for which object)

6 Implementation

6.1 Build System

The first important point

- Cmake as buildsystem
- reasons why cmake
- problems ?

6.2 TurtleGraphic

6.2.1 TestTurtle

6.2.2 CairoTurtle

6.2.3 Further implementations

SVG implementation

7 Tests

8 Examples

9 Problems and Restrictions

10 Outlook

16

[16] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. 2004. [Online]. Available: `http://algorithmicbotany.org/papers/abop/abop.pdf` (visited on 07/16/2020)