Lindenmayer systems - a flexible implementation

Steffen Knoblauch
July 31, 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 describe and model the growth processes of plant development. They are not only restricted to the plant based developments, but they can also be used to generate fractals.

L-systems start with a defined state and use rules, like a formal grammar, to transform the current state to create the next state of the development or the fractal. It is therefore possible to successive calculate each state of the development of a plant or a fractal. 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.

[1] ZITIEREN

1 Introduction

This paper is organised in several topics, beginning with section 2, which is a short introduction to the general idea and the grammar of L- systems. The following sections contain a discussion about the architecture and the final general concept of an implementation of a L-system.

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 [...]"[2] 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 was later on extended with new interpretations, which enabled it to represent complexer and more detailed objects.

2.2 General idea

The general idea of a L-system is the use of a rewriting system based on a formal grammar. The formal grammar is restricted by and based on different principles, which will be discussed in section 2.3.

"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."[3]

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 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.

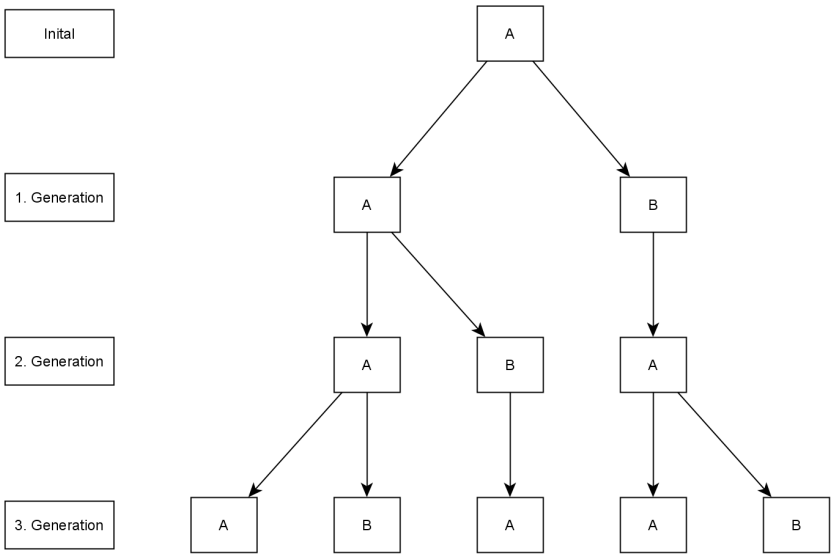

Figure 1: Simple L-system

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]Zitiert ausd abop Preface - Abschnitt Modeling of Plants
[3]Zitiert von abop Chapter 1 - 1.1 Rewriting systems

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.

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.[4]
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.[5]

TODO: Erklären was self similar und development rules sind

Further a terminal, a letter which has no production rule and won't be replaced, will be skipped and only the nonterminals will be rewriten. This can be done because the based on self-similarity and development-algorithms.

2.4 Focus

Fokus auf DOL-Systems -> deterministic Not 3D -> but possible becuse of open implementation

Nicht auf Stochastic L-systems Nicht context sensitve L-Systeme Es gib tweitere arten zb auch paremtererisiert

2.5 Examples

- Koch
- ...

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

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

6

[6]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)