

# Prozedurale Generierung von Baumstrukturen innerhalb der Unreal Engine 4

Procedural generation of tree-like structures in the Unreal Engine 4

David Liebemann

Bachelor-Abschlussarbeit

Betreuer:

Prof. Dr. Christof Rezk-Salama  
Prof. Dr. Georg Rock

Trier, 28.02.2017

---

## Kurzfassung

In der folgenden Abschlussarbeit werden zwei verschiedene Verfahren zur prozeduralen Generierung von Baumstrukturen und die Implementierungen dieser innerhalb des Frameworks der Unreal Engine 4 vorgestellt. Basierend auf der Untersuchung bisheriger Arbeiten werden zwei Ansätze gewählt: Lindenmayer-Systeme und ein Space Colonization Algorithmus.

Ein Lindenmayer-System – kurz: L-System – ist eine Erweiterung von kontextfreien Grammatiken, welche Teile einer übergebenen Zeichenkette anhand festgelegter Regeln durch andere Zeichenketten ersetzen. [PL90, S.2] Die grundlegende Funktionsweise und die für eine Generierung von Baumstrukturen benötigten Erweiterungen werden behandelt sowie eine Methode zur Visualisierung der Ergebnisse von L-Systemen vorgestellt.

Das Prinzip des verwendeten Space Colonization Algorithmus basiert auf einer biologisch motivierten Simulation der Konkurrenz von wachsenden Zweigen um Wachstumsraum. [RLP07, S.2f] Benötigte Eingaben, der Ablauf des Algorithmus und die Prozedur zur Generierung von Baumstrukturen werden erläutert sowie Erweiterungen des ursprünglichen Algorithmus vorgestellt.

Die im Rahmen dieser Arbeit umgesetzten Implementierungen beider Verfahren werden behandelt. Die verwendete Datenstruktur in Form eines graphentheoretischen Baumes und das darauf basierende Modellgenerierungssystem, welches für die Konstruktion der Modelldaten verantwortlich ist, werden vorgestellt. Die aus den Implementierungen resultierenden Baumstrukturen werden präsentiert und der Einfluss von Parametern auf das visuelle Erscheinungsbild sowie die Effizienz der Generierung der Modelle wird behandelt.

Abschließend findet eine Bewertung und ein Vergleich beider Verfahren auf Grundlage der vorgestellten Ergebnisse sowie die Behandlung wünschenswerter Erweiterungen für zukünftige Arbeiten statt.

In the following bachelor thesis we present two different methods for the procedural generation of tree-like structures and their implementations in the framework of the Unreal Engine 4. Based on the analysis of past works we chose two approaches: Lindenmayer-Systems and a Space Colonization Algorithm.

A Lindenmayer-System – in short: L-System – is an extension of context free grammars which replace parts of a given string by other strings, according to a pre-

defined ruleset. [PL90, S.2] We examine the basic functionality and the extensions required for generating tree-like structures and present the chosen visualization method for displaying the results.

The functionality of the Space Colonization Algorithm is biologically motivated by the competition for space between growing branches. We introduce the necessary input for the algorithm, the algorithmic process, the procedure for generating tree-like structures and extensions of the original algorithm.

We present the implementations of both techniques, the graph-theoretical tree data structure used by us and the system responsible for constructing the 3D model data. Afterwards we display the resulting tree structures and discuss the impact of parameter values on the visual appearance and efficiency of the generation methods.

Finally we assess and compare both techniques based on the presented results and discuss desirable extensions reserved for future works.

---

# Inhaltsverzeichnis

|  |    |
|--|----|
| <b>1 Einleitung</b> .....  | 1  |
| 1.1 Prozedurale Generierung von Baumstrukturen .....                               | 1  |
| 1.2 Bisherige Arbeiten .....   | 2  |
| 1.3 Ansatz .....   | 3  |
| 1.4 Unreal Engine 4 .....  | 3  |
| <b>2 Lindenmayer-Systeme</b> .....   | 6  |
| 2.1 Kontextfreie Grammatik .....   | 6  |
| 2.2 D0L-Systeme .....  | 7  |
| 2.2.1 Parametrische L-Systeme .....  | 8  |
| 2.3 Grafische Interpretation von L-Systemen .....                                  | 10 |
| 2.3.1 Turtle-Interpretation .....  | 10 |
| 2.3.2 Verzweigte L-Systeme .....   | 11 |
| 2.3.3 Erweiterung der Turtle-Interpretation in den<br>dreidimensionalen Raum ..... | 13 |
| 2.4 Anpassungen für Baumstrukturen .....   | 15 |
| 2.4.1 Graphentheoretische Bäume .....  | 15 |
| 2.4.2 Tropismus .....  | 17 |
| <b>3 Space Colonization Algorithmus</b> .....                                      | 19 |
| 3.1 Ursprung .....   | 19 |
| 3.2 Aufbau .....   | 19 |
| 3.3 Ablauf .....   | 20 |
| 3.4 Generierung von Baumstrukturen .....   | 22 |
| 3.5 Erweiterungen .....  | 23 |
| <b>4 Implementierung</b> .....   | 26 |
| 4.1 Baumrepräsentation .....   | 26 |
| 4.2 L-Systeme .....  | 27 |
| 4.2.1 Parameter .....  | 27 |
| 4.2.2 Ableitung .....  | 28 |
| 4.2.3 Turtle Interpretation .....  | 28 |
| 4.3 Space Colonization Algorithmus .....   | 28 |

|  |    |
|--|----|
| Inhaltsverzeichnis                               | v  |
| 4.3.1 Einflussbereiche .....                     | 29 |
| 4.3.2 Parameter .....                            | 29 |
| 4.3.3 Ablauf des Algorithmus .....               | 29 |
| 4.4 Modellgenerierung .....                      | 29 |
| 4.4.1 Procedural Mesh Component .....            | 30 |
| 4.4.2 Parameter .....                            | 30 |
| 4.4.3 Operationen auf dem Baum .....             | 30 |
| 4.4.4 Generierung der Zylinder-Meshes .....      | 31 |
| 5 Ergebnisse .....                               | 35 |
| 5.1 L-System-Actor .....                         | 35 |
| 5.1.1 Monopodiales Wachstum .....                | 35 |
| 5.1.2 Sympodiales Wachstum .....                 | 36 |
| 5.1.3 Ternäre Verzweigungen .....                | 37 |
| 5.1.4 Tropismus .....                            | 38 |
| 5.2 Space-Colonization-Actor .....               | 39 |
| 5.2.1 Einflussbereiche .....                     | 39 |
| 5.2.2 Wachstumsparameter .....                   | 40 |
| 5.3 Effizienz .....                              | 46 |
| 5.3.1 Generierungszeit .....                     | 46 |
| 5.3.2 Laufzeitverhalten .....                    | 47 |
| 6 Zusammenfassung und Ausblick .....             | 49 |
| 6.1 Bewertung und Vergleich der Ergebnisse ..... | 50 |
| 6.1.1 Visuell .....                              | 50 |
| 6.1.2 Effizienz .....                            | 51 |
| 6.1.3 Benutzerfreundlichkeit .....               | 52 |
| 6.2 Wünschenswerte Erweiterungen .....           | 52 |
| 6.2.1 L-Systeme .....                            | 52 |
| 6.2.2 Space Colonization Algorithmus .....       | 53 |
| 6.2.3 Allgemeine Erweiterungen .....             | 54 |
| Literaturverzeichnis .....                       | 56 |
| Projektaufbau und Beispielanwendung .....        | 58 |
| A.1 Projektaufbau .....                          | 58 |
| A.1.1 Content .....                              | 58 |
| A.1.2 C++ Classes .....                          | 59 |
| A.2 Beispielanwendung .....                      | 59 |
| A.2.1 Aufbau .....                               | 59 |
| A.2.2 Steuerung .....                            | 60 |
| Erklärung der Kandidatin / des Kandidaten .....  | 61 |

# 1

---

## Einleitung

Die prozedurale Generierung von 3D-Modellen stellt einen wichtigen Bereich der Computergrafik dar. Im folgenden Kapitel werden bisherige Arbeiten zur Generierung von Baumstrukturen, die in dieser Arbeit umgesetzten Verfahren sowie das zur Implementierung verwendete Framework vorgestellt.

### 1.1 Prozedurale Generierung von Baumstrukturen

Die Gestaltung von 3D-Modellen für die Computergrafik erfordert den sicheren Umgang mit 3D-Modellierungssoftware, künstlerisches Geschick und einen hohen Zeitaufwand. Im Bereich der Filmindustrie werden 3D-Modelle eingesetzt, um reale Schauspieler mit virtuellen Umgebungen verschmelzen zu lassen oder Filme allein durch Computersimulation zu erstellen. [DL05, S.5] In Computerspielen werden Modelle verwendet, um beispielsweise virtuelle Charaktere und die sie umgebenden Landschaften darzustellen. Um diese Aufgaben zu erfüllen besteht ein Großteil der Entwicklungsteams aus 3D-Künstlern, welche für die Erstellung der 3D-Modelle verantwortlich sind. [STN16, S.3]

Die Prozedurale Generierung von Modellen ist ein Ansatz für eine schnellere Erfüllung dieser Aufgaben bei geringerem Aufwand und äquivalenter Qualität. Dabei werden 3D-Modelle durch computergenerierte Daten, auf Basis von Algorithmen und mit eingeschränktem Eingriff durch einen Benutzer, erstellt. [STN16, S.1]

Die Generierung von Pflanzenmodellen stellt einen großen Bereich der prozeduralen Modellgenerierung dar, da es nur in Ausnahmefällen möglich ist Szenen von Außenbereichen zu erzeugen ohne Vegetation darzustellen. Die Komplexität dieser Thematik wird bereits bei der Betrachtung einzelner Pflanzen deutlich und erstreckt sich von der Modellierung eines einzelnen Blattes über die Positionierung der Blätter auf einer Pflanze bis hin zur Verteilung dieser Vegetation in einer Landschaft. [DL05, S.3] Eine simple Wiederverwendung von Modellen wird schnell bemerkt und führt zu einem unnatürlichen Eindruck bei Beobachtern – die manuelle Gestaltung von 3D-Modellen ist somit selten praktikabel. [STN16, S.73]

Bäume stellen auffällige Landschaftsmerkmale dar und sind somit ein wichtiger Bestandteil einer realistischen Umgebungsgestaltung. Im Rahmen dieser Arbeit werden daher zwei unterschiedliche Verfahren für die prozedurale Generierung der



Abb. 1.1: „Green“ von Jan Walter Schliep, modelliert mithilfe der prozeduralen 3D-Modellierungssoftware für Vegetation, „Xfrog“. [Gre]

Aststrukturen von Bäumen sowie die jeweiligen Implementierungen in einer – insbesondere für die Verwendung in digitalen Spielen geeigneten – Echtzeitanwendung vorgestellt.

## 1.2 Bisherige Arbeiten

Die Entwicklung von Programmen für die 3D-Modellgenerierung von Baumstrukturen begann mit den Arbeiten von Honda und Fisher sowie den Erweiterungen dieser durch Aono und Kunii. Sie verwendeten einige simple Regeln, die mithilfe von Parametern angepasst werden konnten, um realistisch wirkende Baumstrukturen zu erschaffen. [DL05, S.46f] [RLP07, S.1]

Die bereits zuvor von Aristid Lindenmayer entwickelte Erweiterung von kontextfreien Grammatiken – Lindenmayer-Systeme – für die Beschreibung zellulärer Vorgänge und das Verzweigungsverhalten von Pflanzen wurden später von Prusinkiewicz und Lindenmayer zusätzlich erweitert, um die Generierung ähnlicher Baumstrukturen mithilfe dieser Systeme zu ermöglichen. [DL05, S.43, 48]

Die von P. Oppenheimer und J. Bloomenthal entwickelten Verfahren basieren auf der rekursiven Generierung von Baumstrukturen. Oppenheimers Vorgehen war inspiriert durch die fraktale Natur von Pflanzen und konzentriert auf die schnelle grafische Repräsentation der Modelle, während Bloomenthal die natürliche Darstellung von Verzweigungen beabsichtigte. [DL05, S.49-52]

Weiterhin trugen Reeves und Blau, Weber und Penn, Lintermann und Deussen sowie Prusinkiewicz u.a zur Verbesserung und Erweiterung von rekursiven Prozeduren zur Generierung von realistischen Baumstrukturen bei. [RLP07, S.1]

Ein Ansatz, der sich vom rekursiven Vorgehen unterscheidet, wurde von Rodkaew u.a. entwickelt und basiert auf der Verteilung von Partikeln in Form einer Baumkrone um daraufhin die Aststruktur iterativ von außen bis zur Wurzel aufzubauen. [RLP07, S.2] Diesem Ansatz ähnelt das Vorgehen von Runions u.a., welches die Erweiterung eines zweidimensionalen Verfahrens zur Generierung von Blattvennen in den dreidimensionalen Raum darstellt und als Space Colonization Algorithmus (engl. für Raum-Kolonisierungs-Algorithmus) bezeichnet wird. Im Gegensatz zu Rodkaew u.a. wurde jedoch der iterative Aufbau ausgehend von der Wurzel und in Abhängigkeit des verfügbaren Raums zum Wachsen vorgeschlagen. [RLP07, S.2]

### 1.3 Ansatz

Im Rahmen dieser Arbeit wurden zwei verschiedene Verfahren implementiert: Die von Aristid Lindenmayer entwickelten Lindenmayer-Systeme sowie der von Runions u.a. entwickelte Space Colonization Algorithmus.

Lindenmayer-Systeme arbeiten mithilfe einer vorgegebenen Menge von Regeln, um bestimmte Teile einer Zeichenkette durch andere Zeichen zu ersetzen. Diese Regeln verlängern die ursprüngliche Zeichenkette und ermöglichen somit die Generierung von komplexen Zeichenabfolgen. [PL90, S.2] Um aus den Resultaten Modelldaten zu extrahieren wird die von Prusinkiewicz und Lindenmayer vorgeschlagene Interpretation von Zeichenketten verwendet. [PL90, S.6]

Der Space Colonization Algorithmus nutzt die biologisch motivierte Simulation von Konkurrenz um Platz zwischen sich entwickelnden Zweigen. Es wird ein bestimmter Wachstumsbereich vorgegeben, in dem Zweige in jeder Iteration des Algorithmus wachsen. [RLP07, S.5]

Die Wahl dieser Verfahren basiert auf den sich ergänzenden Funktionsweisen: Lindenmayer-Systeme erlauben eine genaue Kontrolle über die generierten Baumstrukturen mithilfe der Definition fester Regeln. Der Space Colonization Algorithmus hingegen ermöglicht, durch die Wahl einiger numerischer Parameter, die Generierung komplexer, auf räumliche Einschränkungen reagierender Baummodelle. [RLP07, S.5]

Die Verfahren enthalten keine festen Vorgaben in Hinsicht auf die grafische Darstellung der generierten Daten. Um eine Konzentration auf die Implementierung der Ansätze zu ermöglichen, wurde die Unreal Engine 4 als Framework für die visuelle und logische Repräsentation der Baumstrukturen in Echtzeit gewählt. Äste werden vereinfacht als Zylinder dargestellt.

### 1.4 Unreal Engine 4

Die Unreal Engine 4 ist eine Sammlung von Softwarewerkzeugen für die Entwicklung von digitalen Spielen, Echtzeit-3D-Filmen und Simulationen. [Wha] Sie bietet Bibliotheken für die Darstellung von 3D-Modellen und erfüllt unter anderem die grundlegenden Aufgaben der Speicherverwaltung, Serialisierung von Objekten, und Verwaltung von Benutzereingaben. [Eng]

Die Engine ist in *C++* programmiert und der Quellcode ist frei zugänglich. Eine Erstellung von Inhalten ist durch die Programmierung von *C++* Code oder durch die Verwendung der visuellen Skriptsprache „Blueprint“ möglich. [Wha]

Die Implementierungen der gewählten Ansätze wurden in *C++* realisiert. Während die Kapitel 2 und 3, welche die theoretischen Konzepte der Ansätze behandeln, unabhängig von dem Framework verfasst wurden, verwenden die darauf folgenden Kapitel Unreal Engine 4 spezifische Konzepte. Beispielsweise wurden beide Verfahren auf Grundlage der Actor-Basisklasse implementiert.

Ein Actor (engl. für Akteur) in der Unreal Engine 4 ist ein Objekt, das in einem Level platziert werden kann und eine bestimmte Position, Rotation und Skalierung in der Welt besitzt. Die generierte Baumstruktur wird einem Actor als Komponente hinzugefügt und kann daraufhin vom Grafiksystem der Engine dargestellt werden. Die Generierung der Modelldaten findet in dem lokalen Koordinatensystem des Actors statt – wird die Position, Rotation oder Skalierung des Actors verändert, überträgt sich diese Transformation ebenfalls auf die generierten Modelldaten. Dies ermöglicht eine einfache Positionierung von Baumstrukturen in einem Level, ohne neue Modelldaten generieren zu müssen. [Unrb]

Das Actor-Framework lässt Transformationen sowie die Eingabe von Parametern über den visuellen Leveleditor der Engine zu und erlaubt somit eine schnelle Änderung von Parameterwerten und Anpassung der Positionen von generierten Baumstrukturen. [Unrb]

Die in den folgenden Kapiteln verwendete Längeneinheit entspricht der in Unreal Engine 4 verwendeten Einheit „cm“, das Winkelmaß entspricht der Einheit „Grad“. Alle Abbildungen von Baumstrukturen wurden mithilfe von Screenshots des, im Rahmen dieser Arbeit entwickelten, Projekts erstellt.

Der Aufbau des Projekts und der Zugriff auf die entwickelten Inhalte wird in Appendix A erläutert.

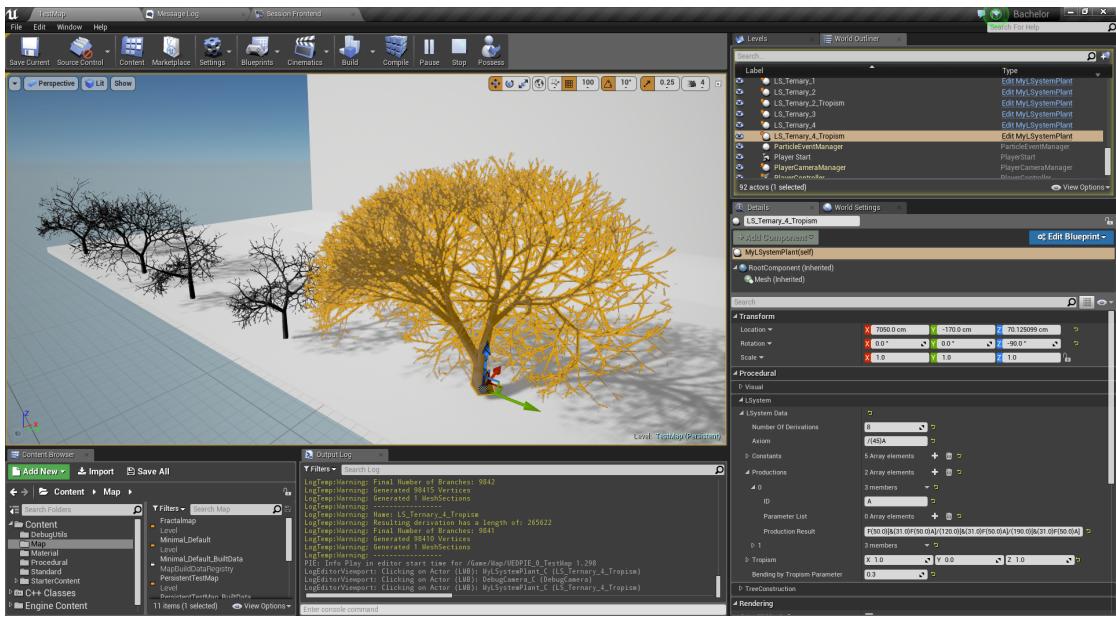


Abb. 1.2: Screenshot des visuellen Leveleditors der Unreal Engine 4. Der aktuell ausgewählte Actor und seine Komponenten sind gelb hervorgehoben. Die in der Bildmitte dargestellten Pfeile sowie die Eingabefelder im unteren rechten Bildbereich erlauben die Transformation des Actors. Ebenfalls im unteren rechten Bildbereich aufgelistet sind weitere Felder um die schnelle Änderung von Parameterwerten zu erlauben, die im C++ Quellcode entsprechend definiert sind.

## 2

---

# Lindenmayer-Systeme

In diesem Kapitel werden Regeln für die Definition von Lindenmayer-Systemen – kurz: L-Systemen – festgelegt und die verwendete Methode zur Visualisierung der Ergebnisse von L-Systemen besprochen. Es findet eine Beschränkung des Themas auf die in dieser Arbeit umgesetzten Konzepte statt.

## 2.1 Kontextfreie Grammatik

Bei L-Systemen handelt es sich um auf Zeichenketten basierende Ersetzungssysteme. Es werden komplexe Objekte beschrieben, indem Teile der Zeichenkette durch andere Zeichen oder Zeichenketten ersetzt werden. Die Beschreibung dieser Ersetzungen findet mittels festgelegter Produktionsregeln statt. [PL90, S.2]

Eine formale Definition eines auf Zeichenketten arbeitenden Ersetzungssystems wird durch kontextfreie Grammatiken gegeben:

**Kontextfreie Grammatik:** 2.1 Eine kontextfreie Grammatik  $G$  ist ein Tupel  $G = (V, N, P, \omega)$  bestehend aus:

- V** Einer nichtleeren, endlichen Menge von Buchstaben (Alphabet).
- N** Einer endlichen Menge von Variablen.
- P** Einer endlichen Menge von Produktionsregeln in der Form  $P : A \rightarrow \alpha$  mit  $A \in N$  und  $\alpha \in (V \cup N)^*$ .
- $\omega \in N$**  Dem Axiom, Startsymbol der Grammatik.

[Sch14, S.343]

Die Menge  $V^*$  ist die Menge aller Wörter über  $V$ , d.h. die Menge aller Wörter, die aus dem Alphabet  $V$  gebildet werden können. [Sch14, S.70]

Eine Grammatik wird als kontextfrei bezeichnet, wenn beispielsweise die Produktionsregel  $A \rightarrow \alpha$  angewendet werden kann, ohne die  $A$  umgebenden Buchstaben – seinen Kontext – beachten zu müssen. [Sch14, S.343]

Eine Grammatik wird als deterministisch bezeichnet, wenn es genau eine Produktionsregel  $r \in P$  für jede Variable  $A \in N$  gibt, sodass  $r : A \rightarrow \alpha, \alpha \in (V \cup N)^*$ . Das bedeutet, dass die Ersetzung einer Variable eindeutig durch eine einzige Regel beschrieben wird. [STN16, S.75]

Die Anwendung der Produktionsregeln findet meist sequentiell statt – die Zeichenkette wird von links nach rechts durchlaufen und Ersetzungen werden direkt auf die untersuchte Zeichenkette angewendet. [STN16, S.75]

## 2.2 D0L-Systeme

Diese Arbeit beschränkt sich auf die Behandlung deterministischer, kontextfreier L-Systeme, auch D0L-Systeme genannt. Diese besitzen die Eigenschaften einer deterministischen und kontextfreien Grammatik, Produktionsregeln werden jedoch parallel und gleichzeitig auf alle Buchstaben des untersuchten Wortes angewendet. Dieses Vorgehen soll die Zellteilung in mehrzelligen Organismen simulieren und ist somit an biologische Vorgänge angelehnt. [PL90, S. 3]

Ein D0L-System kann wie folgt definiert werden:

**D0L-System: 2.1** Ein D0L-System ist ein Tupel  $G = (V, P, \omega)$ , bestehend aus:

**V** Einem nichtleeren, endlichen Alphabet.

**P** Einer endlichen Menge von Produktionsregeln in der Form  $P : a \rightarrow b$  mit  $a \in V$  und  $b \in V^*$ .  $a$  wird als Vorgänger,  $b$  als Nachfolger bezeichnet. Ist für einen Buchstaben  $x \in V$  keine explizite Produktionsregel angegeben, wird die Identitätsproduktion  $P : x \rightarrow x$  angenommen – der Buchstabe wird durch sich selbst ersetzt.

**$\omega \in V^+$**  Dem Axiom, Startsymbol der Grammatik.

[PL90, S.4]

Die Menge  $V^+$  ist die Menge aller nichtleeren Wörter über  $V$ . [Sch14, S.70]

Die Ableitung eines Wortes entspricht der Ersetzung aller Buchstaben anhand der Produktionsregeln. Ein Wort kann mehrmals abgeleitet werden.

**Ableitung: 2.1** Gegeben sei ein Wort  $w = a_1 \dots a_m$  mit  $w \in V^*$  und  $a_i \in V$ . Das Wort  $v = b_1 \dots b_m$  mit  $v \in V^*$  und  $b_i \in V$  ist die Ableitung von  $w$  wenn für alle  $i = 1 \dots m$  eine Produktionsregel  $a_i \rightarrow b_i$  existiert. Die Ableitung wird als  $w \Rightarrow v$  notiert.

Das Wort  $w_n$  ist die  $n$ -te Ableitung des Wortes  $w_0$  wenn eine Folge von Wörtern  $w_0, w_1, \dots, w_n$  mit Ableitungen  $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$  existiert. [PL90, S.4]

Beispiel: Das Wachstum der Blaulalgen-Gattung „Anabaena“ kann durch ein L-System simuliert werden. Die Buchstaben  $a$  und  $b$  beschreiben die Größe und Teilungsbereitschaft einer Algenzelle, während die Indizes  $l$  und  $r$  die Polarität einer Zelle darstellen. Es gelten folgende Produktionsregeln:

$$\begin{aligned}
 p_1 : a_r &\rightarrow a_l b_r \\
 p_2 : a_l &\rightarrow b_l a_r \\
 p_3 : b_r &\rightarrow a_r \\
 p_4 : b_l &\rightarrow a_l
 \end{aligned} \tag{2.1}$$

Die Entwicklung einer Anfangszelle  $a_r$  (Axiom  $\omega : a_r$ ) läuft daraufhin wie in Abbildung 2.1 dargestellt ab. [PL90, S.4]

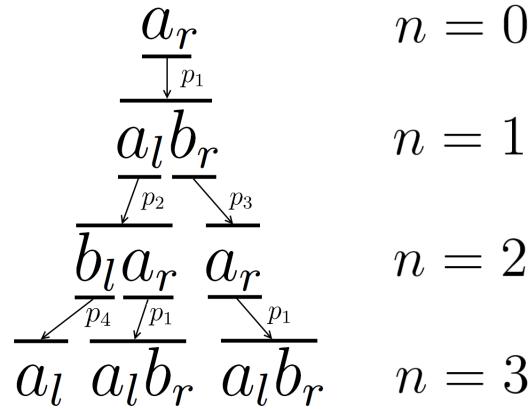


Abb. 2.1: Die n-fache Ableitung der Anfangszelle  $a_r$  anhand der Produktionsregeln  $p_1 \dots p_4$  aus Gleichung 2.1. Eigene Abbildung auf Grundlage von [PL90, S.4].

### 2.2.1 Parametrische L-Systeme

Parametrische L-Systeme stellen eine Erweiterung der D0L-Systeme dar. Die Buchstaben eines verwendeten Alphabets  $V$  werden um zugeordnete Parameter aus der Menge der reellen Zahlen ergänzt. Ein solches parametrisches Wort  $V \times \mathbb{R}^*$  besteht aus einem Zeichen  $A \in V$  und Parametern  $a_1, \dots, a_n \in \mathbb{R}$  und wird als  $A(a_1, \dots, a_n)$  dargestellt. Ein parametrisches Wort ohne Parameter mit dem Zeichen  $A \in V$  wird schlicht als  $A$  dargestellt. [PL90, S.41]

Die obige Definition von parametrischen Worten geschieht mithilfe von numerischen Konstanten, während bei der Angabe eines L-Systems formale Parameter verwendet werden. Im informatischen Kontext entspricht der Begriff eines formalen Parameters einem Funktionsparameter oder einer Funktionsvariablen. [Bec05, S.16f]

Ist  $\Sigma$  eine Menge von formalen Parametern, dann ist  $E(\Sigma)$  ein arithmetischer Ausdruck, in dem Parameter, Konstanten und arithmetische Operatoren auf eine zulässige Weise kombiniert werden. [PL90, S.41]

**Parametrisches L-System:** 2.2.1 Ein Parametrisches L-System ist ein Tupel  $G = (V, \Sigma, P, \omega)$ , bestehend aus:

**V** Einem nichtleeren, endlichen Alphabet.

- $\Sigma$**       *Einer Menge von formalen Parametern.*
- $P$**       *Einer endlichen Menge von Produktionsregeln  $P : (V \times \Sigma^*) \rightarrow (V \times E(\Sigma)^*)^*$*
- $\omega \in M^+$**     *mit  $M = (V \times \mathbb{R}^*)$  – einem Axiom in Form eines nichtleeren, parametrischen Wortes.*
- [PL90, S.41]

Eine Produktionsregel kann auf ein parametrisches Wort angewendet werden wenn das Zeichen, welches dem Wort vorausgeht, und die Anzahl der Parameter mit dem Zeichen und der Parameteranzahl im Vorgänger der Produktionsregel übereinstimmen. [PL90, S.42]

Beispiel: Gegeben sei folgendes, parametrisches L-System:

$$\begin{aligned} \omega &: A(1,1) \\ p_1 &: A(x,y) \rightarrow A(x+1,y*2) \quad B(y) \\ p_2 &: B(x) \rightarrow B(x+1) \quad C \end{aligned} \tag{2.2}$$

Das Alphabet  $V$  und die Menge der formalen Parameter  $\Sigma$  gehen implizit aus der Angabe der Produktionsregeln hervor und werden in zukünftigen L-System-Gleichungen nicht angegeben. Die Entwicklung des L-Systems läuft wie in Abbildung 2.2 gezeigt ab.

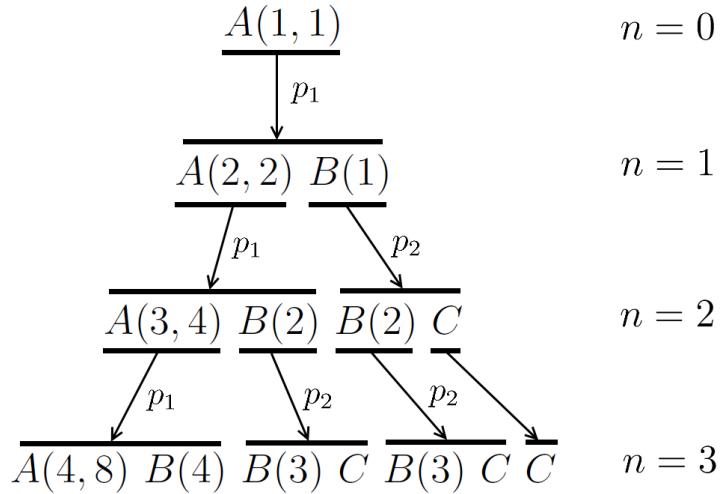


Abb. 2.2: Die n-fache Ableitung des Axioms  $A(1,1)$  anhand der Produktionsregeln aus Gleichung 2.2. Die Ersetzung des Zeichens  $C$  erfolgt anhand der impliziten Identitätsproduktion. Eigene Abbildung.

## 2.3 Grafische Interpretation von L-Systemen

Um die Ergebnisse von L-Systemen in Form von dreidimensionalen Objekten zu visualisieren, muss eine grafische Interpretation der resultierenden Zeichenketten festgelegt werden. Im Folgenden wird die verwendete Visualisierungsmethode – die Turtle-Interpretation – vorgestellt.

### 2.3.1 Turtle-Interpretation

Die Turtle-Interpretation im zweidimensionalen Raum entspricht der Vorstellung einer Turtle (engl. für Schildkröte) auf einem Blatt Papier. Die Turtle besitzt eine Position  $\vec{p}$  sowie einen Einheitsvektor  $\vec{H}$  in kartesischen Koordinaten.  $\vec{p}$  beschreibt die Position der Turtle auf der Ebene und  $\vec{H}$  entspricht der Blickrichtung (Heading) der Turtle. Der Zustand einer Turtle wird somit vollständig durch die Position und Blickrichtung definiert und wird als Tupel  $(\vec{p}, \vec{H})$  angegeben. [GSJ04, S.2] Die Ausgangsposition entspricht dem Ursprung des lokalen Koordinatensystems der Turtle.

Es können drei Aktionen durchgeführt werden, welche durch die folgenden Symbole dargestellt werden:

- $F(l)$**  Die Turtle bewegt sich um  $l > 0$  in die Richtung der aktuellen Blickrichtung. Die neue Position ist  $\overrightarrow{p_{neu}}$  mit:

$$\overrightarrow{p_{neu}} = \vec{p} + l * \vec{H} \quad (2.3)$$

Zwischen der alten Position  $\vec{p}$  und der neuen Position  $\overrightarrow{p_{neu}}$  wird eine Linie gezeichnet.

- $+(d)$**  Die Turtle dreht sich um den Winkel  $d$  nach links. Die neue Blickrichtung ist  $\vec{H}'$  mit:

$$\vec{H}' = \begin{pmatrix} \cos(d) & -\sin(d) \\ \sin(d) & \cos(d) \end{pmatrix} * \vec{H} \quad (2.4)$$

- $-(d)$**  Die Turtle dreht sich um den Winkel  $d$  nach rechts. Die neue Blickrichtung ist  $\vec{H}'$  mit:

$$\vec{H}' = \begin{pmatrix} \cos(d) & \sin(d) \\ -\sin(d) & \cos(d) \end{pmatrix} * \vec{H} \quad (2.5)$$

[GSJ04, S.4,46] [PL90, S.7] Die Symbole „+“ und „-“ werden sowohl im Alphabet eines L-Systems als auch bei arithmetischen Operationen in Parameterangaben verwendet, ihre Bedeutung ist abhängig vom Kontext, in dem sie angewendet werden. [PL90, S.46]

Die grafische Turtle-Interpretation einer Zeichenkette, die durch ein L-System zurückgegeben wird, sind somit die Linien, die auf Grundlage der definierten Symbole gezeichnet werden.

Beispiel: Mithilfe von L-Systemen und einer Turtle-Interpretation können Fraktale, in diesem Beispiel sogenannte Koch-Kurven, visualisiert werden. Diese Kurven bestehen aus einem Initiator – einer einfachen, zweidimensionalen Form – und einem Generator, der einem offenen Polygonzug entspricht. In jedem Ableitungsschritt, angefangen bei dem Initiator, wird jede gerade Linie durch den Generator ersetzt. [Man83, S.39]

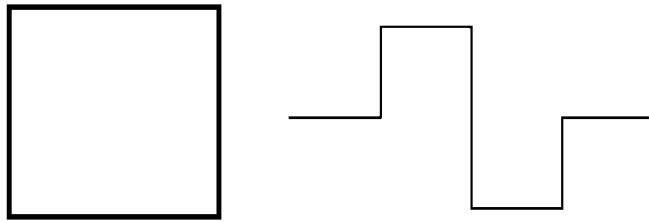


Abb. 2.3: Links: Initiator der Koch-Kurve in Form eines einfachen Quadrats. Rechts: Generator der Koch-Kurve in Form eines offenen Polygonzugs. Eigene Abbildungen.

Dieses Verhalten kann nun auf ein L-System abgebildet werden: Der Initiator entspricht dem Axiom und der Generator einer Produktionsregel des L-Systems. Der in Abbildung 2.3 dargestellte Initiator und Generator entsprechen der Turtle-Interpretation des folgenden L-Systems:

$$\begin{aligned} \omega &: F - F - F - F \\ p_1 &: F \rightarrow F + F - F - FF + F + F - F \end{aligned} \tag{2.6}$$

Für eine bessere Übersicht wurde die Angabe der Parameter weggelassen. Die Turtle interpretiert  $F$  als  $F(l)$ ,  $-$  als  $-(d)$  und  $+$  als  $+(d)$  mit festgelegter Strichlänge  $l$  und Drehwinkel  $d$ . Die Entwicklung des L-Systems läuft, als Turtle-Interpretation visualisiert, wie in Abbildung 2.4 dargestellt ab.

### 2.3.2 Verzweigte L-Systeme

Die bisherigen Definitionen von L-Systemen und die korrespondierende Turtle-Interpretation erlaubt lediglich die Bildung von Grafiken mit einem einzelnen, zusammenhängenden Polygonzug. Um L-Systeme zu bilden, deren Visualisierungen Baumstrukturen ähneln, muss die bisherige Turtle-Interpretation um die Möglichkeit erweitert werden Verzweigungen zu verarbeiten. [PL90, S.24]

Folgende Operationen werden eingeführt:

- [      Der aktuelle Zustand der Turtle in Form ihrer Position und Rotation wird auf einem Stack (engl. für Kellerspeicher) abgelegt.
- ]      Der oberste Zustand der Turtle wird vom Stack genommen. Die aktuelle Position und Rotation der Turtle wird auf die im Zustand gespei-

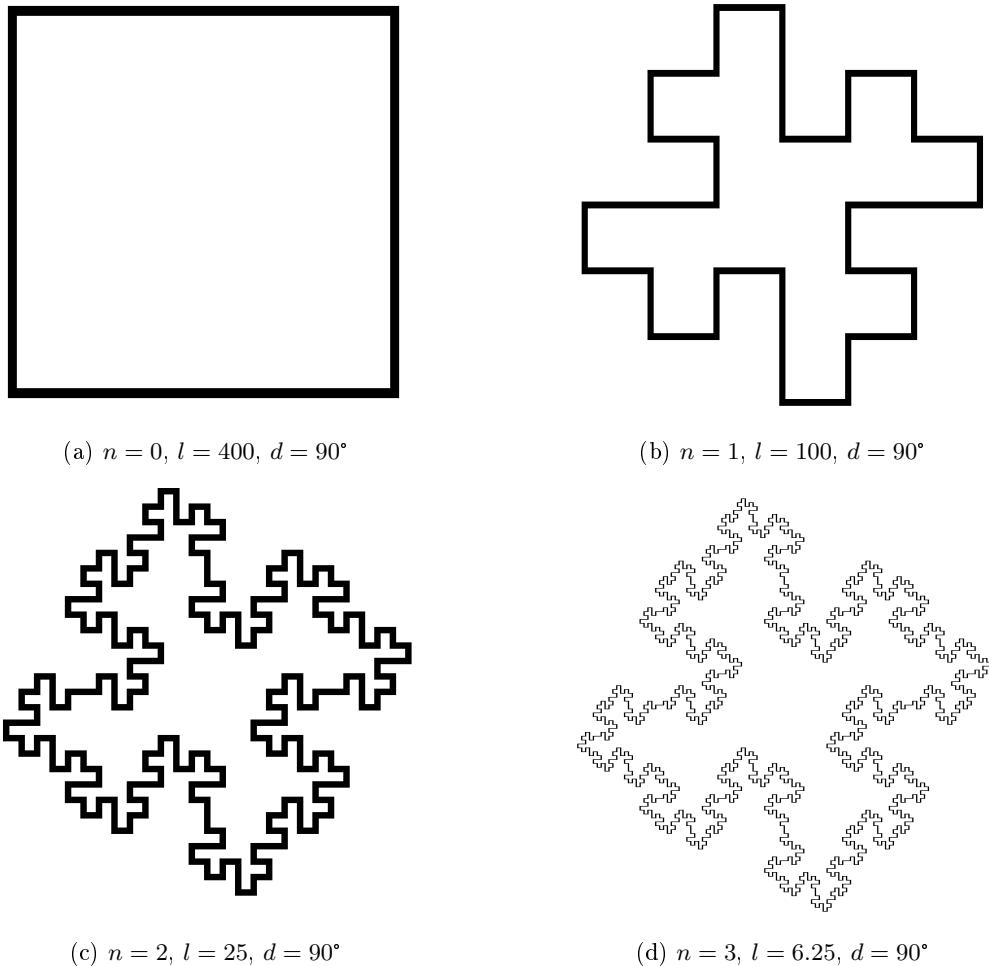


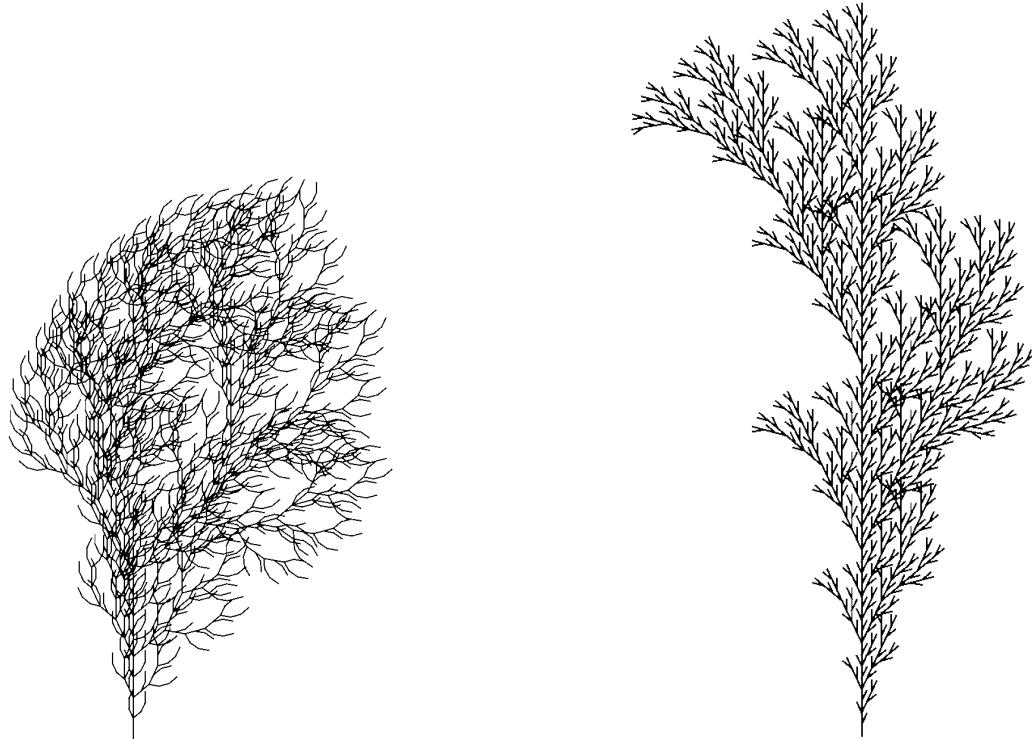
Abb. 2.4: Die  $n$ -fache Ableitung des Axioms  $\omega$  anhand der Produktionsregel  $p_1$  aus Gleichung 2.6, visualisiert mithilfe der implementierten Turtle-Interpretation. Eigene Abbildungen.

cherte Position und Rotation gesetzt. Es wird keine Linie zwischen der alten und neuen Position gezeichnet.

[PL90, S.24]

Diese Erweiterung erlaubt es mehrere Linien zu zeichnen, die von einem einzigen Punkt ausgehen und ermöglicht somit die Visualisierung von Abzweigungen. Die Operatoren [ und ] markieren den Anfang und das Ende eines Zweiges. [PL90, S.24]

Beispiel: Mithilfe von verzweigten L-System-Beschreibungen lassen sich die in Abbildung 2.5 gezeigten Strukturen bilden. Die Turtle-Interpretation folgt der in 2.3.1 beschriebenen Interpretation mit fester Strichlänge  $l$  und festem Drehwinkel  $d$ .

(a)  $n = 4, l = 18, d = 25^\circ$ 

$$\begin{aligned}\omega &: F \\ p_1 &: F \rightarrow FF - [-F + F + F] + [+F - F - F]\end{aligned}$$

[PL90, S.25]

(b)  $n = 5, l = 15, d = 25^\circ$ 

$$\begin{aligned}\omega &: F \\ p_1 &: F \rightarrow F[-F]F[+F][F]\end{aligned}$$

[STN16, S.78]

Abb. 2.5: Die n-fache Ableitung der Axiome anhand der Produktionsregeln, visualisiert mithilfe der implementierten Turtle-Interpretation. Eigene Abbildung.

### 2.3.3 Erweiterung der Turtle-Interpretation in den dreidimensionalen Raum

Die bisherige Definition eines Turtle-Zustands mithilfe einer zweidimensionalen Position und einer Blickrichtung genügt nicht, um Visualisierungen von L-Systemen in Form von dreidimensionalen Baumstrukturen zu ermöglichen. Sowohl die Zustands-Definition als auch die interpretierten Operationen müssen angepasst und erweitert werden.

Der Zustand der Turtle im dreidimensionalen Raum besitzt eine Position  $\vec{p}$  sowie eine  $3 \times 3$  Rotationsmatrix  $\mathbf{R}$ , welche die Orientierung der Turtle im Raum beschreibt. Der Zustand wird als Tupel  $(\vec{p}, \mathbf{R})$  angegeben.  $\mathbf{R}$  entspricht zu Anfang einer Identitätsmatrix. Die Einheitsvektoren  $\vec{H}$ ,  $\vec{L}$  und  $\vec{U}$  sind orthogonal zueinander und bilden das lokale Koordinatensystem der Turtle.

$\vec{H}$

Die Blickrichtung (Heading-Vektor) der Turtle. Eine Rotation um diesen Vektor um den Winkel  $d$  entspricht der Rotationsmatrix  $R_{\vec{H}}(d)$ :

$$R_{\vec{H}}(d) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(d) & \sin(d) \\ 0 & -\sin(d) & \cos(d) \end{pmatrix} \quad (2.7)$$

 $\vec{L}$ 

Der Vektor, der im lokalen Koordinatensystem der Turtle nach links zeigt (Left-Vektor). Eine Rotation um diesen Vektor um den Winkel  $d$  entspricht der Rotationsmatrix  $R_{\vec{L}}(d)$ :

$$R_{\vec{L}}(d) = \begin{pmatrix} \cos(d) & 0 & \sin(d) \\ 0 & 1 & 0 \\ -\sin(d) & 0 & \cos(d) \end{pmatrix} \quad (2.8)$$

 $\vec{U}$ 

Der Vektor, der im lokalen Koordinatensystem der Turtle nach oben zeigt (Up-Vektor). Eine Rotation um diesen Vektor um den Winkel  $d$  entspricht der Rotationsmatrix  $R_{\vec{U}}(d)$ :

$$R_{\vec{U}}(d) = \begin{pmatrix} \cos(d) & -\sin(d) & 0 \\ \sin(d) & \cos(d) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.9)$$

[PL90, S.19] [DL05, S.69]

Die erweiterte Turtle-Interpretation verarbeitet folgende Symbole:

 $F(l)$ 

Die Turtle bewegt sich um  $l > 0$  in Blickrichtung  $\vec{H}$ . Die neue Position ist  $\overrightarrow{p_{neu}}$  mit:

$$\overrightarrow{p_{neu}} = \overrightarrow{p} + l * (\mathbf{R} * \vec{H}) \quad (2.10)$$

Zwischen der alten Position  $p$  und der neuen Position  $p_{neu}$  wird eine Linie gezeichnet.

 $+(d)$ 

Die Turtle dreht sich nach links um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{U}}(d) \quad (2.11)$$

 $-(d)$ 

Die Turtle dreht sich nach rechts um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{U}}(-d) \quad (2.12)$$

 $\&(d)$ 

Die Turtle neigt sich nach unten um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{L}}(d) \quad (2.13)$$

 $^{\wedge}(d)$ 

Die Turtle neigt sich nach oben um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{L}}(-d) \quad (2.14)$$

- \(**d**) Die Turtle rollt sich nach links um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{H}}(d) \quad (2.15)$$

- /(**d**) Die Turtle rollt sich nach rechts um den Winkel  $d$ . Die neue Rotationsmatrix entspricht  $\mathbf{R}'$  mit:

$$\mathbf{R}' = \mathbf{R} * R_{\vec{H}}(-d) \quad (2.16)$$

[PL90, S.19] [DL05, S.69]

Mithilfe der Turtle-Interpretation im dreidimensionalen Raum können nun L-Systeme visualisiert werden, die realen Baumstrukturen ähneln. Ein Beispiel dafür ist das L-System aus Gleichung 2.17, dargestellt in Abbildung 2.6.

$$\begin{aligned} \omega : & / (45) A \\ p_1 : A & \rightarrow F(50) [\&(a)F(50)A] / (d) [\&(a)F(50)A] / (d) [\&(a)F(50)A] \\ p_2 : F(l) & \rightarrow F(l * l_r) \end{aligned} \quad (2.17)$$

[PL90, S.60]

## 2.4 Anpassungen für Baumstrukturen

Die Darstellung von verzweigten L-Systemen mithilfe von Zeichenketten ermöglicht eine Repräsentation in Textform und erlaubt die einfache Definition von Produktionsregeln. Um jedoch eine effiziente Nachbearbeitung und Visualisierung der Ergebnisse eines L-Systems zu ermöglichen, müssen die Aktionen einer Turtle als graphentheoretischer Baum festgehalten werden. Weiterhin wird der Begriff des Tropismus und dessen Einfluss auf die Modellierung von Baumstrukturen eingeführt.

### 2.4.1 Graphentheoretische Bäume

Ein Baum, in der Graphentheorie, ist ein kreisfreier Graph  $G = \langle V, E \rangle$  für den folgende Begriffe festgelegt werden:

- V** Die Menge der Knoten. Jeder Knoten entspricht einem Punkt  $\vec{p}_v$  im dreidimensionalen Raum. [Sch14, S.358]
- E** Die Menge der Kanten, welche die Vorgänger-Nachfolger-Beziehungen zwischen den Knoten darstellt. Eine Kante  $e \in E$  wird als Tupel  $e = (v_1, v_2)$  mit  $v_1, v_2 \in V$  dargestellt.  $v_1$  wird als Vorgänger von  $v_2$  und  $v_2$  als Nachfolger von  $v_1$  bezeichnet. Jeder Knoten besitzt maximal einen Vorgänger und eine endliche Menge von Nachfolgern. [Sch14, S.358] [Lux14, S.29]

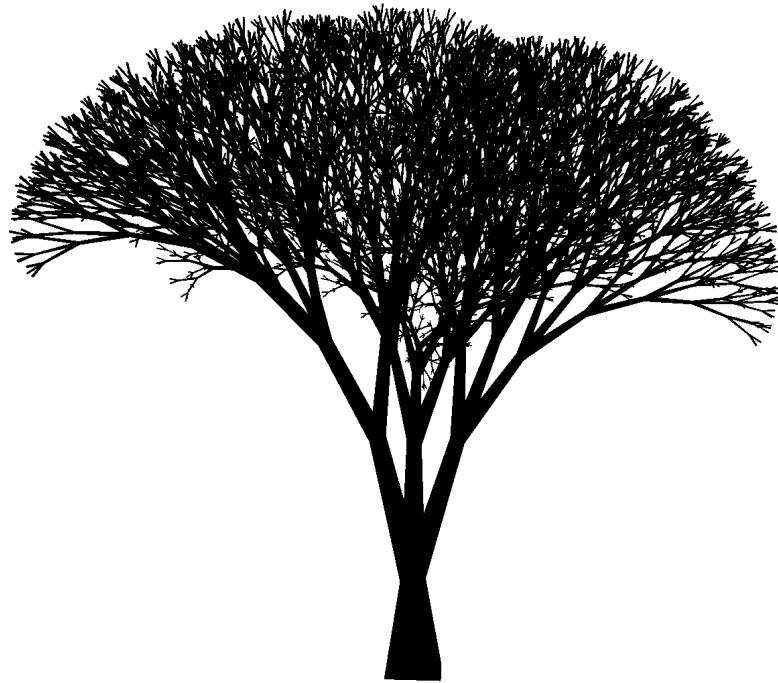
(a)  $n = 8, d = 137.5^\circ, a = 18.95^\circ, l_r = 1.3$ (b)  $n = 8, d = 137.5^\circ, a = 36.0^\circ, l_r = 1.3$ 

Abb. 2.6: Die n-fache Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 2.17, visualisiert mithilfe der implementierten Turtle-Interpretation. Eigene Abbildungen.

**Wurzel** Eine Wurzel ist ein Knoten  $v \in V$  ohne einen Vorgänger. Der Baum besitzt genau eine Wurzel. [Sch14, S.358]

**Grad** Der Grad eines Knotens ist definiert als die Anzahl seiner Nachfolger. [Lux14, S.29]

**Tiefe** Die Tiefe eines Knotens ist definiert als die Länge der Folge von Vorgängern, die durchlaufen werden muss, bis die Wurzel erreicht wurde. Die Wurzel besitzt die Tiefe 0. [Lux14, S.30]

Ein Baum ist grundsätzlich ein rein topologisches Objekt, mithilfe der Zuordnung von Punkten zu Knoten kann jedoch eine geometrische Vorstellung des Aufbaus bewirkt werden. [PL90, S.23]

Um den Aufbau eines Baumes  $G = \langle V, E \rangle$  mithilfe der Turtle-Interpretation zu ermöglichen, muss der Turtle-Zustand um einen Knotenpunkt  $v \in V$  ergänzt werden. Der Zustand wird als Tupel  $(\vec{p}, v, \mathbf{R})$  angegeben.

Die Turtle-Interpretation beginnt mit der Erstellung der Wurzel am Ursprung des lokalen Bezugssystems. Die Verarbeitung folgender Symbole muss erweitert werden:

**$F(l)$**  Die neue Position  $\vec{p}_{neu}$  der Turtle wird entsprechend Gleichung 2.10 berechnet. An diesem Punkt wird ein Knoten  $v_{neu}$  und eine Kante  $(v, v_{neu})$  erstellt sowie  $v$  als Vorgänger von  $v_{neu}$  und  $v_{neu}$  als Nachfolger von  $v$  eingetragen.  $v$  entspricht dem aktuellen Zustandsknoten der Turtle. Der neue Turtle-Zustand ist  $(\vec{p}_{neu}, v_{neu}, \mathbf{R})$ .

- [ Zusätzlich zu der Position und Rotation der Turtle wird auch der aktuelle Knoten auf einem Stack abgelegt.
- ] Der oberste Zustand der Turtle wird vom Stack genommen. Zusätzlich zur Position und Rotation wird auch der Zustandsknoten der Turtle auf den gespeicherten Knoten gesetzt.

Die Kanten zwischen Knoten wurden durch die Turtle-Interpretation bisher als Linien visualisiert und können als Astsegmente von echten Baumstrukturen verstanden werden. [PL90, S.23] Die Repräsentation von Turtle-Aktionen als graphentheoretischer Baum ermöglicht die in Abschnitt 4.4 beschriebene Generierung und Nachbearbeitung von Modelldaten.

### 2.4.2 Tropismus

Tropismus ist die Tendenz einer Pflanze in eine bestimmte Richtung zu wachsen, beispielsweise aufgrund einer Lichtquelle oder der Beugung durch Gravitation. [RLP07, Abschn. 3] Der Einfluss von Tropismus wird als ein dreidimensionaler Vektor  $\vec{T}$  angegeben. Die Bewegung  $F(l)$  der Turtle wird wie folgt erweitert:

**F(l)** Die Turtle bewegt sich um  $l > 0$  in Blickrichtung  $\vec{H}$ . Die neue Position ist  $\overrightarrow{p_{neu}}$  mit:

$$\overrightarrow{p_{neu}} = \overrightarrow{p} + l * \frac{\overrightarrow{H_{rot}} + e * \vec{T}}{\|\overrightarrow{H_{rot}} + e * \vec{T}\|} \quad (2.18)$$

$$\text{und } \overrightarrow{H_{rot}} = \mathbf{R} * \vec{H} \quad (2.19)$$

wobei  $e$  der Anfälligkeit des Baums für die Beugung durch Tropismus entspricht, im Folgenden als Biegsamkeitsfaktor bezeichnet. [PL90, S.58] [RLP07, Abschn. 3]

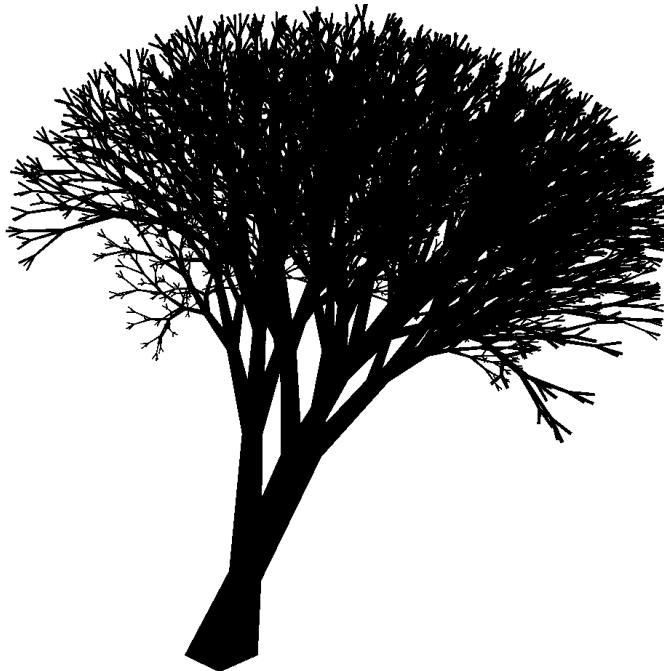


Abb. 2.7: .

$$n = 8, d = 137.5^\circ, a = 36.0^\circ, l_r = 1.3, \vec{T} = \begin{pmatrix} 0 \\ 1 \\ -0.5 \end{pmatrix}, e = 0.27$$

Die n-fache Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 2.17, visualisiert mithilfe der erweiterten Turtle-Interpretation. Zeigt den Einfluss von Tropismus auf den in Abbildung 2.6a dargestellten Baum. Eigene Abbildung.

# 3

---

## Space Colonization Algorithmus

Es werden die benötigten Eingaben und der Ablauf des Space Colonization Algorithmus behandelt. Die Prozedur zur Generierung von Baumstrukturen wird vorgestellt und die implementierten Erweiterungen des ursprünglichen Algorithmus werden erläutert.

### 3.1 Ursprung

Der vorgestellte Space Colonization Algorithmus wurde ursprünglich zur Modellierung und Visualisierung von Blattvenen entwickelt und basiert auf der Wirkung des Pflanzenhormons Auxin. Dieser Hormonstoff entsteht im Blatt und wird von bereits existierenden Blattvenen angezogen, der resultierende Hormonstrom führt zur Bildung von neuen Venen im Blatt. Die Simulation dieses Vorgangs führt zu realitätsnahen Venenmustern. [RFL<sup>+</sup>05, S.3]

Mithilfe einer Erweiterung in den dreidimensionalen Raum und Nachbearbeitung der Resultate kann eine Vielfalt von Baum- und Strauchstrukturen generiert werden. [RLP07, S.2]

### 3.2 Aufbau

Der Algorithmus verarbeitet eine Menge von Einflusspunkten  $S$  und baut darauf basierend einen Baum  $G = \langle V, E \rangle$  auf.

Der Algorithmus benötigt die folgenden Eingaben:

$d_i$  Der Einflussradius. Einflusspunkte prägen den Aufbau des Baums nur, wenn sich Knotenpunkte innerhalb dieses Radius befinden. [RLP07, S.3]

$d_k$  Der Minimalradius. Befindet sich ein Knotenpunkt innerhalb des Minimalradius um einen Einflusspunkt, wird dieser aus der Menge der Einflusspunkte  $S$  entfernt. [RLP07, S.3]

**D** Die Schrittweite. Jeder neu generierte Knotenpunkt wird in diesem Abstand zu seinem Vorgänger positioniert. [RLP07, S.3]

**$\vec{T}$**  Der Tropismusvektor. [RLP07, S.3]

### 3.3 Ablauf

Zu Beginn des Algorithmus werden  $N$  Einflusspunkte in einem vorgegebenen Bereich generiert. Dieser Einflussbereich signalisiert die Verfügbarkeit von Raum, in dem die Baumstruktur entstehen kann. [RLP07, S.3]

Daraufhin wird der Baum iterativ aufgebaut und durchläuft in jeder Iteration die folgenden Schritte:

1. Für jeden Einflusspunkt in  $S$  wird der am nächsten liegende Knotenpunkt  $v \in V$  bestimmt. Befindet sich  $v$  innerhalb des Einflussradius  $d_i$  eines Einflusspunktes, wird der Knotenpunkt einer zugeordneten Menge  $S(v)$  hinzugefügt.  $S(v)$  beinhaltet somit alle Einflusspunkte, die einen Einfluss auf den Knotenpunkt ausüben. [RLP07, S.3]
2. Befinden sich Elemente in  $S(v)$ , wird ein neuer Knotenpunkt  $v_n$  den Nachfolgern von  $v$  hinzugefügt und  $v$  als Vorgänger von  $v_n$  eingetragen. Alle Punkte in  $S(v)$  beeinflussen  $v_n$  in gleichem Maße, die neue Position  $\vec{p}_{v_n}$  des Knotenpunkts berechnet sich somit wie folgt:

$$\vec{p}_{v_n} = \vec{p}_v + D * \vec{n}_T \quad (3.1)$$

$$\text{mit } \vec{n}_T = \frac{\vec{n} + \vec{T}}{\|\vec{n} + \vec{T}\|} \quad (3.2)$$

$$\text{und } \vec{n} = \sum_{s \in S(v)} \frac{\vec{p}_s - \vec{p}_v}{\|\vec{p}_s - \vec{p}_v\|} \quad (3.3)$$

[RLP07, S.3]

3. Für jeden Einflusspunkt wird überprüft, ob sich ein Knotenpunkt innerhalb des Minimalradius  $d_k$  befindet. Existiert ein solcher Knotenpunkt, wird der Einflusspunkt aus der Menge der Einflusspunkte  $S$  entfernt. [RLP07, S.3]

Diese Schritte werden solange ausgeführt, bis alle Einflusspunkte entfernt wurden, sich kein Knotenpunkt im Einflussradius eines Einflusspunktes befindet oder bis eine vorgegebene Maximalanzahl von Iterationen durchgeführt wurde. [RLP07, S.2] Abbildung 3.1 zeigt eine beispielhafte Anwendung des Algorithmus.

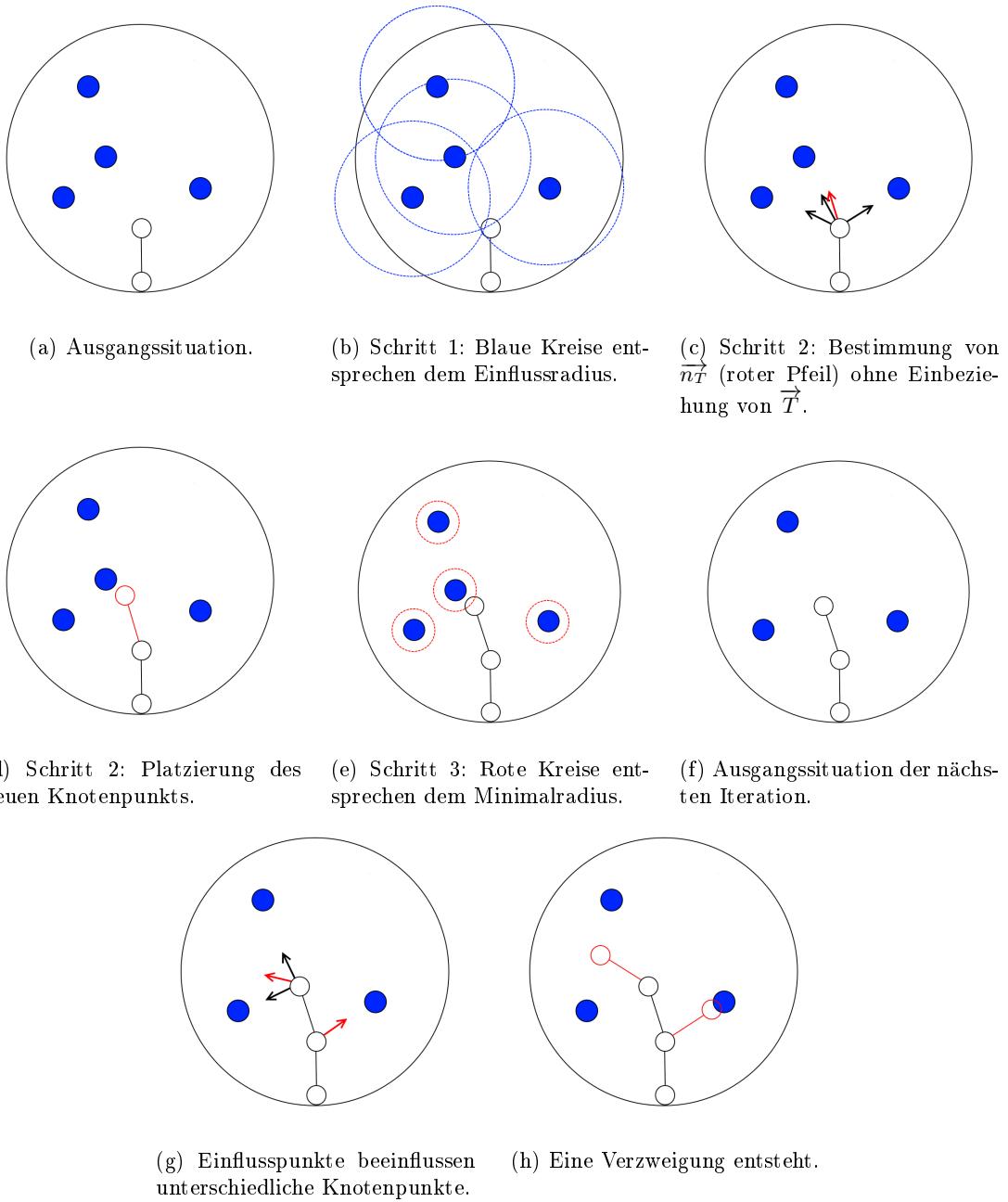


Abb. 3.1: Beispielhafte Anwendung des Space Colonization Algorithmus. Blaue Punkte entsprechen Einflusspunkten, weiße Punkte entsprechen Knotenpunkten. Der unterste Knotenpunkt stellt die Wurzel dar. Eigene Abbildungen auf Grundlage von [RLP07, Abb. 2].

### 3.4 Generierung von Baumstrukturen

Der Space Colonization Algorithmus liefert einen Baum, der Knotenpunkte enthält, welche Positionen im dreidimensionalen Raum entsprechen. Um diese Knotenpunkte in Form von baumähnlichen Strukturen zu visualisieren, wird die Prozedur erweitert. Der Aufbau der Baumstruktur läuft wie folgt ab:

1. Der Einflussbereich wird mit der vorgegeben Anzahl von Einflusspunkten gefüllt. [RLP07, S.2]
2. Der Baum wird, wie in Abschnitt 3.3 beschrieben, iterativ generiert. [RLP07, S.2]
3. Die Nachfolger jedes Knotenpunkts werden einander angenähert, um eine Verringerung der Abzweigungswinkel zwischen den verbindenden Kanten zu erreichen. Dies führt zu einer insgesamt realistischeren Baumstruktur. [RLP07, S.2]
4. Die Kanten, welche die Knotenpunkte verbinden, werden mithilfe von Zylindern visualisiert, um die Aststruktur eines biologischen Baumes zu simulieren. [RLP07, S.2]

Die von Runions u.a. [RLP07] vorgeschlagene Kurven-Unterteilung [RLP07, S.2] wurde in dieser Arbeit nicht behandelt, da durch Angabe der Schrittweite  $D$  eine ausreichende, visuelle Qualität erzielt wurde.

Abbildung 3.2 zeigt die Modellierung einer zweidimensionalen Baumstruktur.

#### *Verringerung der Abzweigungswinkel*

Die  $m$  Nachfolger  $v_1 \dots v_m$  eines Knotens  $v$  werden wie folgt einander angenähert:

$$\vec{m} = \vec{p}_v + \frac{\sum_{i=0}^m (\vec{p}_{v_i} - \vec{p}_v)}{m} \quad (3.4)$$

wobei  $\vec{m}$  dem arithmetischen Mittelpunkt der Nachfolger entspricht. Die Positionen der Nachfolger werden nun wie folgt dem Mittelpunkt angenähert:

$$\vec{p}_{v_i} = \vec{p}_v + \frac{\vec{m} - \vec{p}_v}{2} \quad (3.5)$$

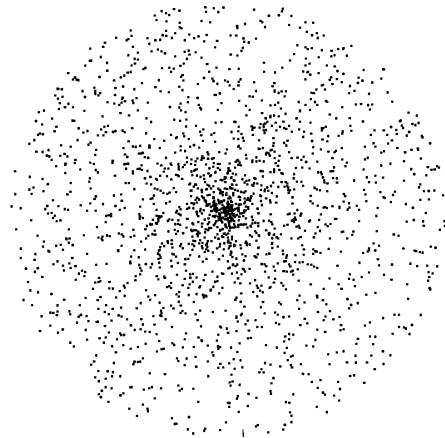
[RLP07, S.2]

#### *Berechnung der Zylinderbreiten*

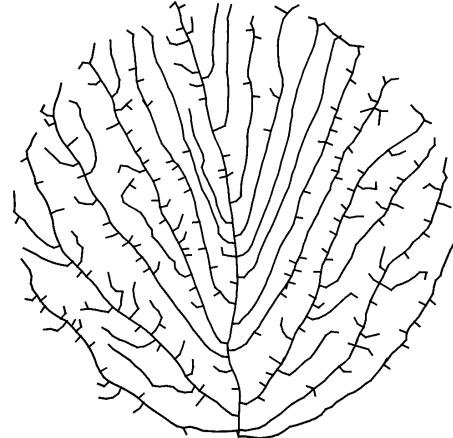
Die Berechnung der Zylinderbreiten erfolgt anhand von Murrays Regel, welche besagt, dass der Radius  $r$  eines Astes auf Grundlage der Radien  $r_{n_1} \dots r_{n_m}$  der nachfolgenden, von ihm abzweigenden Äste wie folgt berechnet werden kann:

$$r^g = r_{n_1}^g + r_{n_2}^g + \dots + r_{n_m}^g \quad (3.6)$$

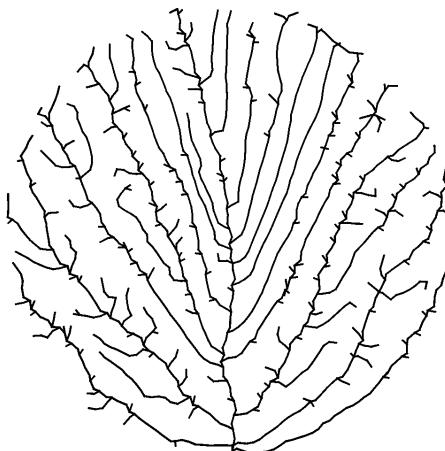
Diese Berechnung kann rekursiv auf dem Baum ausgeführt werden, indem für jeden Knotenpunkt der Radius aus den Radien seiner Nachfolger berechnet wird. Besitzt ein Knotenpunkt keine Nachfolger, wird ein festgelegter Radius  $r_0$  zurückgegeben. Der Wert  $g$  kann frei gewählt werden. [RFL<sup>+</sup>05, S.5]



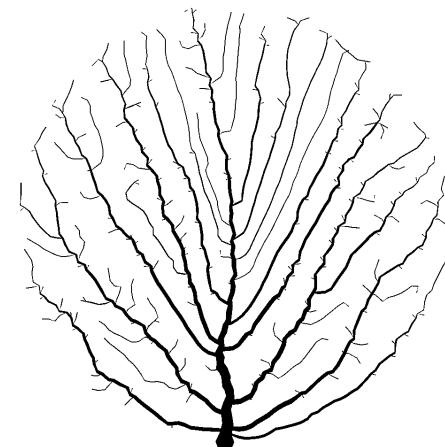
(a)  $N = 2000$  Einflusspunkte, zufällig in einem Ring mit dem Radius  $r = 500$  verteilt.



(b) Ergebnis des Space Colonization Algorithmus mit einem Einflussradius  $d_i = 100$ , Minimalradius  $d_k = 20$ , Schrittwerte  $D = 15$  und  $\vec{T} = \vec{0}$ .



(c) Verringerung der Abzweigungswinkel.



(d) Modellierung des Ergebnis mithilfe von Zylindern mit  $r_0 = 1$  und  $g = 2$ .

Abb. 3.2: Modellierung einer zweidimensionalen Baumstruktur entsprechend der in Abschnitt 3.4 beschriebenen Schritte. Eigene Abbildungen.

### 3.5 Erweiterungen

Im Rahmen dieser Arbeit wurden die folgenden Erweiterungen des ursprünglichen Algorithmus entwickelt und implementiert.

### Zweigtiefe

Die Zweigtiefe  $Z(v)$  eines Knotens  $v \in V$  wird für Erweiterungen des Space Colonization Algorithmus und die Generierung von Modelldaten verwendet. Sie entspricht nicht der Tiefe des Knotens und berechnet sich wie folgt:

$$Z(v) = \begin{cases} 0 & \text{falls } v \text{ die Wurzel ist} \\ Z(v') & \text{falls } v' \text{ genau einen Nachfolger besitzt} \\ 1 + Z(v') & \text{sonst} \end{cases} \quad (3.7)$$

wobei  $v'$  den Vorgänger des Knotens  $v$  darstellt. Die Zweigtiefe wird direkt nach der Erstellung des Knotens berechnet und daraufhin nicht verändert. Ein Beispiel für die Bestimmung der Zweigtiefe wird in Abbildung 3.3 gezeigt.

Der von Prusinkiewicz u.a. [PL90] beschriebene Begriff von axialen Bäumen [PL90, S.21] und die zugehörige Bestimmung der Achsentiefe unterscheidet sich von der Berechnung der Zweigtiefe. Knoten können dieselbe Zweigtiefe besitzen, selbst wenn die Verbindung ihrer Positionen keine gerade Linie bildet.

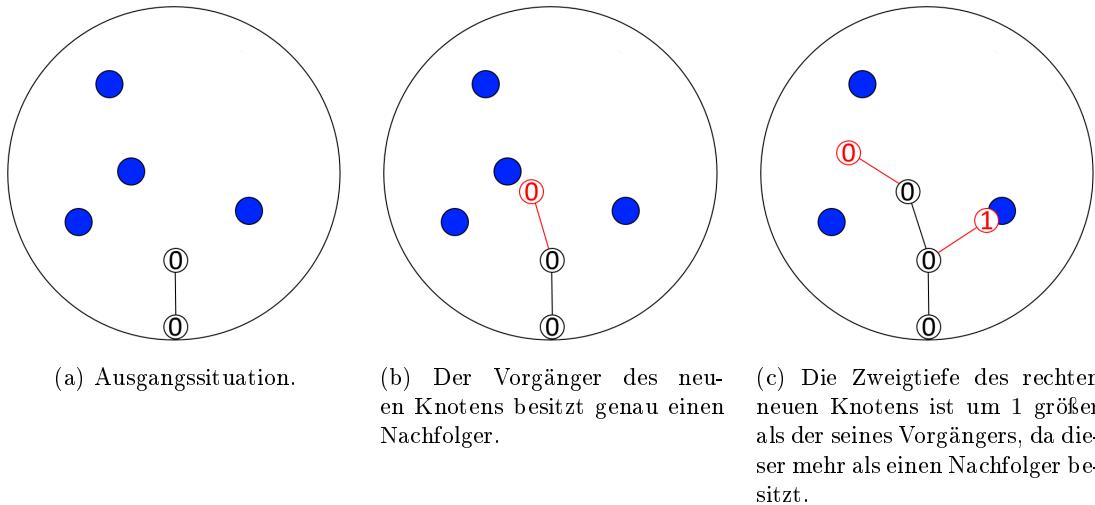


Abb. 3.3: Darstellung der Zweigtiefen von Knotenpunkten aus Abbildung 3.1.

### Zusätzliche Bedingungen

Um die vom Space Colonization Algorithmus generierte Baumstruktur besser kontrollieren zu können, wurden zusätzliche Bedingungen zum Ablauf des Algorithmus hinzugefügt:

**$max_{grad}$**  Der maximale Grad der Knotenpunkte im Baum. Bevor ein neuer Knotenpunkt  $v_n$  als Nachfolger eines Knotenpunkts  $v$  erstellt wird, wird die Anzahl der Nachfolger von  $v$  überprüft. Entspricht diese  $max_{grad}$ , wird der Knotenpunkt  $v_n$  nicht erstellt.

- $max_Z$**  Die maximale Zweigtiefe eines Knotens. Bevor ein neuer Knotenpunkt  $v_n$  als Nachfolger eines Knotenpunkts  $v$  erstellt wird, wird seine Zweigtiefe  $Z(v_n)$  berechnet. Übersteigt diese  $max_Z$ , wird der Knotenpunkt  $v_n$  nicht erstellt.
- $max_{NG}$**  Die maximale Anzahl von Iterationen, in welchen einem Knotenpunkt kein neuer Nachfolger hinzugefügt wurde.<sup>1</sup> Hat ein Knoten diesen Wert erreicht, wird er in Schritt 1 des Algorithmus 3.3 nicht weiter darauf untersucht, ob er sich innerhalb des Einflussradius eines Einflusspunktes befindet. Der Wert von  $max_{NG}$  eines Knotenpunkts wird auf 0 zurückgesetzt, falls diesem ein neuer Nachfolger hinzugefügt wurde. Durch die vorsichtige Wahl von  $max_{NG}$  können Positionsvergleiche vermieden werden ohne den resultierenden Baum visuell erkennbar zu verändern.

### Gewichtetes Wachstum

Gewichtetes Wachstum ist eine Möglichkeit die Schrittweite  $D$  in Abhängigkeit von der Zweigtiefe zu erhöhen. Befindet sich ein Knotenpunkt auf der Zweigtiefe 0, werden neu hinzugefügte Nachfolger im Abstand von  $2*D$  positioniert, befindet er sich auf der maximalen Zweigtiefe, werden neu hinzugefügte Nachfolger im Abstand von  $D$  positioniert. Zwischen diesen beiden Zweigtiefen wird die Schrittweite linear interpoliert.

### Kurvenreduktion

Kurvenreduktion in Abhängigkeit des Abzweigungswinkels ermöglicht es die Baumstruktur mithilfe einer verringerten Datenmenge darzustellen. Besitzt ein Knoten  $v$  genau einen Nachfolger  $v_n$  und einen Vorgänger  $v'$ , werden die zwei Richtungsvektoren zwischen  $v'$  und  $v$  sowie  $v$  und  $v_n$  berechnet:

$$\vec{R}_1 = \frac{\vec{p}_v - \vec{p}_{v'}}{\|\vec{p}_v - \vec{p}_{v'}\|} \text{ und } \vec{R}_2 = \frac{\vec{p}_{v_n} - \vec{p}_v}{\|\vec{p}_{v_n} - \vec{p}_v\|} \quad (3.8)$$

Übersteigt das Skalarprodukt  $\langle \vec{R}_1, \vec{R}_2 \rangle$  einen festgelegten Maximalwert  $max_K$  zwischen 0 und 1, wird  $v$  aus dem Baum entfernt.  $v_n$  wird zum Nachfolger von  $v'$  und  $v'$  zum Vorgänger von  $v_n$ .

Die Kurvenreduktion kann auch auf Bäume angewendet werden, welche wie in Abschnitt 2.4 beschrieben durch eine Turtle-Interpretation aufgebaut wurden.

---

<sup>1</sup> NG steht für „No Grow“ oder „Did not grow“, englisch für „kein Wachstum“.

# 4

---

## Implementierung

Im folgenden Kapitel wird die Implementierung der Vorgehen innerhalb des Frameworks der Unreal Engine 4 behandelt. Die Baumrepräsentation enthält Daten, die von den L-System und Space Colonization Implementierungen generiert werden. Diese Daten werden an das Modellgenerierungssystem übergeben, welches die Modelldaten für eine grafische Darstellung in der Unreal Engine 4 produziert.

### 4.1 Baumrepräsentation

Sowohl L-Systeme als auch der Space Colonization Algorithmus generieren einen graphentheoretischen Baum, auf dessen Grundlage die Modellgenerierung durchgeführt wird. Die implementierte Baumrepräsentation kann daher von beiden Systemen verwendet werden und ermöglicht es, diese mit demselben Modellgenerierungssystem zu visualisieren.

Der Baum wird durch eine Datenklasse repräsentiert, jedes Objekt dieser Klasse beschreibt einen Knoten sowie die Kante, welche vom Vorgänger zu dem Knoten führt. Die Datenklasse bietet Zugriff auf die folgenden Informationen:

**Vorgänger und Nachfolger:** Mithilfe eines Verweises auf den Vorgänger und einer Liste der Nachfolger eines Knotens kann der Baum-Graph vollständig repräsentiert werden. Weiterhin ermöglicht dies die Implementierung einer Reihe von rekursiven Funktionen zur Anpassung von Modelldaten.

**Modell-Daten:** Kanten werden, wie in Abschnitt 4.4.4 beschrieben, mithilfe von Zylindern visualisiert. Um die Generierung von Modelldaten zu vereinfachen, bietet die Datenklasse Zugriff auf Start- und Endposition, Start- und Endradius, Start- und Endnormale sowie einen Rotationswinkel.

Des Weiteren wird die Zweigtiefe des repräsentierten Knoten gespeichert.

**Wachstums-Daten:** Die Wachstums-Daten bestehen aus einer Wachstumsrichtung, einem Einfluss-Zähler und dem „Kein Wachstum“-Zähler (*NG-Counter*), welche für den Ablauf des Space Colonization Algorithmus benötigt werden.

Ein Objekt der Datenklasse kann als Astsegment eines biologischen Baumes angesehen werden und wird durch das Modellgenerierungssystem als solches vi-

sualisiert. Im Folgenden wird der Begriff „Astsegment“ verwendet, um ein Objekt der Datenklasse der Baumrepräsentation zu bezeichnen.

## 4.2 L-Systeme

Die Implementierung der Funktionsweise von L-Systemen wird durch einen Unreal-Actor verwirklicht, der im Level platziert werden kann. Nach Start des Levels wird das angegebene Axiom anhand der Produktionsregeln abgeleitet und die sich ergebende Zeichenkette von der Turtle-Implementierung interpretiert.

### 4.2.1 Parameter

Dem L-System-Actor werden die folgenden Parameter über die Editor-UI übergeben:

**Anzahl der Ableitungen:** Die Anzahl der Ableitungen in  $\mathbb{N}^+$ , welche auf dem Axiom durchgeführt werden.

**Axiom:** Das Axiom in Form einer Zeichenkette.

**Konstanten:** Eine Konstante besteht aus der Angabe eines Identifikationssymbols und eines Wertes in  $\mathbb{R}$ . Konstanten können im Axiom und in den Nachfolgern der Produktionen verwendet werden.

**Produktionen:** Jede Produktion besteht aus Angabe eines Vorgängers, einer Liste von Parametern und einem Nachfolger. Der Vorgänger und jeder Parameter entspricht einem einzelnen Symbol, der Nachfolger wird als Zeichenkette einge tragen. Die Parametersymbole können nur innerhalb des Nachfolgers verwendet werden. Der Vorgänger und die Liste der Parameter bilden das parametrische Wort, welches bei einer Ableitung durch den Nachfolger ersetzt wird.

**Tropismus:** Der Einfluss von Tropismus in Form eines dreidimensionalen Vektors und eines Biegsamkeitsfaktors.

Für das Axiom und die Produktionen gelten die in Kapitel 2 festgelegten Regeln für die Definition von L-Systemen. Weiterhin gelten die Regeln für die Angabe von arithmetischen Operationen. Die Verwendung von Klammern ist jedoch auf die Angabe von Parametern eines parametrischen Wortes beschränkt, ihre Verwendung zur Beeinflussung der Auswertungsreihenfolge eines arithmetischen Ausdrucks wird nicht unterstützt.

Ein Beispiel für die korrekte Eingabe eines L-Systems über die Editor-UI wird in Abbildung 4.1 gezeigt.

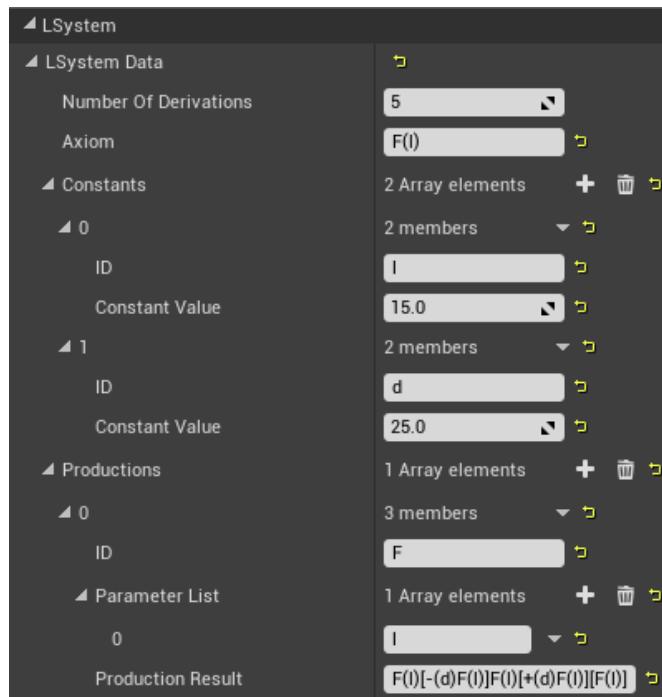


Abb. 4.1: Ein Beispiel für die Angabe des L-Systems aus Gleichung 2.5b mit der resultierenden Baumstruktur aus Abbildung 2.5b.

### 4.2.2 Ableitung

Zu Anfang der Erstellung des L-System-Actors werden alle Konstantensymbole im Axiom und den Produktionen durch die Konstantenwerte ersetzt.

Die Implementierung arbeitet durchgehend auf derselben Zeichenkette, angefangen mit dem Axiom. In jeder Ableitung werden die in den Produktionsregeln definierten parametrischen Wörter durch die angegebenen Nachfolger ersetzt.

Nachdem die vorgegebene Anzahl von Ableitungen durchgeführt wurde, wird die resultierende Zeichenkette an die Turtle-Implementierung weiter geleitet.

### 4.2.3 Turtle Interpretation

Der Turtle-Implementierung wird die aus den Ableitungen resultierende Zeichenkette übergeben. Diese wird daraufhin sequentiell abgearbeitet und entsprechend den in Abschnitt 2.4 vorgestellten Konzepten interpretiert. Der resultierende Baum wird für die Konstruktion des Modells an das Modellgenerierungssystem weitergegeben.

## 4.3 Space Colonization Algorithmus

Die Implementierung einer Space-Colonization-Algorithmus-Baumstruktur stellt sich aus der Platzierung von mindestens einem Actor für die Repräsentation des Einflussbereichs und einem Actor für die Generierungsprozedur zusammen.

### 4.3.1 Einflussbereiche

Durch die Platzierung von Einflussbereich-Actors kann die Verteilung von Einflusspunkten mithilfe des Unreal-Editors angepasst werden. Es sind derzeit zwei Formen von Einflussbereichen wählbar: Eine Kugel- und eine Zylinderform. Die Kugelform erfordert die Angabe eines Kugelradius während die Zylinderform durch Höhe und Radius beschrieben wird.

Einem Space-Colonization-Actor können mehrere Einflussbereiche zugeordnet werden, um eine bestimmte Baumstruktur zu formen. Jeder Einflussbereich-Actor wird weiterhin in einem vorgegebenen Abstand zum Space-Colonization-Actor platziert.

Dem Einflussbereich muss eine positive Anzahl von zu generierenden Einflusspunkten und ein Random-Seed Wert übergeben werden. Ein Random-Seed ist ein Wert, der von einem Zufallsgenerator verwendet wird, um eine zufällig wirkende Folge von Zahlen zu generieren. Bei Verwendung desselben Random-Seeds wird dieselbe Folge von Zufallszahlen erstellt, was eine gewisse Kontrolle über die Generierung ermöglicht. Somit wird, wenn alle anderen Parameter ebenfalls übereinstimmen, mit demselben Random-Seed Wert dieselbe Baumstruktur aufgebaut.

### 4.3.2 Parameter

Dem Space-Colonization-Actor werden Parameter des ursprünglichen Algorithmus sowie Eingaben für die in Abschnitt 3.5 besprochenen Erweiterungen über das Editor-UI übergeben. Zu den ursprünglichen Parametern gehören der Minimalradius, Einflussradius, Schrittweite, ein Tropismusvektor sowie die Anzahl der durchzuführenden Iterationen. Zu den erweiterten Parametern gehören der maximale Grad, die maximale Zweigtiefe, die maximale Anzahl von „Kein Wachstum“-Iterationen und eine Abfrage, ob gewichtetes Wachstum durchgeführt werden soll.

Weiterhin müssen die zugeordneten Einflussbereich-Actors angegeben werden – damit der Algorithmus durchgeführt werden kann, muss mindestens einer dieser Actors mit mindestens einem Einflusspunkt eingetragen werden.

### 4.3.3 Ablauf des Algorithmus

Die Einflusspunkte aller dem Space-Colonization-Actor zugeordneten Einflussbereiche werden diesem zu Beginn der Baum-Generierung übergeben. Daraufhin wird ein Baum entsprechend der in Abschnitt 3.4 und Abschnitt 3.5 vorgestellten Konzepte aufgebaut und für die Konstruktion des Modells an das Modellgenerierungssystem weitergegeben.

## 4.4 Modellgenerierung

L-System- und Space-Colonization-Actors übergeben dem Modellgenerierungssystem nach Ablauf der Generierung einen graphentheoretischen Baum. Das Modellgenerierungssystem erstellt daraus ein dreidimensionales Mesh (engl. für Polygon-

netz) in der vom Framework geforderten Form. Das Mesh entspricht der Visualisierung der Kanten des graphentheoretischen Baums in Form von Zylindern und simuliert dadurch vereinfacht die Aststruktur eines biologischen Baumes. [RLP07, S.2]

#### 4.4.1 Procedural Mesh Component

Die Procedural Mesh Component ist eine Komponente der Unreal Engine, welche die Darstellung von prozedural generierten Polygonnetzen zulässt. Ein Vertex ist ein Punkt in einem Polygonnetz mit zur Visualisierung benötigten Informationen. Der Komponente werden Vertexdaten in Form von Listen aus Positions-, Normalen-, Tangenten-, Textur- und Indexdaten übergeben. Das Grafiksystem der Unreal Engine ist daraufhin in der Lage, die Komponente als dreidimensionales Modell im Level darzustellen. [Proa] Die Vertexdaten werden, basierend auf dem übergebenen Baum und den Parametern, vom Modellgenerierungssystem erstellt.

Jedem L-System-Actor und Space-Colonization-Actor ist eine Procedural Mesh Component zugeordnet.

#### 4.4.2 Parameter

Dem Modellgenerierungssystem werden die folgenden Parameter über das Editor-UI übergeben:

**Radius-Daten:** Diese beinhalten den Blattradius, den Radiuswachstumswert, den Stammbreitenmultiplikator und die Abfrage, ob Radiusberechnungen durchgeführt werden sollen.

**Genauigkeit:** Diese beinhalten die minimale und maximale Anzahl von Zylindersektionen sowie den Kurvenreduktionswert.

**Sonstiges:** Weiterhin wird ein Startrotationswinkel, ein Material und eine Abfrage, ob ein fraktales Mesh erstellt werden soll, übergeben. Ein Material beinhaltet Textur- und Shaderinformationen und ist für die Oberflächenbeschaffenheit des generierten Modells verantwortlich.

#### 4.4.3 Operationen auf dem Baum

Folgende Operationen werden vor Beginn der Modelldatengenerierung auf dem graphentheoretischen Baum durchgeführt:

**Kurvenreduktion:** Die Kurvenreduktion wird, beginnend mit dem Wurzel-Astsegment des Baums, rekursiv ausgeführt. In jedem Schritt wird überprüft ob das aktuelle Astsegment entsprechend der Beschreibung in Paragraph 3.5 entfernt werden kann. Falls nicht, wird die Kurvenreduktion auf allen Nachfolgern des aktuellen Astsegment durchgeführt. Die Rekursion bricht ab, falls ein Astsegment keine Nachfolger besitzt.

Der Parameter „Kurvenreduktionswert“ entspricht dem Maximalwert des Skalarprodukts  $\max_K$ .

**Radiusberechnung:** Der Endradius jedes Astsegments wird, beginnend mit dem Wurzel-Astsegment des Baums, rekursiv anhand von Gleichung 3.6 berechnet. Der Parameter „Blattradius“ entspricht  $r_0$  und der „Radiuswachstumswert“ entspricht  $g$ .

Der Startradius jedes Astsegments wird auf den Wert des Endradius seines Vorgängers gesetzt. Da das Wurzel-Astsegment keinen Vorgänger besitzt, wird der Startradius aus der Multiplikation des Stammbreitenmultiplikators mit dem Endradius des Objekts bestimmt.

**Normalenberechnung:** Normalen werden für die Berechnung der Modelldaten benötigt. Die Endnormale  $\vec{n}_e$  jedes Astsegments wird mithilfe der Startposition  $\vec{p}_s$  und Endposition  $\vec{p}_e$  sowie seiner Startnormale  $\vec{n}_s$  wie folgt berechnet:

$$\vec{n}_e = \frac{(\vec{p}_e - \vec{p}_s) + \vec{n}_s}{\|(\vec{p}_e - \vec{p}_s) + \vec{n}_s\|} \quad (4.1)$$

Die Startnormale entspricht der Endnormale des Vorgängers, im Falle des Wurzel-Astsegments entspricht die Startnormale  $\vec{n}_s = \vec{p}_e - \vec{p}_s$ .

**Verringerung der Abzweigungswinkel:** Die Nachfolger jedes Astsegments werden, entsprechend der Beschreibung in Paragraph 3.4, einander angenähert. Die Verringerung der Abzweigungswinkel wird nur bei durch Space-Colonization-Actors generierten Bäumen durchgeführt.

#### 4.4.4 Generierung der Zylinder-Meshes

Moderne Grafik-APIs stellen dreidimensionale Modelle in Form von geometrischen Primitiven dar. Dem Unreal-Engine-Grafiksystem werden Primitive in Form von Dreiecken übergeben. Je drei Vertizes bilden ein Dreieck, dessen Oberfläche mithilfe eines übergebenen Materials gefärbt wird.

Die Vertizes bestehen zusätzlich zu ihren Positionen aus :

**Normalenvektoren** Stellen die Richtung dar, von der aus das Dreieck sichtbar ist und werden unter anderem für Beleuchtungsberechnungen benötigt. [Sura]

**Tangentenvektoren** Stehen orthogonal zu den Normalenvektoren und werden unter anderem für erweiterte Beleuchtungsberechnungen verwendet. [Sura]

Mithilfe dieser Informationen werden zusätzlich Texturkoordinaten, welche für das korrekte Auftragen von Texturen auf das Modell benötigt werden, und Indexdaten, welche für die Darstellung des Polygonnetzes benötigt werden, berechnet.

Der Mantel eines Zylinders kann nun durch die Verbindung von zwei Kreisen generiert werden.

### Berechnung der Vertizes auf einem Kreis

Da ein Kreis theoretisch aus unendlich vielen Punkten besteht, muss eine gewisse Genauigkeit bei der Darstellung von runden Modellen festgelegt werden – ein Kreis wird aus einer zuvor definierten Anzahl von Segmenten generiert. Mithilfe eines Mittelpunkts  $\vec{c}$  und eines Radius  $r$  kann ein Kreis im zweidimensionalen Raum beschrieben werden. Die Vertexpositionen  $\vec{v}_i$  werden wie folgt berechnet:

$$\vec{v}_i = \vec{c} + r * \begin{pmatrix} \cos(d) \\ \sin(d) \\ 0 \end{pmatrix} \text{ mit } d = rot_z + i * \frac{360^\circ}{n} \text{ und } i = 0 \dots (n-1) \quad (4.2)$$

[Surb]

wobei  $n$  der Anzahl der Segmente und  $rot_z$  einem Startrotationswinkel entspricht.

Mithilfe der Kreisnormalen  $\vec{n}_k$ , die orthogonal zur Kreisebene steht und des Kreismittelpunkts  $\vec{c}$  kann die Kreisebene beschrieben werden, auf welcher die Vertizes zu generieren sind. Eine Vertexposition  $\vec{v}_i$  muss um den Kreismittelpunkt rotiert werden, um auf der Kreisebene zu liegen, wobei Rotationsachse  $\vec{R}$  und Rotationswinkel  $\alpha$  wie folgt berechnet werden:

$$\vec{R} = \frac{(\vec{v}_i - \vec{c}) \times \vec{n}_k}{\|(\vec{v}_i - \vec{c}) \times \vec{n}_k\|} \quad (4.3)$$

$$\alpha = \arccos(\langle \vec{n}_k, \vec{z} \rangle) \text{ mit } \vec{z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (4.4)$$

wobei  $\arccos$  dem Arkuskosinus entspricht. [Bak] Die rotierte Vertexposition  $\vec{v}$  ermöglicht die Berechnung der Vertexnormalen  $\vec{n}_v$  und Vertextangente  $\vec{t}_v$ :

$$\vec{n}_v = \frac{\vec{v} - \vec{c}}{\|\vec{v} - \vec{c}\|} \text{ und } \vec{t}_v = \vec{n}_v \times \vec{n}_k \quad (4.5)$$

### Verbindung der Kreise

Um den Zylindermantel eines Astsegments zu bilden, werden zwei Kreise mithilfe der Start- und Enddaten des Objekts generiert. Jedes Segment eines Kreises wird mit dem entsprechenden Segment des anderen Kreises verbunden und bildet dadurch ein Zylindersegment. Ein Zylindersegment entspricht somit einem Rechteck. Da das Modell jedoch Dreiecksdaten benötigt, wird jedes Rechteck, wie in Abbildung 4.3a dargestellt, aus zwei Dreiecken gebildet. [Surb]

Wird jedes Astsegment durch die Verbindung von genau zwei Kreisen dargestellt, führt dies zu der Generierung redundanter Vertexdaten. Die Enddaten eines Astsegments und die Startdaten seines Nachfolgers entsprechen einander, da

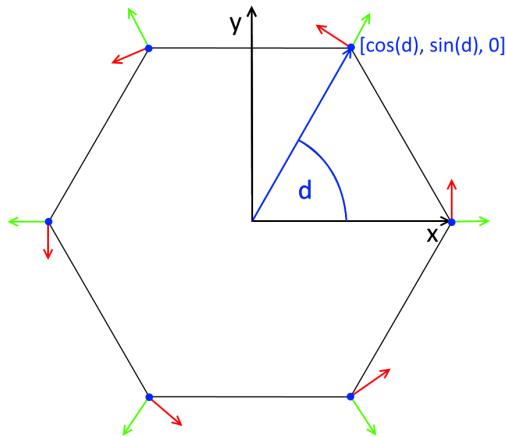


Abb. 4.2: Beispiel für die Berechnung von Vertexpositionen auf einem Einheitskreis mit einer Genauigkeit von sechs Segmenten.  $d = \frac{360^\circ}{6} = 60^\circ$ . Die blauen Punkte entsprechen den Positionen, die grünen Pfeile den Normalen und die roten Pfeile den Tangenten der Vertizes. Eigene Abbildung.

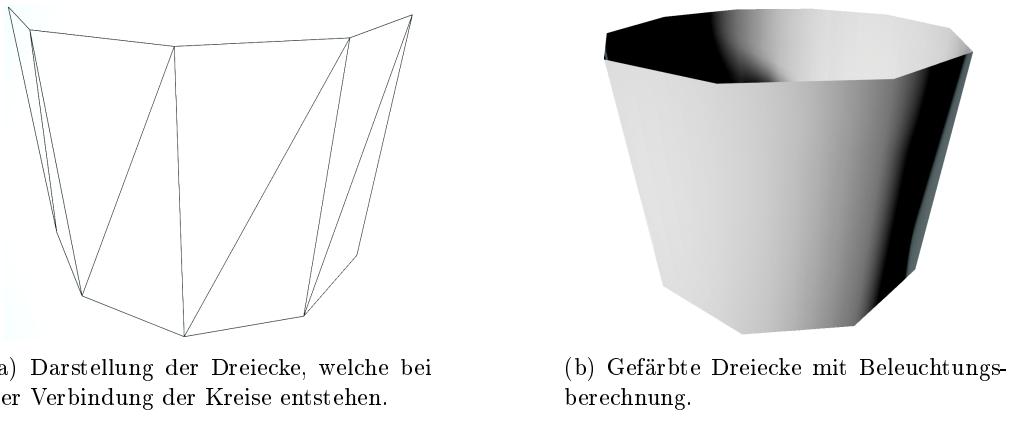
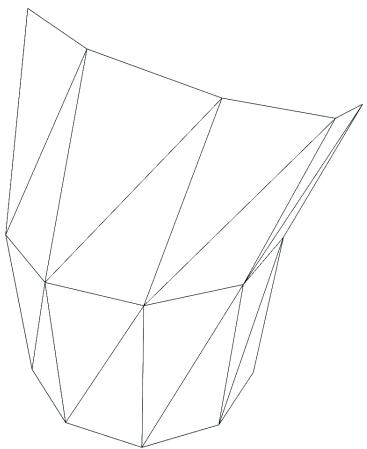


Abb. 4.3: Verbindung zweier Kreise zu einem Zylindermantel. Eigene Abbildungen.

die daraus generierten Kreis-Vertizes genau aufeinander liegen. Anstatt nun vier Kreise für die Generierung zweier Zylinder zu verwenden, können die Vertizes des verbindenden Kreises wiederverwendet werden – somit sind drei Kreise für die Generierung zweier Zylinder ausreichend.

Die Verbindung der Modelldaten kann für alle Astsegmente durchgeführt werden, deren Nachfolger dieselbe Zweigtiefe besitzen und somit eine Folge von zusammenhängenden Zylindermodellen bilden. Für einen Nachfolger mit einer sich unterscheidenden Zweigtiefe wird eine neue Folge von zusammenhängenden Zylindermodellen begonnen. [Surb]

Ein Beispiel für die Generierung zweier Zylinder mithilfe von drei Kreisen wird in Abbildung 4.4a dargestellt.



(a) Darstellung der Dreiecke, welche bei der Verbindung der Kreise entstehen.



(b) Gefärbte Dreiecke mit Beleuchtungsberechnung.

Abb. 4.4: Verbindung dreier Kreise zu einer Folge zusammenhängender Zylindermodelle. Eigene Abbildungen.

# 5

---

## Ergebnisse

In diesem Kapitel werden die Ergebnisse vorgestellt, die mithilfe der Implementierungen von L-Systemen und dem Space-Colonization Algorithmus produziert wurden. Es bietet sich an in vielen Fällen biologisch motivierte Begriffe zu verwenden – beispielsweise Kanten als Äste oder Zweige zu bezeichnen – um eine Verbindung zwischen visueller Repräsentation und graphentheoretischem Hintergrund zu schaffen.

### 5.1 L-System-Actor

L-Systeme ermöglichen es mithilfe bestimmter Produktionsregeln realitätsnahe Baumstrukturen zu generieren. Im Folgenden werden verschiedene natürliche Wachstumsarten mithilfe der L-System Implementierung nachgeahmt und visualisiert.

#### 5.1.1 Monopodiales Wachstum

Ein biologischer Baum mit monopodialen Wachstumsverhalten bildet einen Hauptstamm, der stets weiterwächst, mit davon abzweigenden Nebenästen. [DL05, S.14] Ein Beispiel für ein solches Wachstum ist das folgende L-System:

$$\begin{aligned} \omega &: F(s)A(100) \\ p_1 : A(l) &\rightarrow F(l) [&(a1) B(l)] /(d1) A(l * r1) \\ p_2 : B(l) &\rightarrow F(l) [-(a2) B(l * r2)] /(d2) B(l * r2) \end{aligned} \quad (5.1)$$

[PL90, S.56]

Die Produktionsregel  $p_1$  produziert den Hauptstamm, welcher in jeder Ableitung verlängert wird und einen Nebenast produziert, der anhand von  $p_2$  ebenfalls entlang einer Hauptachse weiterwächst und weitere Nebenäste produziert.  $a_1$  und  $a_2$  entsprechen den Abzweigungswinkeln neuer Äste von der Hauptachse, während die Winkel  $d_1$  und  $d_2$  die Rotation um die Hauptachse beschreiben, bevor ein neuer Nebenast produziert wird.  $r_1$  und  $r_2$  entsprechen Faktoren, welche das Wachstum eines Astes pro Ableitung verkürzen, falls diese Werten unter 1 entsprechen, und verlängern, falls Werte über 1 vorliegen. Die Variable  $s$  beschreibt die Länge des Stammes bevor das Wachstum beginnt. [PL90, S.57]

Abbildung 5.1 zeigt Beispiele für monopodiale Baumstrukturen, Tabelle 5.1 die verwendeten Konstantenwerte. Es findet kein Einfluss durch Tropismus statt.

| Abbildung | $n$ | $a_1$ | $a_2$ | $r_1$ | $r_2$ | $d_1$ | $d_2$ | $s$   |
|-----------|-----|-------|-------|-------|-------|-------|-------|-------|
| 5.1a      | 11  | 45.0  | 35.0  | 1.05  | 0.9   | 137.5 | 70.0  | 100.0 |
| 5.1b      | 11  | 60.0  | 20.0  | 1.05  | 0.9   | 137.5 | 70.0  | 100.0 |
| 5.1c      | 11  | 45.0  | 35.0  | 0.95  | 0.9   | 137.5 | 70.0  | 350.0 |
| 5.1d      | 11  | 77.0  | -37.0 | 1.05  | 0.85  | 137.5 | -70.0 | 200.0 |

Tabelle 5.1: Konstantenwerte der in Abbildung 5.1 dargestellten monopodialen Baumstrukturen.

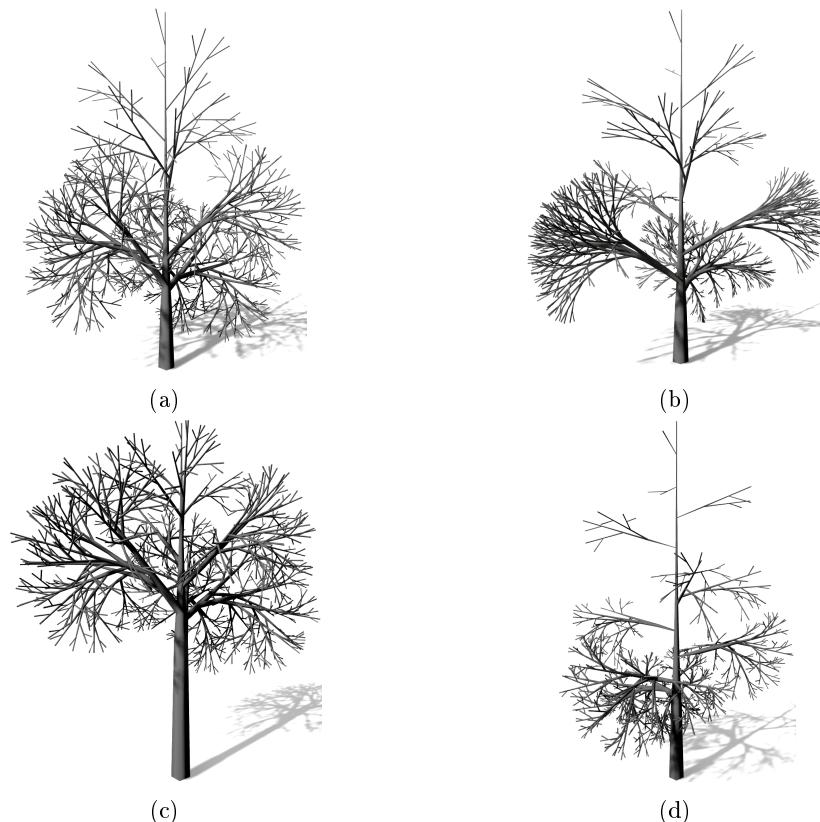


Abb. 5.1: Beispiele monopodialen Wachstums, entspricht der  $n$ -fachen Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 5.1. Eigene Abbildungen.

### 5.1.2 Sympodiales Wachstum

Beim sympodialen Wachstum bildet ein Ast mehrere Abzweigungen in einem Winkel ungleich  $0^\circ$ . Der sich verzweigende Ast wächst daraufhin nicht weiter. [DL05, S.14] [PL90, S.58] Das folgende L-System simuliert ein solches Wachstum:

$$\begin{aligned} \omega &: F(100)A(150) \\ p_1 : A(l) &\rightarrow F(l) [ \&(a1) B(l * r1) ] / (180) [ \&(a2) B(l * r2) ] \\ p_2 : B(l) &\rightarrow F(l) [ +(a1) / (d1) B(l * r1) ] [ -(a2) \backslash (d2) B(l * r2) ] \end{aligned} \quad (5.2)$$

[PL90, S.59]

Die Produktionsregel  $p_1$  beschreibt die Bildung der ersten, sich gegenüberliegenden, Abzweigungen während  $p_2$  das Wachstum der abzweigenden Äste beschreibt, welche sich ebenfalls zu zwei Nebenästen aufteilen.  $a_1$  und  $a_2$  entsprechen den Abzweigungswinkeln,  $r_1$  und  $r_2$  den Wachstumsfaktoren pro Ableitung der Nebenäste.  $d_1$  und  $d_2$  beschreiben die Rotation eines Nebenastes um seine eigene Achse, bevor die nächste Abzweigung entsteht.

Abbildung 5.2 zeigt Beispiele für sympodiale Baumstrukturen, Tabelle 5.2 die verwendeten Konstantenwerte. Es findet kein Einfluss durch Tropismus statt.

| Abbildung | $n$ | $a_1$ | $a_2$ | $r_1$ | $r_2$ | $d_1$ | $d_2$ |
|-----------|-----|-------|-------|-------|-------|-------|-------|
| 5.2a      | 10  | 5.0   | 65.0  | 0.9   | 0.75  | 50.0  | 50.0  |
| 5.2b      | 10  | 20.0  | 50.0  | 0.9   | 0.75  | 50.0  | 50.0  |
| 5.2c      | 10  | 35.0  | 35.0  | 1.0   | 0.8   | 50.0  | 50.0  |
| 5.2d      | 10  | 30.0  | 40.0  | 0.95  | 0.9   | 110.0 | 110.0 |

Tabelle 5.2: Konstantenwerte der in Abbildung 5.2 dargestellten sympodialen Baumstrukturen.

### 5.1.3 Ternäre Verzweigungen

Eine andere Herangehensweise zur Bestimmung von L-Systemen für realistisch wirkende Baumstrukturen ist die Beschreibung von Selbstähnlichkeit, wie sie oft in der Natur zu finden ist. [PL90, S.173] Das L-System aus Gleichung 5.3 entspricht einem Wachstum, in welchem drei Abzweigungen mithilfe von Produktion  $p_1$  auf dieselbe Art erstellt werden. In Produktion  $p_2$  wird beschrieben wie Kanten aus vorhergehenden Ableitungen um einen Faktor  $l_r$  verlängert werden. [PL90, S.58]

$$\begin{aligned} \omega &: /(45)A \\ p_1 : A &\rightarrow F(l_s) [ \&(a) F(l_s) A ] / (d1) [ \&(a) F(l_s) A ] / (d2) [ \&(a) F(l_s) A ] \\ p_2 : F(l) &\rightarrow F(l * l_r) \end{aligned} \quad (5.3)$$

[PL90, S.60]

$a$  entspricht dem Abzweigungswinkel von der Ursprungsachse,  $d_1$  und  $d_2$  der Rotation um die Achse, bevor eine neue Abzweigung produziert wird.  $l_s$  ist die Anfangslänge der Kanten. [PL90, S.58]

Abbildung 5.3 zeigt verschiedene ternär verzweigte Baumstrukturen, die auf den in Tabelle 5.3 gelisteten Konstantenwerten aufgebaut sind.

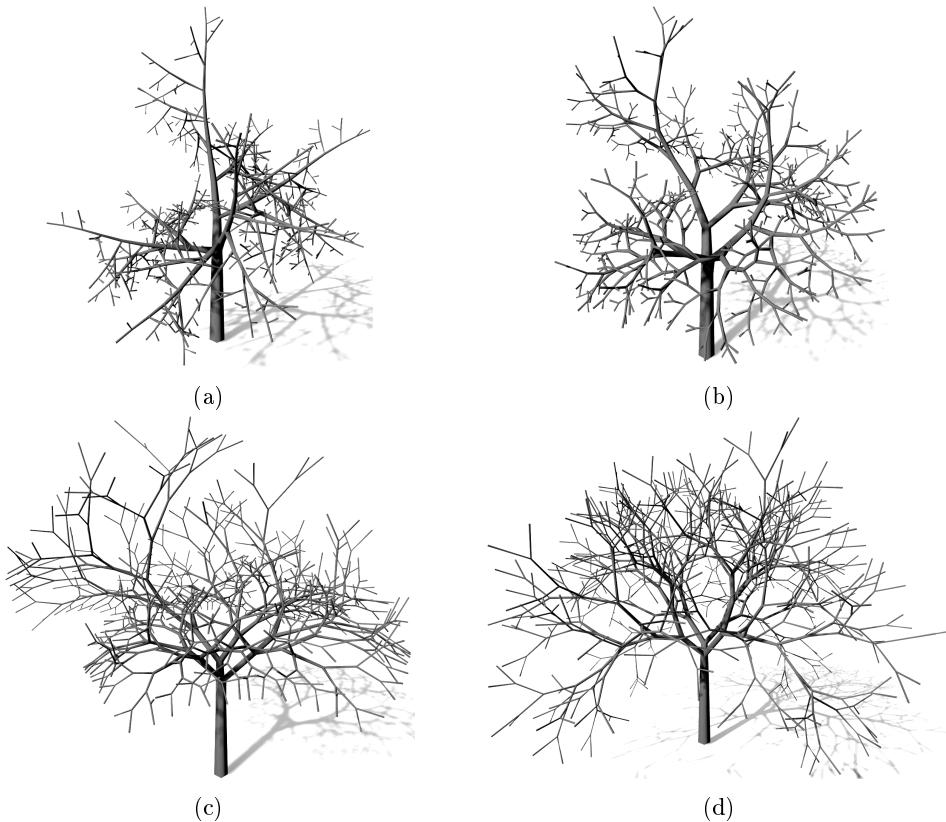


Abb. 5.2: Beispiele sympodialen Wachstums, entspricht der n-fachen Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 5.2. Eigene Abbildungen.

| Abbildung | $n$ | $a$  | $d_1$ | $d_2$ | $l_r$ | $l_s$ |
|-----------|-----|------|-------|-------|-------|-------|
| 5.3a      | 6   | 19.0 | 95.0  | 132.0 | 1.1   | 50.0  |
| 5.3b      | 8   | 19.0 | 137.5 | 137.5 | 1.2   | 50.0  |
| 5.3c      | 8   | 27.0 | 77.0  | 77.0  | 1.3   | 50.0  |
| 5.3d      | 8   | 31.0 | 120.0 | 190.0 | 1.2   | 50.0  |

Tabelle 5.3: Konstantenwerte der in Abbildung 5.3 dargestellten ternär verzweigten Baumstrukturen.

#### 5.1.4 Tropismus

Der Einfluss von Tropismus führt, bei Eingabe derselben Werte bei den restlichen Parametern, zu visuell unterschiedlichen Ergebnissen. Dadurch können reale Einflüsse wie die Einwirkung von Wind, Sonne und Gravitation simuliert sowie die Wiederverwendbarkeit von L-System Definitionen erhöht werden. [PL90, S.58]

Abbildung 5.4 zeigt den Einfluss von Tropismus auf Baumstrukturen aus Abbildung 5.3, welche ohne die Einwirkung von Tropismus produziert wurden.

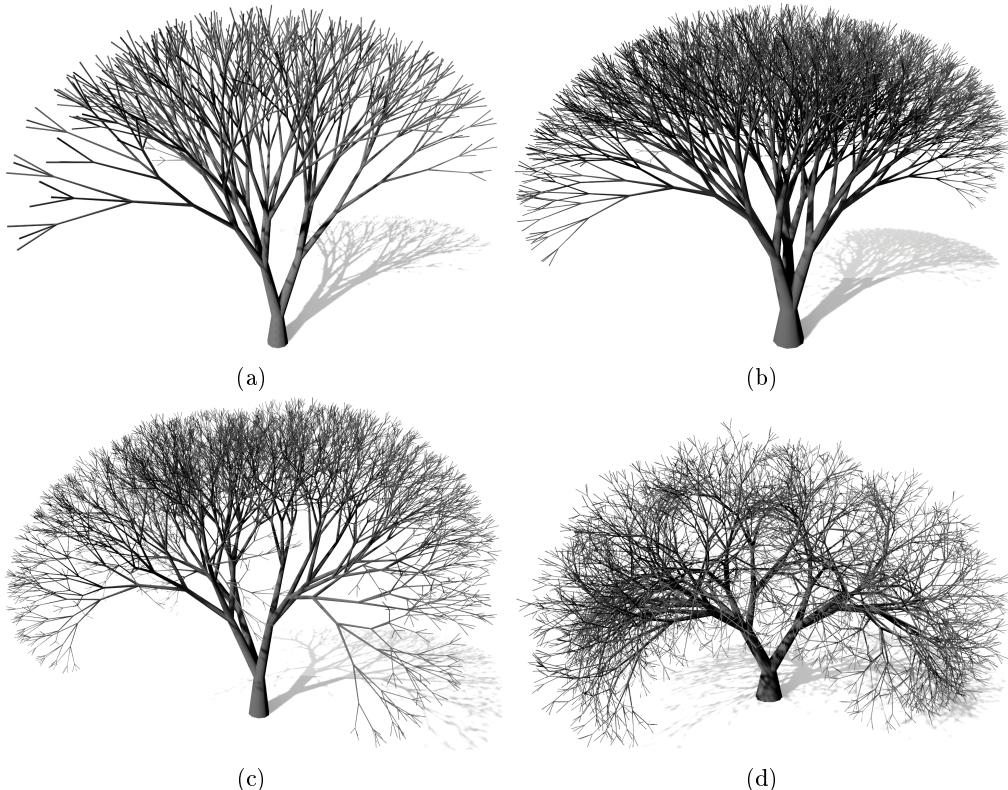


Abb. 5.3: Beispiele ternären Wachstums, entspricht der n-fachen Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 5.3. Eigene Abbildungen.

## 5.2 Space-Colonization-Actor

Die Space-Colonization Implementierung ermöglicht es durch die Angabe einiger weniger Parameter eine Vielzahl verschiedener Baumstrukturen zu generieren. [RLP07, S.3] Im Folgenden wird der Einfluss dieser Parameter auf die Ergebnisse untersucht. Es werden jeweils sich unterscheidende Parameterwerte angegeben.

### 5.2.1 Einflussbereiche

Die Verteilung und Anzahl der Einflusspunkte hat starke Auswirkungen auf die sich ergebende Baumstruktur. Je weniger Einflusspunkte existieren, desto größer ist die Wirkung jedes Punktes auf die Position der neuen Knotenpunkte und resultiert in einer spärlich bewachsenen Baumkrone mit kantigen, ungleichmäßig verteilten Ästen. Wird die Anzahl der Einflusspunkte erhöht, entsteht eine dicht bewachsene Baumkrone mit leicht kurvenförmigen, gleichmäßig verteilten Ästen. [RLP07, S.3]

Weiterhin passt sich die generierte Baumstruktur dem Bereich an, in welchem die Einflusspunkte generiert werden. Es können entweder einzelne Bereichsprimitive, wie eine Kugel- oder Zylinderform, oder mehrere miteinander verknüpfte Einflussbereiche verwendet werden. [RLP07, S.4]

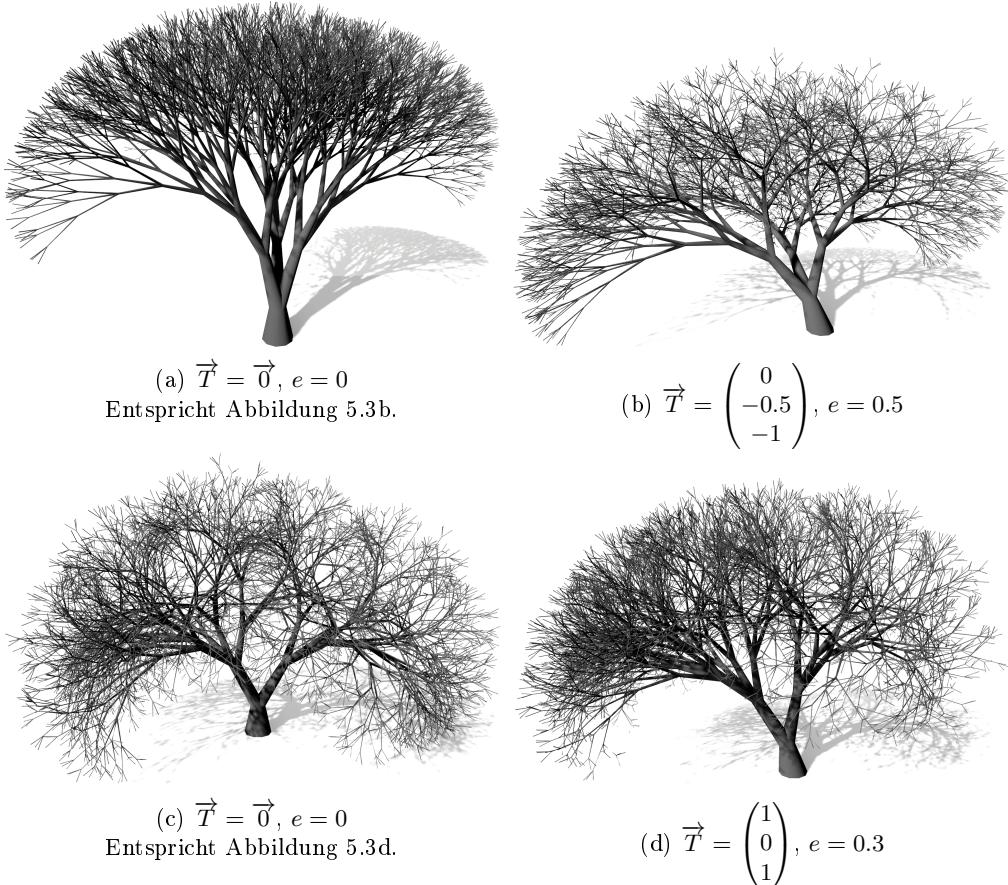


Abb. 5.4: Beispiele ternären Wachstums mit und ohne Einfluss durch Tropismus. Entspricht der n-fachen Ableitung des Axioms anhand der Produktionsregeln aus Gleichung 5.3. Die linken und rechten Abbildungen unterscheiden sich jeweils im Tropismusvektor  $\vec{T}$  und im Biegsamkeitsfaktor  $e$ . Eigene Abbildungen.

### 5.2.2 Wachstumsparameter

Bei den Parametern, welche sich direkt auf die Positionierung neuer Knoten auswirken, handelt es sich um Minimalradius, Einflussradius, Schrittweite, Tropismus, Maximale Zweigtiefe, gewichtetes Wachstum, den maximalen Grad eines Knotens und die maximale Anzahl von durchgeföhrten Iterationen.

#### *Minimalradius und Einflussradius*

Der Minimalradius  $d_k$  hat eine ähnliche Wirkung auf eine generierte Baumstruktur wie die Anzahl der Einflusspunkte  $N$ . Je höher der Minimalradius  $d_k$ , desto weniger Abzweigungen werden gebildet, da nach der Bildung eines neuen Astsegments Einflusspunkte entfernt werden, die zuvor eine solche Abzweigung bewirkt hätten. Die Baumkrone erscheint spärlicher bewachsen, die entstandenen Äste sind jedoch kurvenreicher, da auf die wenigen wachsenden Äste mehr Einflusspunkte einwirken – das Entfernen oder Hinzufügen einzelner Einflusspunkte hat somit nur eine kleine

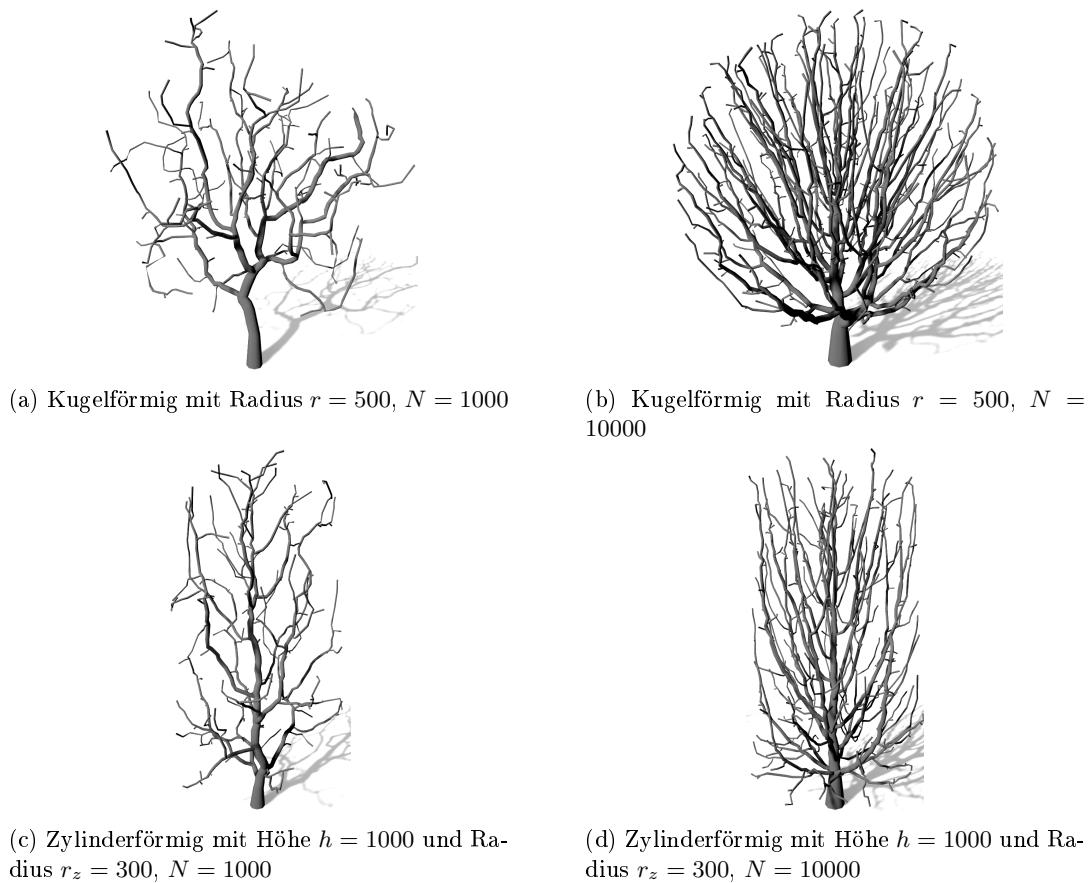


Abb. 5.5: Wirkung der Anzahl der Einflusspunkte  $N$  und der Form des Einflussbereichs auf die generierte Baumstruktur. Eigene Abbildungen.

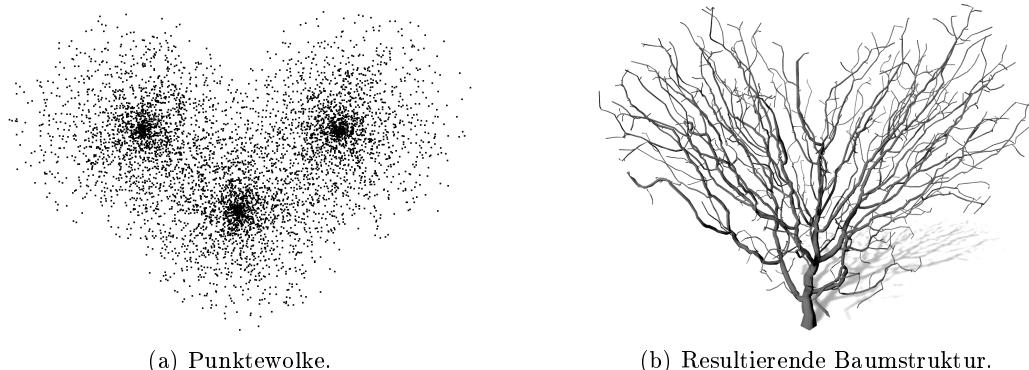


Abb. 5.6: Verbindung von drei kugelförmigen Einflussbereichen zu einem Bereich. Die Kugeln haben jeweils den Radius  $r = 500$  und  $N = 2000$ . Eigene Abbildungen.

Auswirkung auf die Position eines in der nächsten Iteration neu hinzugefügten Astsegments. [RLP07, S.3]

Je kleiner der Einflussradius  $d_i$ , desto knorriger wirkt ein Baum, da Astsegmente in jedem Iterationsschritt in den Wirkungsbereich eines Einflusspunktes geraten und diesen im nächsten Schritt wieder verlassen. [RLP07, S.4]

Abbildung 5.7 zeigt Beispiele der Auswirkung unterschiedlicher Werte für Einflussradius und Minimalradius.

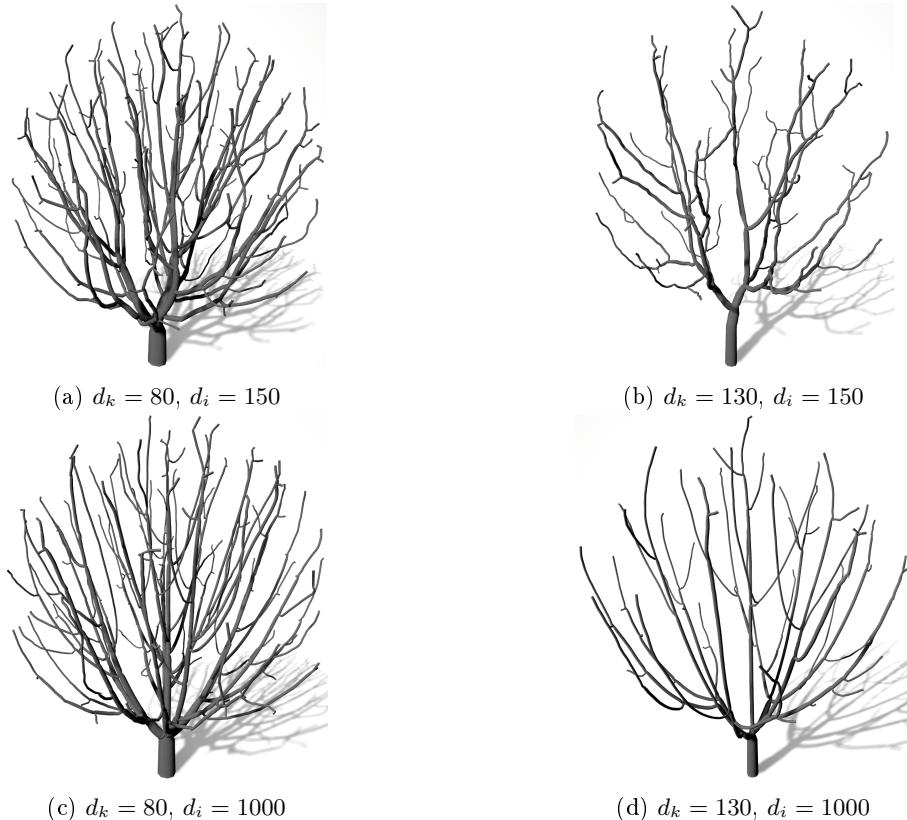


Abb. 5.7: Wirkung von Einflussradius  $d_i$  und Minimalradius  $d_k$  auf die generierte Baumstruktur. Es wurde die Schrittweite  $D = 10$  verwendet. Eigene Abbildungen.

### *Tropismus, maximale Zweigtiefe, maximaler Grad und gewichtetes Wachstum*

Ebenso wie bei der Implementierung von L-Systemen kann der Einfluss eines Tropismusvektors  $\vec{T}$  das Wachstumsverhalten der Space-Colonization Implementierung bei gleichen Parameterwerten stark in eine bestimmte Richtung beeinflussen. Werte auf der horizontalen Ebene erwecken den Anschein von Einwirkung durch Wind. Positive Werte auf der Z-Achse entsprechen dem Streben einer Pflanze der Sonnenposition entgegen zu wachsen, negative Werte auf der Z-Achse simulieren die Einwirkung auf das Wachstum durch Gravitation. [RLP07, S.5]

Die maximale Zweigtiefe  $max_Z$  begrenzt die Anzahl der Abzweigungen, die von dem Wurzel-Astsegment ausgehend zu jedem Astsegment ohne Nachfolger

gebildet werden können. Bei niedrigen Werten führt dies zur Bildung weniger, kurvenförmiger Astsegmente.

Der maximale Grad  $\max_{grad}$  begrenzt die Anzahl der Abzweigungen, die von einem einzelnen Astsegment ausgehen können. Da Astsegmente nur in sehr dichten Baumstrukturen – Strukturen mit vielen Einflusspunkten und einem niedrigen Minimalradius – mehr als zwei Nachfolger bilden, hat dieser Parameter eine eingeschränkte visuelle Einwirkung. In dichten Baumstrukturen ermöglicht er es jedoch, bei minimaler visueller Einwirkung, die Anzahl der erstellten Vertizes zu begrenzen. Die Wahl von  $\max_{grad} = 1$  sorgt für die Bildung eines durchgängigen, in sich verdrehten Astes ohne Abzweigungen.

In Abbildung 5.8 werden Beispiele für die Wirkung der beschriebenen Parameter gezeigt.

#### *Schrittweite und maximale Anzahl durchzuführender Iterationen*

Je höher die Schrittweite  $D$ , desto knorriger wirkt ein Baum – ähnlich dem Einflussradius – da ein Astsegment in jedem Schritt Einflussradien betritt und verlässt oder ein neues Astsegment auf der gegenüberliegenden Seite eines Einflusspunktes gebildet wird und von diesem nun aus der entgegengesetzten Richtung beeinflusst wird. Weiterhin besitzen die Äste scharfe Kanten, da aufgrund der größeren Schrittweite weniger Vertizes generiert werden. Die Wahl einer niedrigen Schrittweite führt zur Bildung von Ästen mit leichten Kurven.

Die maximale Anzahl durchzuführender Iterationen  $N_I$  beschränkt die Zahl von Space-Colonization Iterationen, die durchgeführt werden sollen. Die bisher gezeigten Beispiele wurden mit  $N_I = 400$  durchgeführt und vor Erreichen dieser Zahl abgebrochen. Bei Festlegung niedrigerer Zahlen kann das Wachstum des Baumes beschränkt oder das Wachstumsverhalten im Laufe der Iterationen beobachtet werden.

Abbildung 5.9 zeigt die Auswirkungen verschiedener Werte für die Schrittweite und die maximale Anzahl durchzuführender Iterationen bei gleichbleibender Verteilung der Einflusspunkte.

#### *Gewichtetes Wachstum*

Gewichtetes Wachstum erhöht die Wachstums-Schrittweite der Astsegmente in Abhängigkeit von der Zweigtiefe. Dies führt zu der erkennbaren Bildung eines Stammes und den davon abzweigenden Hauptästen.

Die Einstellung erzielt den besten visuellen Eindruck in bestimmten Situationen: Bei Anwendung des Algorithmus auf große Werte für den Radius des Einflussbereichs  $r$ , der Anzahl von Einflusspunkten  $N$ , dem Einflussradius  $d_i$  und bei einer Begrenzung des maximalen Anzahl durchzuführender Iterationen  $N_I$ , sodass die Baumstruktur nur einen Teil des Einflussbereichs ausfüllen kann. Dies erweckt den Anschein eines Baumes, dessen Wachstum noch nicht beendet ist.

Abbildung 5.10 zeigt ein Beispiel für die Anwendung von gewichtetem Wachstum in der beschriebenen Situation.

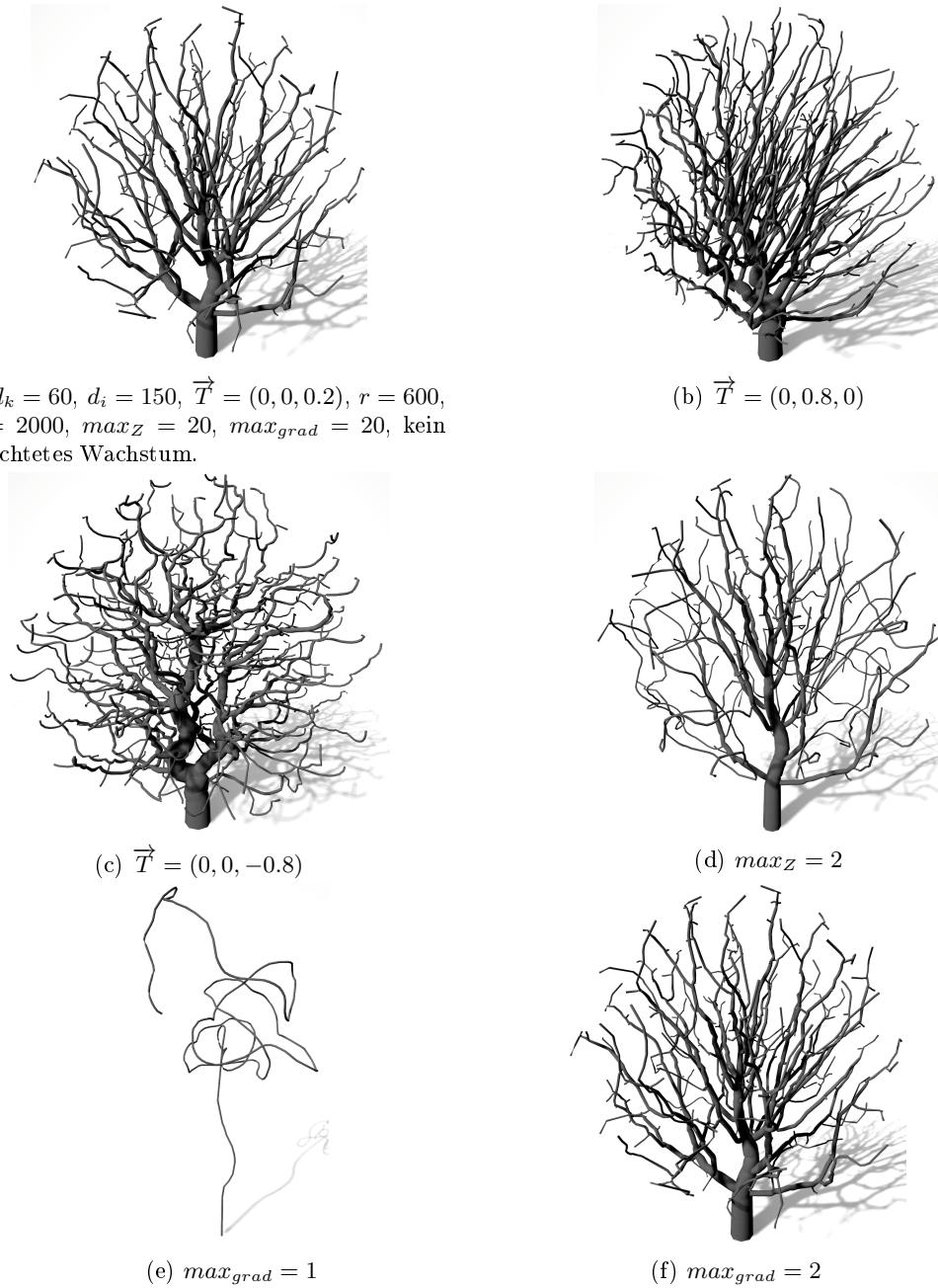


Abb. 5.8: Wirkung von Tropismus  $\vec{T}$ , maximaler Zweigtiefe  $max_Z$  und maximalem Knotengrad  $max_{grad}$  auf die generierte Baumstruktur. Falls nicht weiter angegeben, wurden die in Abbildung 5.8a angegebenen Parameterwerte verwendet. Eigene Abbildungen.

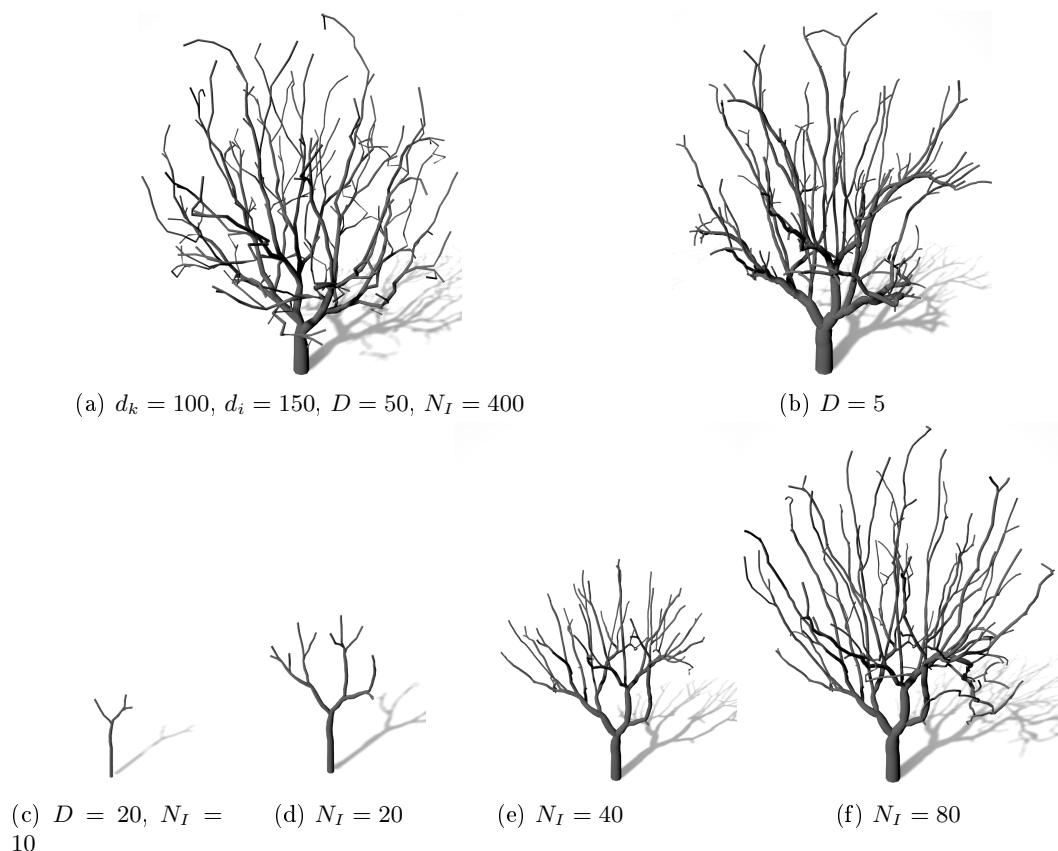


Abb. 5.9: Wirkung von Schrittweite  $D$  und maximaler Anzahl von Iterationen  $N_I$  auf die generierte Baumstruktur. Eigene Abbildungen.



Abb. 5.10: Wirkung von gewichtetem Wachstum auf die generierte Baumstruktur. Eigene Abbildungen.

## 5.3 Effizienz

Die Effizienz einer Software ist ihre Fähigkeit, mit den durch das Betriebssystem bereitgestellten Ressourcen ein bestimmtes Leistungsniveau zu erzielen. [Bal98, S.259] In Hinsicht auf prozedurale Generierung muss auf zwei wesentliche Punkte geachtet werden: Die Generierungszeit und das Laufzeitverhalten.

### 5.3.1 Generierungszeit

Die Zeit welche benötigt wird um eine Baumstruktur aufzubauen wird als Generierungszeit bezeichnet und bei den Implementierungen durch die übergebenen Parameter beeinflusst. Mithilfe des Unreal Engine Profilers wurde die Ausführungszeit bestimmter Funktionen oder Funktionsabschnitte gemessen um die Abhängigkeiten zwischen Parametern und Generierungszeit zu bestimmen. [Prob]

#### *L-System-Actor*

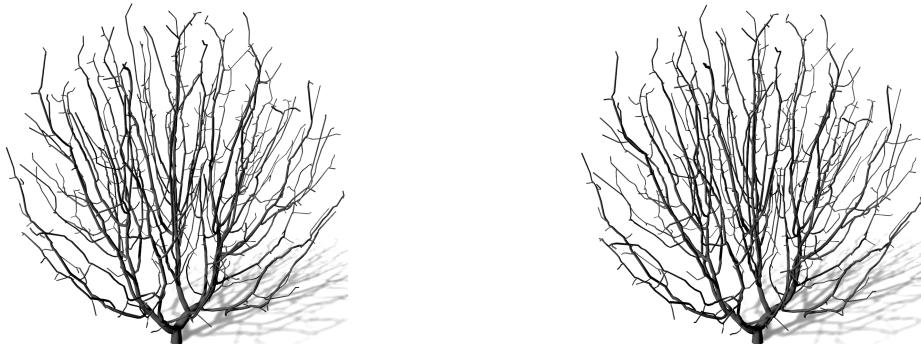
Die Generierungszeit der L-System Implementierung hängt linear von den durch die Turtle-Interpretation ausgeführten Bewegungen und Rotationen ab und macht diese somit abhängig von der Anzahl der Ableitungen sowie den angegebenen Produktionsregeln. Je öfter ein Axiom abgeleitet wird und je mehr Rotations- und Bewegungsaktionen in den Nachfolgern der Produktionsregeln angegeben sind, desto mehr Aktionen enthält die zu interpretierende Zeichenkette.

#### *Space-Colonization-Actor*

Die Generierungszeit der Space-Colonization Implementierung wird durch eine Vielzahl von Parametern festgelegt, welche die Anzahl der Einflusspunkte, wachsenden Astsegmente und durchzuführenden Iterationen beeinflussen. In einer Iteration wird jeder Einflusspunkt daraufhin überprüft, ob sich mindestens einer der wachsenden Astsegmente in seinem Einflussradius befindet. Falls dies zutrifft wird zusätzlich die Liste der Astsegmente im Einflussradius daraufhin überprüft, welches Segment den geringsten Abstand besitzt.

Die eingeführte Bedingung  $max_{NG}$  – die maximale Anzahl von Iterationen, in welchen einem Astsegment kein neuer Nachfolger hinzugefügt wurde – begrenzt die Anzahl der aktuell wachsenden Astsegmente und konnte somit erfolgreich zur Verringerung der Generierungszeit eingesetzt werden, ohne die resultierende Baumstruktur visuell erkennbar zu verändern.

Abbildung 5.11 zeigt zwei mit denselben Parametern generierte Space-Colonization Baumstrukturen, lediglich  $max_{NG}$  wurde angepasst. Die Generierungszeit der Baumstruktur aus Abbildung 5.11b beträgt ungefähr das fünffache der Generierungszeit des Modells aus Abbildung 5.11a. Die Messung der Generierungszeit fand unter gleichbleibenden Bedingungen auf einem Rechner mit 3.3GHz Prozessor statt.



(a)  $N = 5000, r = 1000, N_I = 1000, d_k = 100, d_i = 250, D = 20, \vec{T} = (0, 0, 0.2)$ .  
 $\max_{NG} = 2, \text{Generierungszeit} = 1.12s.$

(b)  $\max_{NG} = 1000, \text{Generierungszeit} = 6.08s.$

Abb. 5.11: Einfluss von  $\max_{NG}$  auf die Generierungszeit. Die Messungen der Generierungszeit fanden unter gleichbleibenden Bedingungen statt. Eigene Abbildungen.

### 5.3.2 Laufzeitverhalten

Das Laufzeitverhalten entspricht in diesem Fall der Anzahl von Bildern (engl.: Frames), die pro Sekunde produziert werden können, auch als Bildrate (engl.: Frames per second oder fps) bezeichnet. In Hinsicht auf die generierten Baumstrukturen ist das Laufzeitverhalten abhängig von der dargestellten Geometrie, da eine Grafikkarte lediglich eine begrenzte Menge Vertexdaten pro Sekunde verarbeiten und darstellen kann. Eine Verbesserung des Laufzeitverhaltens wird durch eine Reduktion der Vertexmenge einer generierten Baumstruktur erreicht.

Das Laufzeitverhalten muss stets gegen die Qualität der zu generierenden Modelle abgewägt werden – eine Verringerung der Vertexdaten führt zwar zu einer höheren Bildrate, im Gegenzug jedoch auch zu einer niedrigeren visuellen Modellqualität. [DL05, S.5]

#### *L-System-Actor*

Die Vertexanzahl eines durch ein L-System generierten Baummodells wird durch die Anzahl der Schritte bestimmt, welche von der Turtle-Interpretation durchgeführt werden und ist somit abhängig von der Anzahl an Ableitungen und den definierten Produktionsregeln.

#### *Space-Colonization-Actor*

Die Anzahl von Vertizes, welche durch die Space-Colonization Implementierung generiert werden, hängt von Parametern ab, welche die Anzahl der erstellen Astsegmente beeinflussen. Insbesondere eine Verringerung der Schrittweite und des Minimalradius tragen zu einer Erhöhung der Anzahl an generierten Astsegmenten bei.

### Modellgenerierungssystem

Das Modellgenerierungssystem bietet drei Parameter zur Anpassung der Modellqualität: die minimale und maximale Anzahl von Zylindersektionen sowie den Kurvenreduktionswert, welche sich auf generierte Bäume beider Implementierungen anwenden lassen.

Für jedes Astsegment wird die Anzahl an Zylindersektionen festgelegt, indem, abhängig von der Zweigtiefe des Segments, zwischen der minimalen und maximalen Anzahl  $n_{min}$  und  $n_{max}$  linear interpoliert wird. Das Resultat ist, dass Astsegmente mit einer hohen Zweigtiefe und geringem Radius mit weniger Vertizes dargestellt werden als Astsegmente mit einer niedrigen Zweigtiefe und großem Radius.

Der Kurvenreduktionswert  $max_K$  reduziert die Zahl der Astsegmente, indem Segmente, die in einem geringen Winkel voneinander abstehen, zusammen gefasst werden. Die geringere Astsegmentanzahl führt zu einer Verringerung der Vertexanzahl.

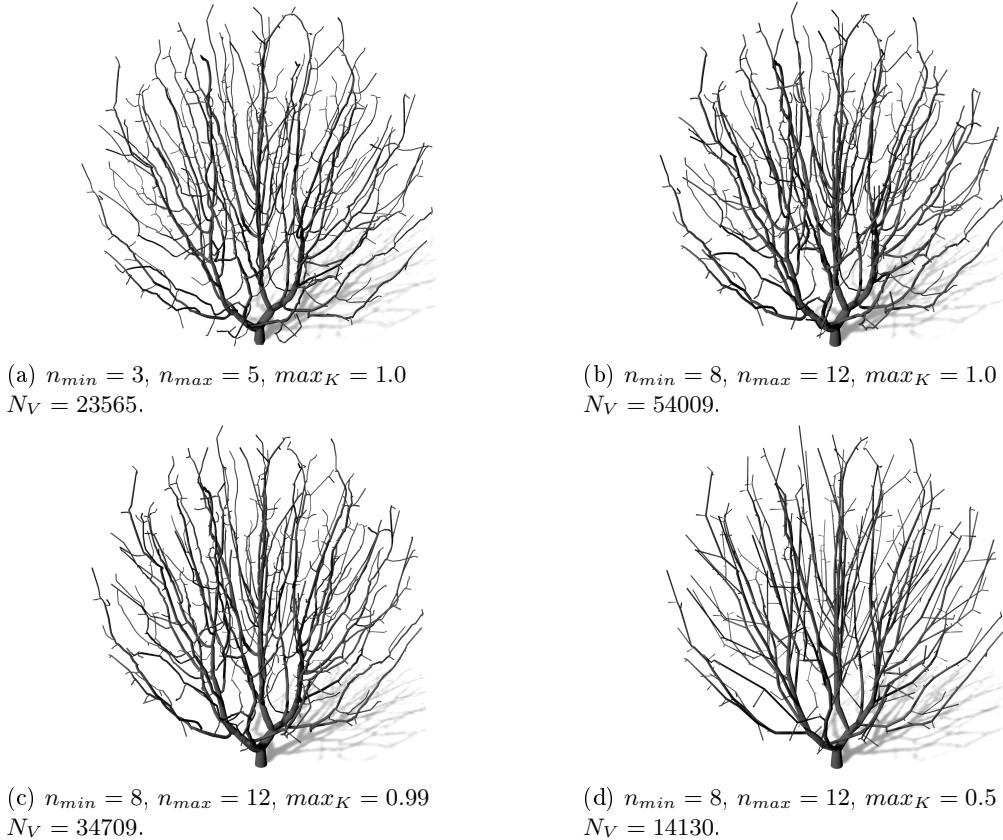


Abb. 5.12: Einfluss der minimalen und maximalen Anzahl an Zylindersektionen  $n_{min}$  und  $n_{max}$  sowie des Kurvenreduktionswerts  $max_K$  auf die Zahl der generierten Vertizes  $N_V$ . Die Parameter für die Generierung des zugrunde liegenden graphentheoretischen Baums stimmen überein. Eigene Abbildungen.

# 6

---

## Zusammenfassung und Ausblick

Es wurden zwei Verfahren für die 3D-Modellgenerierung von Baumstrukturen vorgestellt:

### *Lindenmayer-Systeme*

Lindenmayer-Systeme stellen eine Erweiterung von kontextfreien Grammatiken dar, welche Teile einer übergebenen Zeichenkette anhand festgelegter Regeln durch andere Zeichenketten ersetzen. D0L-Systeme, die fundamentalste Form der L-Systeme, werden zu parametrischen L-System erweitert. [PL90, S.3] Dies ermöglicht die Ableitung von komplexen Zeichenfolgen, welche mithilfe der vorgestellten, grafischen Interpretation von L-System-Resultaten visuell repräsentiert werden. Diese sogenannte Turtle-Interpretation wird für die Verwendung im dreidimensionalen Raum erweitert und die 3D-Modellgenerierung von Baumstrukturen angepasst. [PL90, S.7, 18f, 21ff, 51ff]

### *Space Colonization Algorithmus*

Der vorgestellte Space Colonization Algorithmus basiert auf dem biologisch motivierten Konkurrenzverhalten um Wachstumsraum zwischen wachsenden Ästen eines Baumes. Es werden Punkte in einem festgelegten Bereich generiert, welche die Verfügbarkeit von Wachstumsraum repräsentieren und in jeder Iteration des Algorithmus das Wachstum der Äste beeinflussen. Nähert sich ein Ast einem der Punkte, wird dieser entfernt, um zu signalisieren, dass in diesem Bereich kein Wachstumsraum mehr verfügbar ist. [RLP07, S.2f] Die Verteilung der Einflusspunkte und das Wachstumsverhalten des Baums in jeder Iteration kann durch Übergabe von numerischen und Booleschen Parameterwerten gelenkt werden. [RLP07, S.5]

Weiterhin wurden Erweiterungen des ursprünglichen Algorithmus vorgeschlagen, welche das Wachstumsverhalten beeinflussen, die Generierungszeit verkürzen und zu einer verminderten Datenmenge des generierten 3D-Modells beitragen können.

### *Implementierung und Ergebnisse*

Die innerhalb der Unreal Engine 4 umgesetzte Implementierung und damit verbundene Visualisierung beider Vorgehen wurde vorgestellt. Über die visuelle Editoro-

berfläche werden den implementierten Unreal-Actors die jeweils benötigten Parameterwerte übergeben. Beide Verfahren bauen einen graphentheoretischen Baum auf, der von dem implementierten Modellgenerierungssystem zu 3D-Modelldaten verarbeitet und mithilfe des bereitgestellten Grafiksystems dargestellt wird.

Der Einfluss übergebener Parameter auf die generierte Baumstruktur wird behandelt. Definitionen von L-Systemen können sich an dem Verzweigungsverhalten realer Pflanzen orientieren, um realistische Ergebnisse zu liefern. [PL90, S.51ff] Die verschiedenen Parameter des Space Colonization Algorithmus können mit bestimmten Wachstumsmustern von Bäumen in Zusammenhang gesetzt werden, um komplexe Baumstrukturen zu generieren. [RLP07, S.5]

## 6.1 Bewertung und Vergleich der Ergebnisse

Es wurden zwei sich ergänzende Verfahren gewählt, um eine Vielfalt verschiedener Baumstrukturen generieren zu können. Diese Verfahren unterscheiden sich in Hinsicht auf die visuellen Ergebnisse, die Effizienz bei der Anwendung und ihrer Benutzerfreundlichkeit.

### 6.1.1 Visuell

Der Space Colonization Algorithmus kann insbesondere für die Darstellung der unregelmäßigen Form von Laubbäumen aus den gemäßigten Breiten verwendet werden. Auch ohne die Ergänzung durch Blätter oder Blüten besitzen die sich ergebenden Baumstrukturen eine realistische Form. Die Funktionsweise des Algorithmus stellt sicher, dass keine Überschneidungen von Zweigen stattfinden und ermöglicht eine Anpassung der Baumstruktur an einschränkende Bedingungen hinsichtlich des Wachstumsraumes. [RLP07, S.5]

Es ist jedoch schwer andere Sorten von Bäumen, wie Nadelhölzer oder tropische Bäume, mit einem klar definierten Stamm und tragenden Ästen darzustellen. Die Einführung des gewichteten Wachstums stellt einen Ansatz für die Generierung solcher Baumsorten dar – Abbildung 5.10b zeigt ein Beispiel für eine Tannenbaumähnliche Struktur. Diese Erweiterung liefert jedoch nur unter einer Vielzahl von Bedingungen visuell ansprechende Ergebnisse.

L-Systeme bieten eine größere Kontrolle über die generierten Baumstrukturen und erlauben die Simulation natürlicher Wachstumsmuster – unter anderem ist die Generierung von Baumsorten möglich, die mithilfe des Space Colonization Algorithmus nur schwer oder überhaupt nicht generierbar sind. Durch die Anpassung von Konstanten können mithilfe derselben L-System-Definition sich visuell stark unterscheidende Baumstrukturen erstellt werden. Da die Generierung der Modelle jedoch strikt nach den festgelegten Regeln abläuft, sind in manchen Fällen bestimmte Muster und Regelmäßigkeiten in der Baumstruktur erkennbar, was den Modellen eine unnatürliche Wirkung verleiht. Ein solcher Eindruck kann durch die Anpassung von Konstanten und die Verwendung eines Tropismus-Einflusses reduziert werden, erlaubt jedoch nicht dieselben, unregelmäßigen Formen, die mithilfe des Space Colonization Algorithmus möglich sind. [RLP07, S.6]

### 6.1.2 Effizienz

Da L-Systeme auf Grundlage der Ersetzung von Zeichenketten arbeiten, besteht ein Großteil der Generierungszeit des L-System-Actors aus der Konstruktion des graphentheoretischen Baumes durch die Turtle-Interpretation und die darauf folgende Berechnung der Vertexdaten durch das Modellierungssystem. Dies erlaubt eine vergleichsweise schnelle Generierung von komplexen 3D-Modellen. Selbst Modelle mit mehreren Millionen Vertizes – einer Datenmenge die in Echtzeitanwendungen normalerweise nicht für einzelne Modelle benötigt wird – werden auf einem Desktop-Computer mit 3.3GHz-Prozessor in wenigen Sekunden erstellt.

Der Nachteil ist jedoch, dass für die Darstellung einer realistisch wirkenden Baumstruktur durch ein L-System, ohne dass die Aststruktur größtenteils von Blättern verdeckt wird, eine vergleichsweise große Menge Modelldaten benötigt wird. Die vorgestellte Kurvenreduktion kann angewandt werden, hat jedoch kaum Auswirkungen auf die Datenmenge, da sowohl Astsegmente als auch Verzweigungen in den Produktionsregeln definiert und somit nicht auf eine automatische Kurvenreduktion angewiesen sind.



(a)  $N = 5000, D = 20.0, N_I = 1000, n_{min} = 8, n_{max} = 8, max_K = 1.0$ , Generierungszeit = 2.416s, 103725 Vertizes.

(b) Ableitung anhand Gleichung 5.3 entsprechend den Werten aus Abbildung 5.4d.  $n_{min} = 8, n_{max} = 8, max_K = 1.0$ , Generierungszeit = 0.233s, 206658 Vertizes.

Abb. 6.1: Vergleich von Generierungszeit und resultierender Datenmenge zwischen L-System-Actor und Space-Colonization-Actor. Eigene Abbildungen.

Die Zeit, welche für die Generierung einer Baumstruktur durch den Space Colonization Algorithmus benötigt wird, stellt derzeit insbesondere für die Anwendung in digitalen Spielen ein Problem dar – die Generierung einer einzigen, recht simplen Baumstruktur kann mehrere Sekunden beanspruchen. Die Generierungszeit ist jedoch stark abhängig von der Anzahl an Einflusspunkten, der Schrittweite sowie der maximalen Iterationsanzahl und kann somit von einem Benutzer eingeschätzt und angepasst werden. Weiterhin bietet die vorgestellte Implementierung Potenzial für zukünftige Erweiterungen, welche eine Verminderung der Generierungszeit als Ziel haben.

Die durch den Space Colonization Algorithmus generierte Datenmenge lässt sich durch eine Vielzahl von Parametern beeinflussen – insbesondere unter Verwendung

der implementierten Kurvenreduktion kann die Datenmenge eines Baummodells verringert werden, ohne die Baumstruktur visuell stark zu beeinträchtigen.

### 6.1.3 Benutzerfreundlichkeit

Für die Verwendung eines L-System-Actors muss der Benutzer eine genaue Vorstellung über den Aufbau der zu generierenden Baumstruktur sowie Kenntnisse über die korrekte Definition von L-Systemen besitzen, um diese in Form von Axiomen, Produktionsregeln und Konstanten angeben zu können. Dies beansprucht ein gewisses Basiswissen über das Wachstumsverhalten von Bäumen sowie ein Verständnis für die entsprechende Umsetzung als L-System. [DL05, S.86]

Bei der Verwendung eines Space-Colonization-Actors hingegen gibt es klare Zusammenhänge zwischen der Form einer Baumstruktur sowie den numerischen und Booleschen Parameterwerten, ohne dass ein tieferes Verständnis der Funktionsweise des Algorithmus erforderlich ist. Ein Benutzer kann immer wieder Parameter verändern, die Auswirkungen beobachten und weitere Änderungen durchführen, bis die gewünschte Baumstruktur generiert wurde. [DL05, S.89]

## 6.2 Wünschenswerte Erweiterungen

Die vorgestellten Implementierungen der gewählten Vorgehen bieten Potenzial für Verbesserungsmöglichkeiten, die in zukünftigen Arbeiten behandelt werden können.

### 6.2.1 L-Systeme

#### *Konditionale L-Systeme*

In der von Prusinkiewicz u.a. vorgestellten Form fordern parametrische L-Systeme die Auswertung einer Bedingung vor der Ersetzung des parametrischen Wortes durch den Nachfolger. Die Bedingung bestimmt, ob die Ersetzung des Wortes durch den Nachfolger stattfindet. Konditionen sind nicht zwingend notwendig für die Generierung realistischer Baumstrukturen, würden dem Benutzer jedoch einen weiteren Kontrollmechanismus für die Gestaltung von Baumstrukturen bieten. Ein Benutzer könnte beispielsweise festlegen, dass nach einer vorgegebenen Anzahl von Ableitungen die Bildung großer, tragender Äste durch die Produktion kleiner Verzweigungen ersetzt wird. [PL90, S.41f]

#### *Stochastische L-Systeme*

Stochastische L-Systeme erweitern die Definition von Produktionsregeln um ein zufälliges Element und unterscheiden sich somit von der deterministischen Funktionsweise der vorgestellten L-Systeme. Demselben parametrischen Wort werden unterschiedliche Produktionsregeln und zugehörige Wahrscheinlichkeitswerte zugeordnet. Anhand dieser Werte wird zufällig bestimmt, welche der Produktionsregeln bei einer Ableitung des Wortes verwendet wird. Bei gleichbleibender L-System-Definition ergeben sich dadurch unterschiedliche Ergebnisse. Stochastische

L-Systeme stellen somit eine Lösung von visuell unnatürlich wirkenden Regelmäßigkeiten in Baumstrukturen dar. [PL90, S.28]

#### *Benutzerfreundlichkeit*

Um die Verwendung von L-Systemen zu vereinfachen, können vordefinierte Wachstumsarten implementiert und durch den Benutzer mithilfe von numerischen Parametern anpassbar gemacht werden. Diese Erweiterung würde die intuitive Verbindung zwischen den Parametern und Veränderungen an der Baumstruktur erlauben, wird jedoch durch die Anzahl der vordefinierten Wachstumsarten limitiert. Um diesem Problem entgegenzuwirken, könnte beispielsweise die Angabe von Anzahl, Länge und Abzweigungswinkel von Verzweigungen in dem Nachfolger einer Produktionsregel mithilfe von numerischen Parametern ermöglicht werden. Aus den eingegebenen Informationen werden daraufhin die entsprechenden Produktionsregeln automatisch definiert.

#### *Einsatz vordefinierter Modelle*

Die Möglichkeit benutzerdefinierte Zeichen durch vordefinierte 3D-Modelle zuersetzen würde den Einsatz von Blüten, Blättern oder anderer Strukturen im Baummodell ermöglichen. Aufgrund der Selbstähnlichkeit von, durch L-Systeme generierten, Baumstrukturen besteht ebenfalls die Möglichkeit einen Teilbaum des graphentheoretischen Baums in mehreren Rotationen und Positionen wiederzuverwenden um somit den benötigten Interpretationsaufwand zu verringern. [DL05, S.82]

### **6.2.2 Space Colonization Algorithmus**

#### *Positionsabfragen*

Ungefähr 90% der Generierungszeit, die für einen Space-Colonization-Actor benötigt wird, entsteht bei der Überprüfung, welche Astsegmente sich innerhalb des Einflussradius eines Einflusspunktes befinden. Derzeit wird jede Position eines Punktes mit jeder Endposition eines Astsegments verglichen. Es gibt einige Möglichkeiten eine solche Brute-Force-Methode zu verbessern, beispielsweise durch die Verwendung von Octrees. Diese stellen eine Methode dar, dreidimensionalen Raum in Form eines graphentheoretischen Baumes aufzuteilen um somit die Anzahl von benötigten Positionsabfragen zu reduzieren. [EKH10, S.1]

#### *Einflussbereiche*

Die derzeit implementierte Möglichkeit Einflussbereiche anzugeben ist begrenzt auf die Definition von Primitiven in Form von Kugeln und Zylindern. Werden mehrere Primitive als Einflussbereiche miteinander kombiniert, gibt es keine Möglichkeit die Punkteverteilung in sich überschneidenden Bereichen zu normalisieren. Eine Erweiterung der Primitive um Rechteck-, Kegel- und Pyramidenformen sowie die Möglichkeit, Bereiche zu definieren, in welchen keine Einflusspunkte generiert werden, würden die Gestaltungsmöglichkeiten der generierbaren Baumstrukturen verbessern.

Die Anpassung der Verteilung von Einflusspunkten innerhalb der Einflussbereiche stellt eine zusätzliche Erweiterung dar. Runions u.a. zeigen, dass die ausschließliche Platzierung von Punkten auf der Oberfläche eines Einflussbereichs oder das dynamische Hinzufügen von Punkten in jeder Iteration des Algorithmus die Generierung zuvor unmöglichener Baumstrukturen zulässt. [PL90, S.5f]

### 6.2.3 Allgemeine Erweiterungen

Die folgenden Vorschläge befassen sich mit Erweiterungen, welche sich auf alle generierten Baumstrukturen beziehen.

#### *Generierung zur Laufzeit*

Im Bereich der Echtzeitanwendungen ist die Zeit, welche für den Start einer Anwendung aufgebracht wird, limitiert. Die Anzahl von Baumstrukturen, welche in einer solchen Zeit generiert werden können ist begrenzt. Eine Möglichkeit dieses Problem zu umgehen ist, die Baumstrukturen während der Laufzeit zu generieren. Beispielsweise könnte sich ein Spieler bereits im Level bewegen, während um ihn herum Baumstrukturen nach ihrer Fertigstellung dargestellt werden. Das resultierende, plötzliche Auftauchen von Objekten ist ein Nachteil, der gegen den Vorteil einer verringerten Ladezeit abgewägt werden muss.

#### *Level-of-Detail (LOD)*

Je weiter ein 3D-Modell vom Beobachter entfernt ist, desto kleiner erscheint es auf dem Bildschirm. Eine komplexe Baumstruktur ist dann nicht mehr von einem Baummodell mit reduzierter Datenmenge unterscheidbar. Die derzeitige Implementierung stellt, unabhängig von der Entfernung, alle Modelle mit derselben Datenmenge dar. Unterschiedlich komplexe Modelle einer Baumstruktur – die LOD-Qualitätsstufen des Modells – können in Abhängigkeit von der Entfernung zum Beobachter dynamisch ausgetauscht werden. Dadurch wird die, in jedem Frame darzustellende, Datenmenge reduziert. [DL05, S.267]

#### *Visuell*

Den generierten Baumstrukturen fehlen zwei ausschlaggebende Merkmale von Bäumen: Die Darstellung von Baumrinde und Blättern.

Die Oberflächenbeschaffenheit der Strukturen kann derzeit lediglich durch Angabe eines Materials durch den Benutzer selbst beeinflusst werden. Die Möglichkeit prozedural generierte Texturen zu verwenden, welche Baumrinde simulieren, würde jedoch eine wünschenswerte Erweiterung darstellen.

Die Blattform und ihre Verteilung stellen grundlegende, charakteristische Merkmale von laubtragenden Bäumen dar. Die Angabe von Blattformen durch Benutzer oder die prozedurale Generierung sowie Verteilung auf einer Baumstruktur wäre somit ebenfalls eine Erweiterung, welche in zukünftigen Arbeiten behandelt werden könnte.

### *Verteilung*

Die derzeit generierten Baumstrukturen müssen einzeln in einem Level platziert und ihre Parameter festgelegt werden. Dies ist für eine kleine Szene oder die Platzierung von wenigen Bäumen an bestimmten Positionen praktikabel, nicht jedoch für Szenen mit hunderten von darzustellenden Bäumen. Die Verteilung von Baumstrukturen in einer Szene, durch vollständig prozedurale Methoden oder mit Benutzerbeteiligung wäre eine wünschenswerte Erweiterung. Ein Benutzer könnte beispielsweise bestimmte Wachstumsbereiche angeben, in welchen vorgegebene Baumarten mit zufälligen Parametern platziert werden.

---

## Literaturverzeichnis

- Bak. BAKER, MARTIN JOHN: *Maths - Angle between vectors.* <http://www.euclideanspace.com/maths/algebra/vectors/angleBetween/index.htm>.
- Bal98. BALZERT, HELMUT: *Lehrbuch der Software-Technik : Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung.* Spektrum Akademischer Verlag GmbH, 1998.
- Bec05. BECKER, PETE: *Working Draft, Standard for Programming Language C++*, 2005. <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1905.pdf>.
- DL05. DEUSSEN, OLIVER und BERND LINTERMANN: *Digital Design of Nature - Computer Generated Plants and Organics.* Springer-Verlag Berlin Heidelberg 2005, 2005.
- EKH10. EBERHARDT, HENNING, VESA KLUMPP und UWE D. HANEBECK: *Density Trees for Efficient Nonlinear State Estimation*, 2010. [http://isas.uka.de/Publikationen/Fusion10\\_EberhardtKlumpp.pdf](http://isas.uka.de/Publikationen/Fusion10_EberhardtKlumpp.pdf).
- Eng. *Unreal Engine Documentation : Engine Features.* <https://docs.unrealengine.com/latest/INT/Engine/index.html>.
- Gre. *Green One - A landmark render of XfrogPlants by Jan Walter Schliep.* [http://xfrog.com/gallery/landscapes/green01\\_big-1600small.jpg.php](http://xfrog.com/gallery/landscapes/green01_big-1600small.jpg.php).
- GSJ04. GOLDMAN, RON, SCOTT SCHAEFER und TAO JU: *Turtle Geometry in Computer Graphics and Computer Aided Design*, 2004. <http://www.cs.wustl.edu/~taoju/research/TurtlesforCADRevised.pdf>.
- Lux14. LUX, PROF. DR. ANDREAS: *Algorithmen und Datenstrukturen - Vorlesungsskript Kapitel 4*, 2014.
- Man83. MANDELBROT, BENOIT B.: *The Fractal Geometry of Nature.* W. H. Freeman and Company, 1983.
- PL90. PRUSINKIEWICZ, PRZEMYSŁAW und ARISTID LINDENMAYER: *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, eBook Auflage, 1990. <http://algorithmicbotany.org/papers/abop/abop.pdf>.
- Proa. *Procedural Mesh Component in C++ : Getting Started.* [https://wiki.unrealengine.com/Procedural\\_Mesh\\_Component\\_in\\_C%2B%2B:Getting\\_Started](https://wiki.unrealengine.com/Procedural_Mesh_Component_in_C%2B%2B:Getting_Started).

- Prob. *Profiling, How To Count CPU Cycles Of Specific Blocks Of Your Game Code.* [https://wiki.unrealengine.com/Profiling,\\_How\\_To\\_Count\\_CPU\\_Cycles\\_Of\\_Specific\\_Blocks\\_Of\\_Your\\_Game\\_Code](https://wiki.unrealengine.com/Profiling,_How_To_Count_CPU_Cycles_Of_Specific_Blocks_Of_Your_Game_Code).
- RFL<sup>+05</sup>. RUNIONS, ADAM, MARTIN FUHRER, BRENDAN LANE, PAVOL FEDERL, ANNE-GAËLLE ROLLAND-LAGAN und PRZEMYSLAW PRUSINKIEWICZ: *Modeling and visualization of leaf venation patterns*, 2005. <http://algorithmicbotany.org/papers/venation.sig2005.pdf>.
- RLP07. RUNIONS, ADAM, BRENDAN LANE und PRZEMYSLAW PRUSINKIEWICZ: *Modeling Trees with a Space Colonization Algorithm*, 2007. <http://algorithmicbotany.org/papers/colonization.egwnp2007.pdf>.
- Sch14. SCHMITZ, PROF. DR. HEINZ: *Theoretische Informatik - Vorlesungsskript*, 2014.
- STN16. SHAKER, NOOR, JULIAN TOGELIUS und MARK J. NELSON: *Procedural Content Generation in Games*. Springer International Publishing Switzerland 2016, 2016.
- Sura. SURIDGE, JAYELINDA: *Modelling by numbers: Part One A: An introduction to procedural geometry*. [http://www.gamasutra.com/blogs/JayelindaSuridge/20130903/199457/Modelling\\_by\\_numbers\\_Part\\_One\\_A.php](http://www.gamasutra.com/blogs/JayelindaSuridge/20130903/199457/Modelling_by_numbers_Part_One_A.php).
- Surb. SURIDGE, JAYELINDA: *Modelling by numbers: Part Two A: The cylinder*. [http://www.gamasutra.com/blogs/JayelindaSuridge/20130905/199626/Modelling\\_by\\_numbers\\_Part\\_Two\\_A.php](http://www.gamasutra.com/blogs/JayelindaSuridge/20130905/199626/Modelling_by_numbers_Part_Two_A.php).
- Unra. *Unreal Engine Documentation : Content Examples*. <https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/>.
- Unrb. *Unreal Engine Documentation : Unreal Engine 4 Terminology*. <https://docs.unrealengine.com/latest/INT/GettingStarted/Terminology/index.html>.
- Wha. *Unreal Engine Features*. <https://www.unrealengine.com/unreal-engine-4>.

# A

---

## Projektaufbau und Beispielanwendung

Für die Visualisierung der Ergebnisse dieser Arbeit in einer 3D-Echtzeitanwendung wurde eine auf Windows ausführbare Beispielanwendung erstellt, welche auf dem in der Unreal Engine 4 erstellten Projekt basiert. [Unrb] Das in der Engine-Version 4.14.3 entwickelte Projekt enthält alle Inhalte und Quellcodedateien, welche im Rahmen dieser Arbeit erstellt wurden. Das Projekt trägt den Arbeitstitel „Bachelor“.

Ein Git-Repository ermöglicht den Zugriff auf die Projektdateien und ist über den folgenden Link erreichbar:

<https://goo.gl/QTKitm>

Die Beispielanwendung ist über folgenden Link verfügbar:

<https://drive.google.com/open?id=0Bw88ARHio-5sTkERkc4ZjB6NOU>

Um die Umgebung der Anwendung zu gestalten und Informationstafeln zu erstellen wurden frei verfügbare Beispielinhalte aus den Content Examples verwendet. [Unra]

### A.1 Projektaufbau

Die Inhalte und der Quellcode können nach dem Öffnen des Projekts im sogenannten Contentbrowser (engl. für Inhaltsnavigation) eingesehen werden. Die enthaltene Ordnerstruktur ist wie folgt aufgebaut:

#### A.1.1 Content

Der Content-Ordner enthält alle Inhalte, die im Projekt verwendet werden. Dieser ist in weitere Unterordner unterteilt:

**DebugUtils:** Enthält steuerbare Hilfs-Actors für die Entwicklung und das Testen des Projekts.

**ExampleContent:** Enthält die aus den Content Examples [Unra] verwendeten Inhalte, wie 3D-Modelle, Texturen und Blueprint-Klassen für die einfache Erstellung des Beispielraumes.

**Levels:** Enthält die im Projekt verwendeten Levels.

**Material:** Enthält die selbstständig erstellten Materials.

**ProceduralPlants:** Enthält Blueprint-Klassen, welche von den im Quellcode erstellten Actors ableiten, um eine einfache Positionierung zu ermöglichen.

**UI:** Enthält alle selbstständig erstellten Menü-Blueprint-Klassen sowie die verwendete Hintergrundtextur.

### A.1.2 C++ Classes

Der C++ Classes Ordner enthält alle von der Actor-Basisklasse ableitenden C++ Klassen, was eine Beschränkung auf die im Level platzierbaren Klassen darstellt. Der Ordner enthält somit nicht alle im Rahmen dieser Arbeit entwickelten C++ Codedateien. Um eine vollständige Liste der Dateien einzusehen, muss das mit dem Unreal-Projekt verbundene Visual-Studio-Projekt geöffnet werden. Die C++ Codedateien sind wie folgt aufgebaut:

**Procedural:** Enthält die in Abschnitt 4.1 beschriebene Datenklasse für die Baumrepräsentation sowie alle für die Implementierungen der beiden Generierungsverfahren verwendeten Klassen. Beinhaltet weiterhin die für das Modellgenerierungssystem zusätzlich benötigten Datenklassen.

**Utility:** Enthält die statischen Funktionsklassen, welche das in Abschnitt 4.4 beschriebene Modellgenerierungssystem bilden.

## A.2 Beispielanwendung

Die Beispielanwendung wurde für eine Demonstration der generierten Baumstrukturen erstellt und kann auf Windows 32-Bit und 64-Bit Systemen ausgeführt werden.

### A.2.1 Aufbau

Die Beispielanwendung besteht aus Hauptmenü, Ladebildschirm und dem Beispiellevel. Im Hauptmenü kann zwischen dem Start des Levels und dem Verlassen der Anwendung gewählt werden. Nach dem Start des Levels geht das Programm zur

Generierung der Baumstrukturen über – dieser Vorgang kann, je nach Prozessorkapazität, einige Zeit in Anspruch nehmen.

Nach Abschluss der Generierung wird das aus zwei Haupträumen bestehende Beispiellevel dargestellt. Die Räume enthalten jeweils durch L-System-Actors und Space-Colonization-Actors generierte Baumstrukturen.

### A.2.2 Steuerung

Das Programm besitzt folgende Tastenbelegung:

| Funktion                                 | Taste            |
|--|------------------|
| Vorwärts bewegen                         | W                |
| Rückwärts bewegen                        | S                |
| Nach links bewegen                       | A                |
| Nach rechts bewegen                      | D                |
| Nach oben bewegen                        | E                |
| Nach unten bewegen                       | Q                |
| Bewegung verschnellern                   | L-Shift gedrückt |
| Nicht verwendete Einflusspunkte anzeigen | V                |
| Programm pausieren                       | Esc              |
| Vollbildmodus / Fenstermodus umschalten  | F11              |

Tabelle A.1: Tastenbelegung

---

## B

---

### Erklärung der Kandidatin / des Kandidaten

- Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.
- Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

---

Datum

---

Unterschrift der Kandidatin / des Kandidaten