Lindenmayer systems - a C++ implementation

Steffen Knoblauch
August 1, 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

This paper is organised in several topics, beginning with section 2, which is a short introduction to the general idea, based around object rewriting, and the grammar of L- systems.

The following sections contain a discussion about the architecture and possible implementation steps. The discussion results in a final concept for an implementation. The final concept is realized and the code is available via my github repository.[2]

The end of the paper sums up with a conclusion section and an outlook for possible extensions in the future.

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

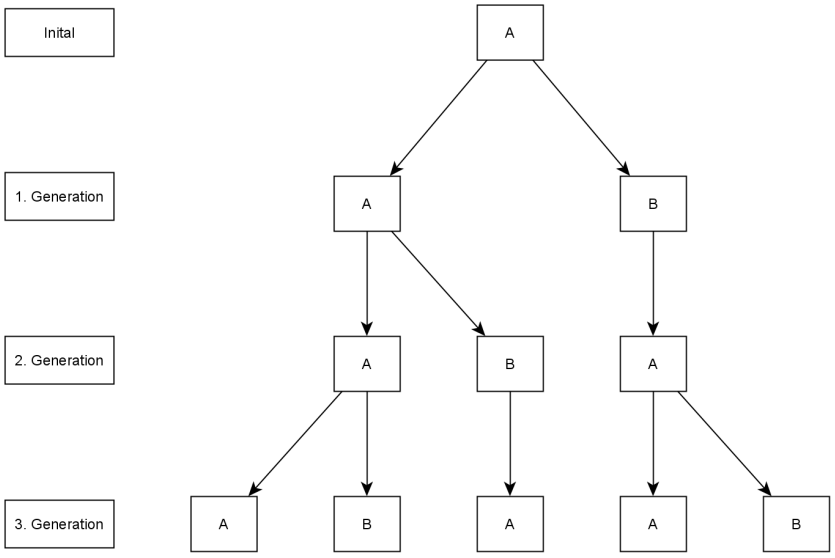

Figure 1: Simple L-system

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.

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 coniststs 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:

- 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 evalutaion of the data

With these new grammars it is possible to generate more complexe fractals and more detailed plant structures.

2.4 Interpretation as turtle graphic

The L-systems introduced to this points are capable of creating a string based on the grammar. To create a graphic of the generation of the L-system, the resulting string can be interpreted as commands of a turtle graphic.

WHAT IS A TURTLE

This is based on a mapping between the symbols in the alphabet and the commands which should be called. 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 the following alphabet is used, which is a slightly extended version of the basic version of a L-system:

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

Symbol	Turtle interpretation
F	cell
f	cell
+	cell
-	cell
[	cell
]	cell

2.5 Examples

- Koch
- ...

3 Architecture

- general focus on flexibility
- interfaces for future use cases
- different independent parts in the sample application

4 Build System

- Cmake as buildsystem
- reasons why cmake
- problems ?

5 FileHandler

- Load init data from file
- convinient way to configure the sample code
- no hard coded l system rules and axioms - use of the data in other apllications

6 LSystemHandler

- Suksessiver aufbau des L Systems
- Nutzung von beliebiger datenstruktur mit speziellen eigenschaften -> Semantische Schnittstelle
- Bekommt die Daten aus dem FileHandler, kann aber aus allem kommen - belieib in andere Sachen einbindbar

7 LSystem Datastructure

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

8 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)

9 TurtleGraphic

9.1 Abstract class

9.2 TestTurtle

9.3 CairoTurtle

9.4 Further implementations

SVG implementation

10 Tests

11 Problems and Restrictions

12 Outlook

[9]

[9] 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)