Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Heuer

Algorithmische Erzeugung von 3D-Low-Poly-Bäumen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Heuer

Algorithmische Erzeugung von 3D-Low-Poly-Bäumen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Zhen Dai

Eingereicht am: 6. Januar 2022

Florian Heuer

Thema der Arbeit
Algorithmische Erzeugung von 3D-Low-Poly-Bäumen

Stichworte
L-Systeme, Marching-Cubes-Algorithmus, Distanzfunktionen, Dreiecksnetze

Kurzzusammenfassung
Es wird eine Anwendung vorgestellt, die 3D-Baummodelle in einem Low-poly-Stil prozedural erzeugen soll. Es soll gezeigt werden, ob dies mit der vorgestellten Methode möglich ist. Dazu wird eine Baumsynthese präsentiert, die L-Systeme, Turtleinterpreter, Distanzfunktionen und den Marching-Cubes-Algorithmus kombiniert, um ein 3D-Baummodell in Form eines Dreiecksnetzes zu erzeugen. Ein Anwender ist in die Baumgenerierung involviert und ändert Parameter und L-Systeme in einer Benutzeroberfläche. Dadurch werden Unterschiede in Aussehen und Form der Bäume erzielt. Mit der Anwendung konnten diverse Bäume erzeugt werden. Die Darstellung verschiedener Baumarten ist in einem Low-poly-Stil möglich. Die Verwendung der erzeugten Modelle ist in anderen Anwendungen möglich.
Florian Heuer

Title of the paper
Algorithmic generation of 3D low poly trees

Keywords
l-system, marching cubes algorithm, distance function, triangle mesh

Abstract
An application is presented that is intended to procedurally generate 3D tree models in a low-poly style. It shall be shown whether this is possible with the presented method. To this end, a tree synthesis is presented that combines l-systems, turtle interpreter, distance functions, and the Marching Cubes algorithm to generate a 3D tree model in the form of a triangle mesh. A user is involved in the tree generation and changes parameters and l-systems in a user interface. This results in differences in the appearance and shape of the trees. Various trees could be generated with the application. The representation of different tree types is possible in a low-poly style. The use of the generated models is possible in other applications.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

[1](2021). Cg science for artists part 2: The real-time rendering pipeline, Adresse: http://www.cgchannel.com/2010/11/cg-science-for-artists-part-2-the-real-time-rendering-pipeline/ (besucht am 10. 12. 2021).

[2](2021). Combination sdf, Adresse: https://www.shadertoy.com/view/lt3BW2 (besucht am 31. 12. 2021).

Akronyme

3DTG 3D-Turtlegrafik. 25–27, 31, 32, 34–38, 40, 41, 51

3DTI 3D-Turtle-Interpreter. ix, 25, 26, 31, 33–35, 41, 51, 54, 57

DF Distanzfunktion. 25, 26, 32, 40–43, 45, 46, 48, 51, 52

LPTG Low-Poly-Tree-Generator. vii, 2, 5, 22, 24, 25, 28, 52

MCA Marching-Cubes-Algorithmus. 14, 15, 25, 27, 41, 43, 45, 46, 48, 50–53

TLT Triangle Lookup Table. 15–17

UI User-Interface. vii, 25–27, 29, 30, 33, 35, 50, 51, 54

Glossar

Computer Generated Imagery (CGI) Fachausdruck für erzeugte Bilder im Bereich der Filmproduktion, Computersimulation und visueller Effekte. 1

Dichasium Verzweigungstypus der Sprossachse, bei dem die Hauptachse ihr Wachstum einstellt, und jeweils zwei Seitensprossen gleichwertig auswachsen. Ein sympodiales System mit zwei Achsen (Dichasium) ist besonders beim Flieder (Syringa) deutlich zu beobachten[3]. 34

jMonkey JMonkey zählt zu den Spiel-Engines und ist Java umgesetzt. Eine Spiel-Engine ist ein Framework, welches die Darstellung von Spielen und Spielabläufen steuert. Dazu kapselt das Framework Methoden für z.B. die Erzeugung von polygonalen Netzen, physikalischen Berechnungen, Audiowiedergabe, Ein- und Ausgabe von Hardware, Datenaustausch über das Netzwerk u.v.m. Sie dient dem Entwickler dabei, aufwendige Implementierungen zu vereinfachen oder zu ersparen. xi, 6, 25, 51

Low-poly-Stil Eine Art (Stil) Bilder oder Objekte mit einem Netz bestehend aus Polygonen darzustellen. Die Anzahl der Polygone (poly) im Netz ist dabei gering (low). 1–3, 19, 22, 50–52

PCG-Framework Framework der HAW das Implementierungen zum Thema der prozeduralen Content-Generierung zusammen fasst und Funktionen bereit stellt. Das Framework bindet JMonkey ein. . 25, 33, 43, 51

prozedurale Content-Generierung Methoden in der Informatik, um Inhalte eines Programms wie z.B. Texturen, 3D-Objekte, Musik usw. mithilfe von Algorithmen und bereits vorhandenen von Menschen erzeugten Inhalten zu generieren. 1, 21

Renderings Aus Rohdaten computergenerierte Bilder. Rohdaten können z.B. Dreiecksnetze sein. 1

Splines Ein Spline n-ten Grades ist eine Funktion, die stückweise aus Polynomen mit maximalem Grad n zusammengesetzt ist [4]. 19, 54

Stack Zu dt. als Stapelspeicher bezeichnet, ist eine Datenstruktur, die nach dem LIFO-Prinzip funktioniert. Daten, welche als letztes gespeichert werden, werden wieder als erste aus diesem Speicher genommen. Ähnlich wie bei einem Kartenstapel. 7, 36, 37

[3](2022). Dichasium, Adresse: https://www.spektrum.de/lexikon/biologie-kompakt/dichasium/3036 (besucht am 03. 01. 2022).

1. Einführung

1.1. Motivation

Die Erstellung von Bäumen und Pflanzen ist in der Computergrafik eine aufwendige und zeitintensive Aufgabe. Deswegen ist es von besonderem Interesse, die Arbeit von 3D-Designern in diesem Bereich mithilfe von prozeduraler Content-Generierung zu unterstützen. In der Vergangenheit sind einige beeindruckende Resultate in diesem Bereich entstanden. Sehr realitätsnahe Renderings wurden bereits Mitte der achtziger Jahre mit der Arbeit von Bloomenthal [5] erreicht. Der Detailgrad und die Möglichkeiten wurden seitdem bis heute gesteigert. Ein Beispiel für das heutige Maß an Realitätsnähe in Computerspielen oder Computer Generated Imagery (CGI) in Filmen und in Bezug auf die Darstellung von Vegetation bietet die Firma "SpeedTree". Als Beispiel ist die Vegetation des Films "Avatar" zu nennen[1].

In den letzten Jahren ist aber durchaus ein Trend zu erkennen, Computerspiele in einem reduzierten Grad an Detail und Form in einem Low-poly-Stil darzustellen[27]. Was in den neunziger Jahren noch Mittel zum Zweck war, um virtuelle Welten in flüssigen Bildraten berechnen zu können, ist heute zum Stil geworden, der von Künstlern und Designern aufgegriffen wird. 3D-Modelle bestehen aus Netzen, diese wiederum aus Polygonen. Polygone bestehen häufig aus Dreiecken. Je höher die Anzahl der Dreiecke in einem 3D-Modell ist, desto aufwendiger ist es, das Modell von der Hardware zu rendern. Die damalige Hardware war zu schwach, um hochauflösende 3D-Modelle in Computerspielen in flüssigen Bildraten zu rendern. Gerade die Spieleindustrie war also angehalten, die 3D-Modelle simpel zu halten, um ein flüssiges Spielerlebnis zu ermöglichen. Heute gilt dies nicht mehr so stark. Die heutige Hardware leistet ein Vielfaches, sodass sehr realistische und dabei flüssige Renderings möglich sind.

[1](2009). Speedtree in avatar, Adresse: https://store.speedtree.com/downloads/Avatar_User_Profile.pdf (besucht am 22. 12. 2021).

Trotzdem ist das aktuell meistverkaufte Spiel[2] weltweit "Minecraft"[3] im Low-poly-Stil gehalten (siehe Abb. 1.1). Dieser Stil scheint sich also bei Konsumenten, wie bei Designern und Entwicklern großer Beliebtheit zu erfreuen.

Abbildung 1.1.: Low-poly-Stil am Beispiel von Minecraft

Auch die HAW hat ein Projekt, welches sich mit prozeduraler Content-Generierung beschäftigt. Es sind bereits Arbeiten entstanden, die sich mit der Erzeugung von prozedural generierten Häusern, Autos oder urbanen Umgebungen beschäftigt haben. Diese sind ebenfalls in einem Low-poly-Stil gehalten. In diesen Rahmen ist auch die folgende Arbeit eingebettet. Sie beschäftigt sich mit der Generierung von Bäumen im Low-poly-Stil.

1.2. Zielsetzung

Im Rahmen dieser Arbeit soll eine Anwendung entstehen, im Folgenden Low-Poly-Tree-Generator (LPTG) genannt, mit der es möglich ist, Bäume in einem Low-poly-Stil zu erzeugen. Der Anwender soll Teil eines teil-automatisierten Prozesses sein und so zusammen mit der Software Bäume erstellen können. Die folgenden Methoden werden dafür verwendet. Mithilfe

[2](2021). Absatzzahlen der weltweit meistverkauften videospiele in millionen stück, Adresse: https://de.statista.com/statistik/daten/studie/36854/umfrage/verkaufszahlen-der-weltweit-meistverkauften-videospiele/ (besucht am 22. 12. 2021).

[3](2021). Minecraft, Adresse: https://www.minecraft.net/de-de/about-minecraft (besucht am 22. 12. 2021).

von L-Systemen (siehe Abb. 1.2, Kasten A) und einem Turtleinterpreter wird ein ”Baumskelett” (siehe Abb. 1.2, Kasten B) erzeugt. Daraus wird im Anschluss ein Baumkörper mit Krone und Stamm modelliert. Die Oberfläche (siehe Abb. 1.2, Kasten C, gestrichelte Linie) von Krone und Stamm wird mithilfe von Distanzfeldern beschrieben. Diese Oberfläche wird mithilfe des Marching-Cubes-Algorithmus polygonisiert und so jeweils ein Dreiecksnetz erzeugt, welches den Baumstamm und die Krone als 3D-Modell darstellt (siehe Abb. 1.2, Kasten D).

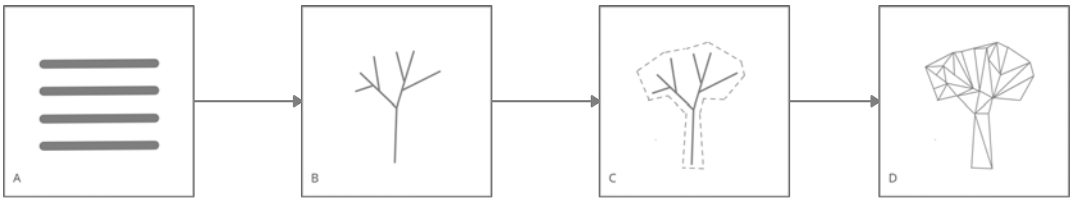


Abbildung 1.2.: Baumsynthese (Quelle: Eigene Darstellung)

Laut aktuellem Wissensstand verfolgt diese Arbeit als erste das Ziel, Bäume im Low-poly-Stil prozedural zu generieren. Es soll geprüft werden, ob die oben vorgestellte Synthese ein geeignetes Mittel dafür ist.

1.3. Aufbau der Arbeit

Diese Arbeit folgt einer aufbauenden Logik. Zuerst werden die nötigen Grundlagen beschrieben, welche zum Verständnis dieser Arbeit zwingend nötig sind. Danach folgen Konzept und Umsetzung. Hier soll deutlich werden, wie die Anwendung im Detail funktioniert. Anschließend erfolgt eine Evaluation der Ergebnisse und eine Prüfung, ob das Ziel dieser Arbeit erreicht wurde. Abschließend wird ein Ausblick auf mögliche Erweiterungen für die Anwendung gegeben. Dabei werden diverse Verbesserungsmöglichkeiten aufgezeigt.

Aufgrund der großen Anzahl an Grafiken, die im Zuge der Arbeit entstanden sind, wurden diese überwiegend in den Anhang ausgegliedert, um den Lesefluss nicht zu beeinträchtigen. Einige teilweise lange Listen und Parameter wurden ebenfalls im Anhang untergebracht.

2. Grundlagen

In dem folgenden Kapitel werden die Grundlagen beleuchtet, welche für die in Abbildung 1.2 gezeigte Baumsynthese benötigt werden. Des Weiteren wird ein Blick auf ähnliche Arbeiten und alternative Lösungsansätze geworfen.

2.1. Computergrafik

Fundamental für die oben beschriebene Baumsynthese ist das Anzeigen und Darstellen von Grafiken auf dem Computerbildschirm, weil das fertige Baummodell auf dem Bildschirm angezeigt werden soll. Dazu ist in heutigen Computersystemen dedizierte Hardware vorhanden, die GPU bzw. Grafikkarte, welche über eine Rendering-Pipeline (siehe Abb. 2.1) verfügt, um Grafiken auf dem Bildschirm darzustellen.

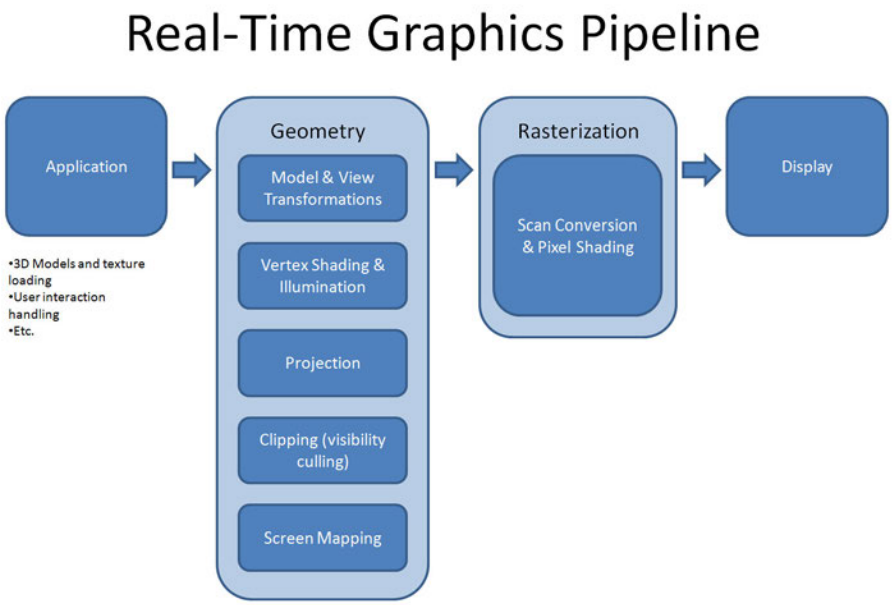


Abbildung 2.1.: Rendering Pipeline (Quelle: CG channel[1])

Durch diese Pipeline werden Grafiken aus Rohdaten in verschiedenen Schritten sukzessive bis zum fertigen Bild erzeugt, welches auf dem Bildschirm angezeigt wird. Die Rohdaten kommen aus der Anwendung selbst. In dieser Arbeit aus dem Low-Poly-Tree-Generator (LPTG) in Form von polygonalen Netzen. Dies ist eine Möglichkeit, die Oberfläche eines geometrischen Körpers auf einer abstrakten Ebene zu beschreiben. In Bezug auf den LPTG ist dies die Oberfläche des Baums, die sich aus einer Kombination aus Oberflächen geometrischer Körper zusammensetzt.

Als spezielle Form polygonaler Netze sind die Dreiecksnetze besonders gut geeignet, um geometrische Körper auf dem Bildschirm darzustellen [16]. Die Vorteile werden im nächsten Unterkapitel beschrieben.

2.1.1. Dreiecksnetz

Ein Dreiecksnetz ist ein Sonderfall des polygonalen Netzes, welches auf die Topologie [24] zurückzuführen ist. Dieses Netz besteht aus Knoten (engl. Vertex) und Kanten (engl. Edge), welche in einer Netzstruktur zusammen Dreiecke bilden (siehe Abb. 2.2).

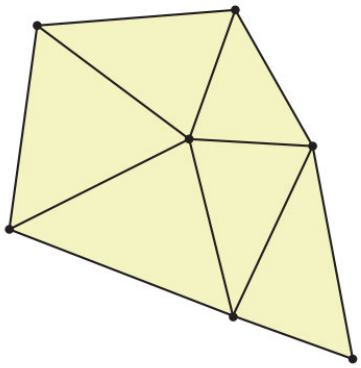


Abbildung 2.2.: Dreiecksnetz [26]

Eine nützliche Eigenschaft ist, dass ein Dreieck im 3D-Raum planar ist. Dies ermöglicht z.B. die Berechnung der Normale [13] eines Dreiecks. Damit wird festgestellt, in welche Richtung

[1](2021). Cg science for artists part 2: The real-time rendering pipeline, Adresse: http://www.cgchannel.com/2010/11/cg-science-for-artists-part-2-the-real-time-rendering-pipeline/ (besucht am 10. 12. 2021).

die Oberfläche eines Dreiecks im 3D-Raum zeigt, was essentiell ist für weitere Verfahren in der Computergrafik z.B. beim Culling [31] und bei diversen Beleuchtungsmodellen [32].

2.1.2. Speicherung von Dreiecksnetzen

Es gibt verschiedene Datenstrukturen, die es ermöglichen, Dreiecksnetze zu speichern. Je nach Art haben diese unterschiedliche Laufzeiten beim Zugriff und unterschiedlichen Speicherbedarf. Der Artikel von Campagna, Kobbelt und Seidel [14] vergleicht diese miteinander und gibt Informationen über Speicherbedarf und Zeitkomplexität der jeweiligen Datenstrukturen an. Aus dem Artikel geht unter anderem hervor, dass die Datenstruktur "Shared Vertex" (siehe Abb. 2.3) den geringsten Speicherbedarf hat, aber in einigen Fällen auch eine deutlich höhere Zugriffszeit als $\mathcal{O}(1)$ besitzt. Es wird eine Liste an Eckpunkten (V_0, V_1 siehe Abb. 2.3) und Dreiecken (Δ_0, Δ_1, Δ_2 siehe Abb. 2.3) zur Speicherung verwaltet. In der Dreiecksliste wird auf die Eckpunktliste verwiesen (siehe Pfeile in Abb. 2.3). Ein Eckpunkt kann so mehrfach für unterschiedliche Dreiecke genutzt werden, belegt aber nur einmal Speicherplatz.

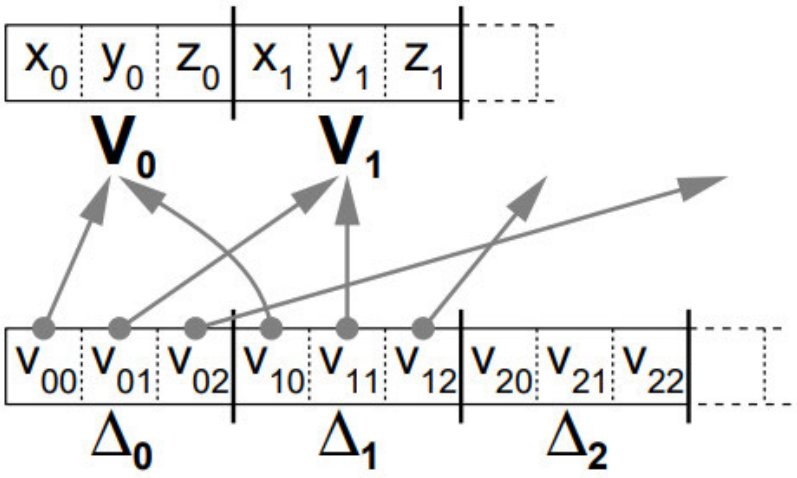


Abbildung 2.3.: Shared vertex Datenstruktur
(Quelle: Campagna, Kobbelt und Seidel [14])

Diese Art der Speicherung von Informationen ist heute üblich und findet auch bei der in dieser Arbeit eingesetzten Spiel-Engine JMonkey Anwendung.

2.2. L-Systeme

Nachdem ein kurzer Einblick gegeben wurde, wie ein Baummodell auf den Bildschirm gezeichnet werden kann und welche Abstraktionen dafür nötig sind, wird im Folgenden ein Blick auf die globale Baumstruktur geworfen, d.h. wie es möglich ist, Verästelungen des Baumes zu beschreiben.

Die Darstellung von Pflanzen ist in der Computergrafik schon seit mehr als 40 Jahren ein Thema, an dem geforscht wird und zu dem es in der Vergangenheit verschiedene Publikationen gab. Eine wegweisende Arbeit ist die von Lindenmayer [2]. Darin beschrieb Lindenmayer erstmals das Wachstum von Pflanzenzellen in Form formaler Grammatiken [1][29], den später nach ihm benannten L-Systemen. Dies ist die Grundlage für das Buch *The Algorithmic Beauty of Plants* von Prusinkiewicz, Lindenmayer, Hanan u. a. [8], in dem beschrieben wird, wie L-Systeme in Grafiken umgesetzt werden und sich so die für diese Arbeit interessanten Baumstrukturen darstellen lassen. Die Darstellung geschieht mit einem Interpreter, der in Unterkapitel 2.3 beschrieben wird.

Zunächst gibt es grundsätzlich verschiedene Arten von L-Systemen. Für diese Arbeit von besonderem Interesse ist ein L-System mit folgenden Eigenschaften.

2.2.1. L-System mit Parametern und Zustandsspeicher

Um Verästelungen, wie sie für eine Baumstruktur typisch sind, darstellen zu können, benötigt man einen Zustandsspeicher, welcher die aktuelle Position des Turtleinterpreters speichert. Dafür eignet sich ein Stack, weil es damit einfach möglich ist, die vorherige Position abzurufen.

Zusätzliche Parameter sorgen für mehr Möglichkeiten bei der Erzeugung von Baumstrukturen, z.B. können darüber Winkel, Länge oder Dicke der Verästelungen bestimmt werden. Bedingungen für Produktionsregeln sind ebenfalls denkbar, z.B. wird ab einer bestimmten Dicke der Verästelungen eine andere Produktionsregel für neue Verästelungen gewählt.

2.2.2. Formale Definition

$$G = \langle V, \Sigma, \omega, P \rangle$$

- V ist das Alphabet
- Σ ist die Menge an Parametern

- $\omega \in (V \times R^*)^+ \mid R \in \mathbb{R}$ ist ein nicht leeres Wort, auch Axiom genannt
- P $\subset (V \times \Sigma^*) \times (\mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma))^*$ ist die Menge endlicher Produktionsregeln, wobei $\mathcal{C}(\Sigma)$ die Menge logischer Ausdrücke und $\mathcal{E}(\Sigma)$ die Menge arithmetischer Ausdrücke Σ bilden

2.2.3. Funktion

Grundsätzlich wird aus dem L-System ein Wort ω' abgeleitet. Dieses Wort entsteht in festgelegten Iterationsschritten n. Die Ableitung läuft wie folgt ab.

Man startet mit dem Axiom ω. Dies wird im ersten Iterationsschritt abgeleitet und durch eine Produktionsregel P_1 aus P für ω ersetzt. Im zweiten Iterationsschritt wird P_1 durch eine oder mehrere Produktionsregeln aus P ersetzt. Die weiteren Schritte folgen dem gleichen Schema. Die Folge an Ableitungen endet, wenn das festgelegte n erreicht ist oder keine Ableitungen mehr stattfinden können, weil keine Ersetzungsregeln für ein Symbol existieren. Ein einfaches Beispiel sei:

$$V = \{+, -, A, B, F\} \tag{2.1}$$
$$\Sigma = \{\emptyset\} \tag{2.2}$$
$$\omega = P_1 \tag{2.3}$$
$$P = \{P_1 \to +A - B, A \to AB, B \to F\} \tag{2.4}$$

n sei 3. Dann ist das Wort ω' (siehe Funktion 2.7) nach den folgenden 3 Iterationsschritten abgeleitet worden. Die Funktionen 2.5 und 2.6 zeigen ω' nach Iteration 1 und 2.

$$\omega' = +A - B \tag{2.5}$$
$$\omega' = +AB - F \tag{2.6}$$
$$\omega' = +ABF - F \tag{2.7}$$

2.3. Turtleinterpreter

Jedes Symbol im Alphabet eines L-Systems lässt sich einer Aktion im 2D- oder 3D-Raum zuordnen, welche der Turtleinterpreter ausführt und so eine Grafik erzeugt. Der Begriff Turtleinterpreter geht darauf zurück, dass man sich eine Schildkröte im Raum vorstellt, die Befehle

erhält und durch den Raum wandert. Die Befehle sind mit den Symbolen aus dem Alphabet V enthalten im Wort ω' gleichzusetzen. Die Schildkröte liest dieses Wort und führt für jedes Symbol eine Bewegungen im Raum aus.

Im Folgenden wird ein konkretes Beispiel gezeigt, wie Bäume mithilfe eines parametrischen L-Systems erzeugt werden können. Das Beispiel ist aus dem Buch von Prusinkiewicz, Lindenmayer, Hanan u. a. [8] und geht auf Honda [3] zurück.

Die unten gelisteten Parameter werden mit dem nachfolgend definierten L-System kombiniert und ergeben mit den in Tabelle 2.1 aufgelisteten Turtle-Interpretationen die Grafiken wie dargestellt in Abbildung 2.4b.

L-System

$$
\begin{aligned}
V &= \{!, F, -, +, /, \&, [,], A, B, C\} \\
\Sigma &= \{r_1, r_2, a_0, a_2, d, w_r\} \\
\omega &= \{A\} \\
P &= \left\{ \begin{array}{lcl}
A(l,w) & \rightarrow & !(w)F(l)[\&(a_0)B(l*r_2, w*w_r)]/(d)A(l*r_1, w*w_r), \\
B(l,w) & \rightarrow & !(w)F(l)[-(a_2)\$C(l*r_2, w*w_r)]C(l*r_1, w*w_r), \\
C(l,w) & \rightarrow & !(w)F(l)[+(a_2)\$B(l*r_2, w*w_r)]B(l*r_1, w*w_r)
\end{array} \right\}
\end{aligned}
$$

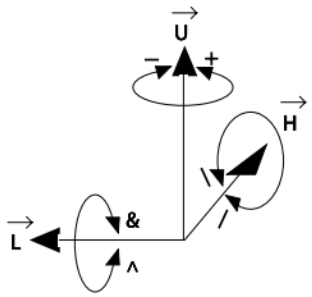
Parameter

- n = 10 (Anzahl an Iterationen)
- r_1 = 0.9 (Ast)
- r_2 = 0.6
- a_0 = 45
- a_2 = 45
- d = 137.5
- w_r = 0.707

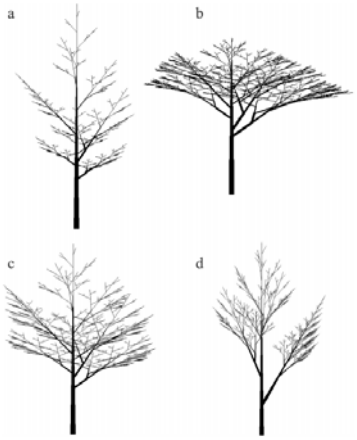
Turtle-Interpretationen

Symbol $\in$ V	Interpretation
!(α)	setzt die Linienstärke auf α
F(α)	geht um α nach vorn und zeichnet dabei eine Linie
-(α)	drehe um $\overrightarrow{U}$ nach links
+(α)	drehe um $\overrightarrow{U}$ nach rechts
/(α)	rolle um $\overrightarrow{H}$ im Uhrzeigersinn
&(α)	drehe um $\overrightarrow{L}$ nach unten
[	lege Interpreterstatus auf den Stack
]	entferne Interpreterstatus vom Stack

Tabelle 2.1.: Turtle-Interpretation



(a) Orientierung des Turtleinterprters [8]



(b) Generierte Bäume mit obigen L-System [8]

Abbildung 2.4.

2.4. Distanzfelder

Ein Distanzfeld ist neben dem Dreiecksnetz eine weitere Möglichkeit, ein Objekt O im dreidimensionalen Raum zu repräsentieren. Ein Dreiecksnetz von O ist eine explizite Beschreibung. Das Distanzfeld ist eine implizite Form der Beschreibung von Körpern und Oberflächen. Das Werk von Osher und Fedkiw [18] liefert dazu im ersten Teil ausführliche Erklärungen zur impliziten Darstellung von Oberflächen und Körpern. Der Vorteil gegenüber der expliziten Darstellung ist, dass Mengenoperationen verwendet werden können, um Grundformen miteinander zu kombinieren. Man kann so beliebig komplexe Körper erzeugen. Es wird für die folgenden Erklärungen der $\mathbb{R}^n$ betrachtet.

$$O \subset \mathbb{R}^n \tag{2.8}$$

O ist eine Teilmenge des Raums $\mathbb{R}^n$. Um ein Distanzfeld d_0 zum Objekt O zu definieren, wird ein geeignetes Abstandsmaß $\| \, . \, \|$ eingeführt.

$$d_0 : \mathbb{R}^n \to \mathbb{R} \tag{2.9}$$

$$d_0(x) : min \parallel x - p \parallel | p \in O \tag{2.10}$$

Aus 2.10 wird ersichtlich, dass die kleinste Distanz zum Objekt O berechnet wird. d_0 ist die allgemeine Distanzfunktion für O im $\mathbb{R}^n$. Daraus erschließt sich, dass alle Punkte des Raums in O die Entfernung 0 haben und umgekehrt alle Punkte außerhalb von O > 0 sein müssen.

2.4.1. Signierte Distanzfunktion

Ein Sonderfall der Distanzfunktion ist die signierte Distanzfunktion, sie unterscheidet ein Objekt in Inneres und Äußeres und ist für die Verarbeitung im Marching-Cubes-Algorithmus von Bedeutung.

$$s(x) = \begin{cases} -d(x), fallsx \in O^i \\ d(x) \end{cases} \tag{2.11}$$

Wobei O^i das Innere von O definiert.

Dadurch ergibt sich, dass in dem Fall für alle x, für die $s(x) = 0$ ist, die Oberfläche des implizit beschriebenen Körpers definiert ist. Dies macht sich der Marching-Cubes-Algorithmus zunutze, um aus der implizit beschriebenen Form eines Körpers die Oberfläche zu extrahieren.

Ein einfaches Beispiel für die Beschreibung eines Körpers ist die Kugel. Die signierte Distanzfunktion dafür sieht so aus:

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r = laenge(x, y, z) - r \tag{2.12}$$

Mit der obigen Funktion 2.12 wird der Abstand zur Kugel bestimmt (siehe Abb. 2.5). Außerhalb ist der Wert der Distanzfunktion positiv, innerhalb negativ. Null auf der Oberfläche.

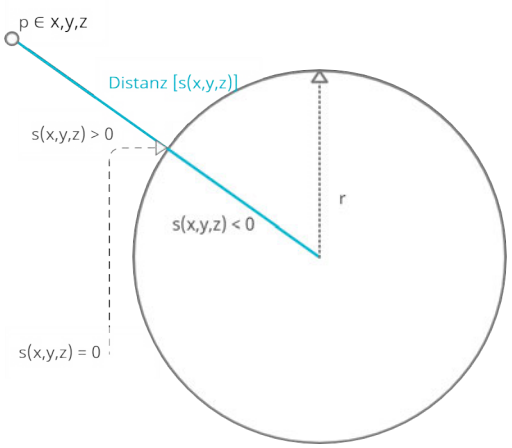


Abbildung 2.5.: Visualisierte Distanzfunktion der Kugel (Quelle: Eigene Darstellung)

2.4.2. Operationen

Um komplexere Objekte implizit zu beschreiben, wie z.B. in dieser Arbeit Bäume, können Mengenoperationen ausgeführt werden. Da ein komplexer Körper durch eine Menge von einfachen Körpern angenähert werden kann, sind die folgenden Mengenoperationen ein geeignetes Mittel. Die Vereinigung, die Differenz und der Schnitt werden in Abbildung 2.6 verdeutlicht.

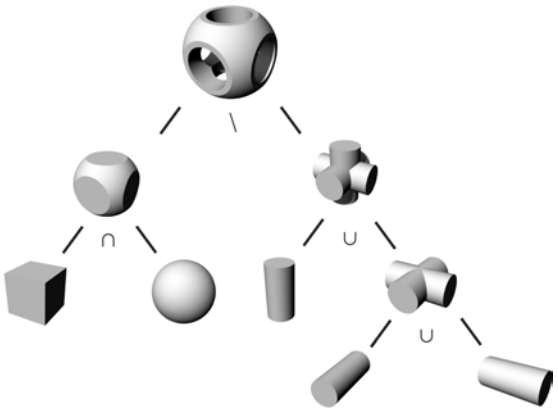


Abbildung 2.6.: Mengenoperationen mit primitiven Körpern (Quelle: Wassermann, Kollmannsberger, Bog u. a. [28])

Verschmelzung

Eine weitere für diese Arbeit wichtige Operation ist die Verschmelzung (siehe Abb. 2.7). Dadurch erreicht man, dass die Oberflächen der Körper glatter ineinander übergehen. Pasko und Savchenko [9] haben dafür die folgende Formel vorgestellt.

$$\bigcup(f_1, f_2) = \frac{1}{1+\alpha}(f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2}) \tag{2.13}$$

Die Parameter f_1, f_2 sind die Funktionswerte der zu vereinenden Objekte. α steuert die Intensität der Verschmelzung.

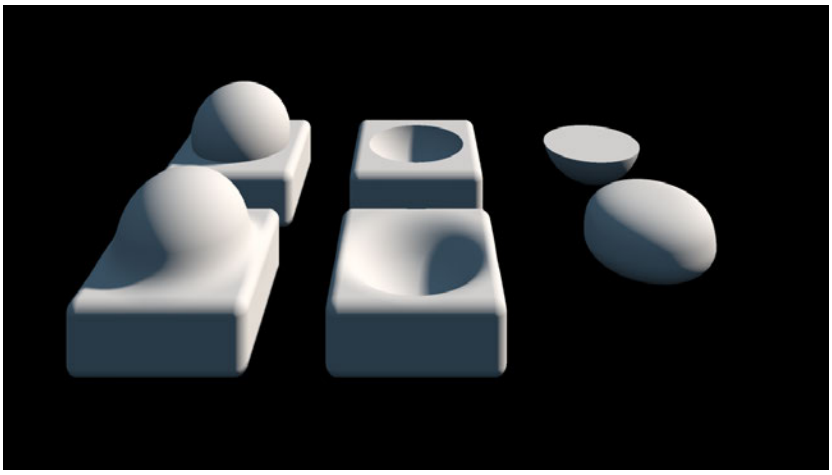


Abbildung 2.7.: Verschmelzen von Objekten (vordere Reihe links) vs. Vereinigung (hintere Reihe links) (Quelle: Shadertoy[2])

Um implizit beschriebene Körper in eine für die Grafikhardware verarbeitbare Form zu bringen, muss für gewöhnlich ein Dreiecksnetz vorliegen. Der folgend beschriebene Algorithmus erzeugt ein Dreiecksnetz mithilfe von Distanzfunktionen.

2.5. Marching-Cubes-Algorithmus

Der Marching-Cubes-Algorithmus (MCA) ist ein von Lorensen und Cline [6] vorgestelltes, bilderzeugendes Verfahren, welches ursprünglich entwickelt wurde, um medizinische Daten, beispielsweise erzeugt von einem CT oder MRT in ein Dreiecksnetz zu überführen. So können z.B. unterschiedliche Bereiche des menschlichen Körpers sichtbar gemacht und dreidimensional begutachtet werden. Im Folgenden wird gezeigt, wie der Algorithmus im Detail funktioniert. Zunächst erfolgt ein Blick auf das grundlegende Prinzip.

2.5.1. Prinzip

Das grundlegende Prinzip des MCA ist, einen Volumendatensatz zunächst in eine kleinere Anzahl an Würfeln aufzuteilen. Diese Würfel (Cubes) werden nun durchschritten ("durchmarschiert"), bis alle Würfel besucht worden sind. Für jeden Würfel wird ein Teil des Dreiecksnetzes generiert, insofern er Teil des anzunähernden Objekts ist. Am Ende wird ein komplettes Dreiecksnetz ausgegeben.

[2](2021). Combination sdf, Adresse: https://www.shadertoy.com/view/lt3BW2 (besucht am 31. 12. 2021).

2.5.2. Datenstrukturen

Für die Eingabe, Verarbeitung und Ausgabe werden Daten in bestimmter Form benötigt. Der folgende Abschnitt beschreibt die benötigten Datenstrukturen.

Eingabe

Die Eingabedaten für den MCA sind ein Volumendatensatz eines beliebigen Objekts und ein Dichtewert ρ. Beispielsweise könnte ein beliebiges Objekt ein menschlicher Kopf sein, welcher mithilfe eines MRTs untersucht werden soll. Der Kopf besitzt Bereiche unterschiedlicher Dichte. Das Knochengewebe ist z.B. deutlich dichter als das Hirngewebe. Diese Informationen über die Dichte werden in einem 3D-Gitter (siehe Abb. 2.8) zu jedem definierten Punkt im Volumen gespeichert. Bestimmt man nun einen Schwellenwert für ρ, definiert man, ob Werte innerhalb oder außerhalb des anzunähernden Körpers liegen sollen.

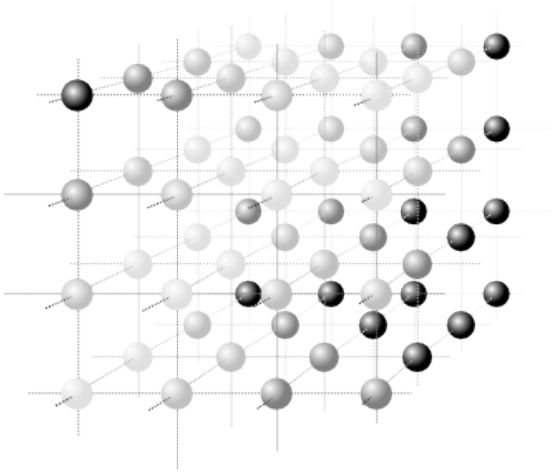


Abbildung 2.8.: Voxelgitter / Volumendatensatz (Quelle: Lorensen und Cline [6])

Verarbeitung

Zur Generierung des Dreiecksnetzes benötigt man eine Triangle Lookup Table (TLT) für die zu erzeugenden Dreiecke. In dieser Tabelle sind $2^8 = 256$ Möglichkeiten gespeichert, auf welche Weise ein Würfel in Innen- und Außenbereich eingeteilt werden kann. Aufgrund von Symmetrien sind aber nur 15 relevant (siehe Abb. 2.9).

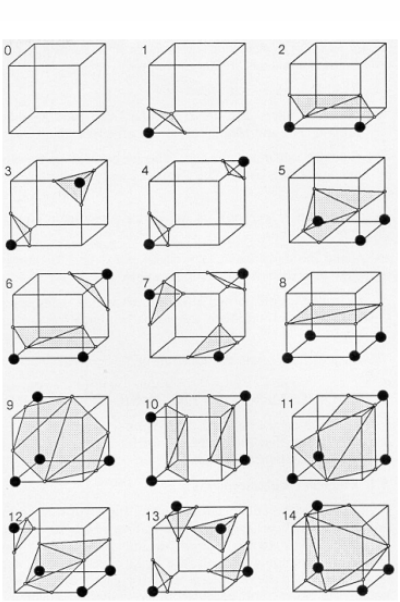


Abbildung 2.9.: Mögliche Dreiecksnetzvarianten für einen Würfel
(Quelle: Lorensen und Cline [6])

Für den Index liefert die TLT eine Liste aller benötigten Dreiecke für einen Würfel. Der Index wird nach der folgenden Abbildung 2.10 bestimmt.

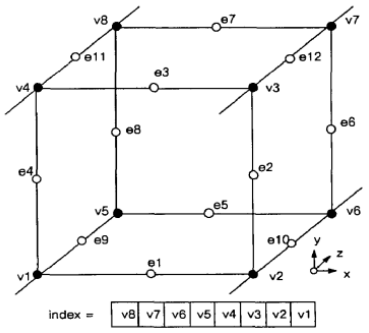


Abbildung 2.10.: Generierung des Index [6]

Ausgabe

Der Algorithums liefert nach Ausführung ein Dreiecksnetz zurück.

2.5.3. Funktionweise

Es wird zunächst im Raum $\mathbb{R}^3$ ein Bereich bestimmt, der in Würfel aufgeteilt wird. Typischerweise sind alle Würfel gleich groß. Die Größe kann beliebig sein. Diese werden nun durchschritten.

Für jeden Würfel werden die Eckpunkte betrachtet. Für jeden Eckpunkt wird nun der Dichtewert ρ aus dem Volumendatensatz ausgelesen und überprüft, ob er kleiner oder größer als der Schwellenwert ist. Ist $\rho < 0$, dann wird dies als 0 interpretiert. Ist $\rho > 0$ als 1. Daraus resultieren 8 Werte $\in 0, 1$. Aus diesen Werten wird der Index für die TLT bestimmt (siehe Abb. 2.10).

Für diese Arbeit gibt es keinen vorher existenten Volumendatensatz, d.h. es wird für jeden Eckpunkt die Signierte Distanzfunktion $s(x, y, z)$ des implizit beschriebenen Körpers aufgerufen. Der Funktionswert von s wird als ρ angenommen.

Mit dem Index können die entsprechenden Dreiecke aus der TLT in Form einer Kantenliste bestimmt werden. Durch lineare Interpolation werden die genauen Punkte der Dreicke bestimmt.

Im Anschluss können die Dreiecksnormalen bestimmt werden. Dann wird zum nächsten Würfel "marschiert".

3. Stand der Technik

Es gibt viele Anwendungen im Bereich der Generierung von 3D-Bäumen, die meisten mit dem Ziel, realitätsnahe Bäume zu entwerfen. Der Entwurfsprozess in diesen Anwendungen ist zumeist automatisch bis teilautomatisiert. Die Gestaltung ist durch eine Vielzahl an Parametern zu bestimmen. Die technisch aufwendigeren und beeindruckenderen Ergebnisse kommen aus dem kommerziellen Bereich, aber auch freie Software kann mit optisch ansprechenden Ergebnissen aufwarten. Die wissenschaftlichen Arbeiten leisteten die nötige Pionierarbeit und sorgen für Innovationen. Im Folgenden wird ein kleiner Rundumblick über den aktuellen Stand der Technik gegeben.

3.1. Kommerzielle Software

Die technisch fortgeschrittenste Anwendung kommt von der US-amerikanischen Firma IDV mit dem Namen SpeedTree[1]. Sie wird in einem Lizenzmodell in verschiedenen Versionen vertrieben und häufig von der Gaming- sowie der Filmindustrie in Projekten genutzt. Laut Website wurde die Software in der Produktion der Filme ”Avatar”, ”Star Wars” und in Spielen wie ”Assassin’s Creed Valhalla” eingesetzt, um nur einige prominente Beispiele zu nennen. Alternative Anwendungen im kommerziellen Sektor sind zum Beispiel The Grove 3D[2], Xfrog[3] und Plant Factory[4].

3.2. Freie Software

Es gibt inzwischen eine Vielzahl an freien Anwendungen, die sich mit dem Thema der prozeduralen Content-Generierung mit Fokus auf Bäume auseinandersetzen. Ein interessantes

[1](2022). About us, Adresse: https://store.speedtree.com/about-us/ (besucht am 01. 01. 2022).
[2](2022). The grove, Adresse: https://www.thegrove3d.com/ (besucht am 02. 01. 2022).
[3](2022). Xfrog, Adresse: http://xfrog.com/ (besucht am 02. 01. 2022).
[4](2022). Plant factory, Adresse: https://info.e-onsoftware.com/plantfactory/overview (besucht am 02. 01. 2022).

Beispiel ist die Software Arbaro[5], da die Implementierung auf den Artikel von Weber und Penn [10] zurückgeht, und man konkrete Details zur Umsetzung erfährt. Die Autoren halten nicht an den botanischen Prinzipien fest, auf welchen das Wachstum von Pflanzen beruht, sondern verfolgen in ihrer Arbeit einen geometrischen Ansatz, mit dem 3D-Bäume erzeugt werden.

Ein weiteres nennenswertes Beispiel ist das Blender-Plugin Sapling Tree[6]. Es besticht durch seine Integrierbarkeit in das weit verbreitete 3D-Modellierungswerkzeug Blender[7]. Es ist simpel und übersichtlich gestaltet, und man kommt schnell zu optisch ansprechenden und realitätsnahen Ergebnissen. Des Weiteren lassen sich hiermit sogar Bäume animieren.

Ein weiterer Ansatz ist der Tree Generator[8] von Andrew Marsh. Die Software ist im Besonderen an dieser Stelle zu nennen, da diese, anders als die vorangegangen Anwendungen, die Möglichkeit bietet, Bäume semi-automatisiert in einem Low-poly-Stil zu rendern. Die Software ist zudem webbasiert und kann ohne Installation im Browser ausgeführt werden.

3.3. Wissenschaftliche Arbeiten

Blickt man nun auf die Zeit, die in der Wissenschaft seit 1968 und dem Werk von Lindenmayer [2] vergangen ist, sind einige beeinduckende Arbeiten entstanden. Der folgende Abschnitt soll einen Überblick über die Entwicklung in dieser Zeit geben und Arbeiten vorstellen, die ein ähnliches Ziel, wie das hier vorgestellte, verfolgten.

Ein erster Ansatz, Bäume als Grafik darzustellen, gelang Honda [3]. Mithilfe von verzweigenden Linien und Parametern erstellte er 2D-Bäume.

Die Arbeit von Bloomenthal [5] befasst sich mit der Konstruierung eines Ahornbaums als 3D-Modell. Als Basis dient ein “Baumskelett”, das aus vorgegebenen Punkten generiert wird. Die Linien des Skeletts werden durch Splines beschrieben. Die Oberfläche des Baummodells wird so generiert, dass in Abständen Kreise entlang des Splines orthogonal positioniert werden.

[5](2022). Arbaro - tree generation for povray, Adresse: http://arbaro.sourceforge.net/ (besucht am 01. 01. 2022).
[6](2022). Sapling tree gen, Adresse: https://docs.blender.org/manual/en/latest/addons/add_curve/sapling.html (besucht am 02. 01. 2022).
[7](2022). Blender, Adresse: https://docs.blender.org/manual/en/latest/addons/add_curve/sapling.html (besucht am 02. 01. 2022).
[8](2022). Tree generator, Adresse: http://andrewmarsh.com/software/tree3d-web/ (besucht am 02. 01. 2022).

Zwischen einer endlichen Menge an Punkten auf diesen Kreisen werden verbindende Linien gespannt. Ein Oberflächennetz entsteht.

Basierend auf den Regeln der Botanik, gelingt es der Arbeit von De Reffye, Edelin, Françon u. a. [7], realistische Grafiken von Bäumen zu erzeugen. Geometrische Grundformen werden hier benutzt, um die Bäume zu visualisieren. Beeindruckend hierbei ist die Fülle an unterschiedlichen Bäumen, die generiert werden können.

In dem Werk von Prusinkiewicz, Lindenmayer, Hanan u. a. [8] wurde erstmals der Turtleinterpreter eingeführt. Dieser baut auf den Arbeiten von Honda [3] und Lindenmayer [2] auf. Der Interpreter setzt die Symbole aus dem L-System und Parametern in Befehle um, die Baumgrafiken, bestehend aus Linien, erzeugen.

Ein weiterer Ansatz, 3D-Baummodelle zu erzeugen, liefern Weber und Penn [10]. Basierend auf einem geometrischen Grundkörper, dem Kegel, wird zusammen mit Parametern ein Baum erzeugt. Durch Beschneidung wird der Baum in eine vorgegebene Grundform gebracht. Ein weiterer Bestandteil der Arbeit ist ein Algorithmus zur Detailverringerung in Abhängigkeit der Distanz zum Baum.

Lintermann und Deussen [15] verbesserten die realitätsnahe Darstellung von Pflanzen und stellten einen Editor vor, mit dem es einfacher war als zuvor, Pflanzen zu modellieren.

Ein etwas anderer Ansatz, die Struktur und Verzweigungen von Bäumen zu generieren, stellten Runions, Lane und Prusinkiewicz [20] vor. Basierend auf ihren vorherigen Arbeiten stellten sie einen neuen Algorithmus (Space-Colonization-Algorithmus) vor, der innerhalb einer vordefinierten Hülle anziehende Partikel verteilt und daraus schrittweise Verästelungen bildet, die schließlich die Struktur eines Baumes ergeben.

Das von Beneš, Andrysco und Šťava [21] vorgestellte Verfahren kombiniert unter anderem die Arbeiten von Runions, Lane und Prusinkiewicz [20] und De Reffye, Edelin, Françon u. a. [7] für das darin vorgestellte Wachstumsmodell von Bäumen und ermöglicht es, ein Ökosystem zu erzeugen, um dessen Ressourcen (Raum, Licht) mehrere Pflanzen konkurrieren.

Das Einpassen bestehender “Baumskelette” als Dateneingabe behandelt die Arbeit von Pirk, Stava, Kratt u. a. [23]. Das Ergebnis sind Bäume, die sich selbst in ihre Umgebung einpassen.

Dazu werden bestehende “Baumskelette” verformt und anschließend als realitätsnahes 3D-Modell gerendert. Das Verfahren ist performanter als Modelle, die auf L-Systemen oder dem Space-Colonization-Algorithmus beruhen.

Das realitätsnahe Simulieren von Baumwachstum in Abhängigkeit von äußeren Umweltfaktoren ist in der Arbeit von Yi, Li, Guo u. a. [30] umgesetzt worden. Es werden Gleichungen vorgestellt, die das realitätsnahe Wachstum von Bäumen beschreiben. Mithilfe dieser Funktionen werden Bäume erzeugt, die untereinander um Raum konkurrieren können und von Hindernissen in ihrem Wachstum gebremst werden.

Die Arbeit von Liu, Guo, Benes u. a. [37] umfasst einen anderen Weg der Baummodellierung. Aus bereitgestellten Bildern von echten Bäumen werden Punktwolken erzeugt, die mithilfe eines neuronalen Netzes in ein 3D-Modell des fotografierten Baumes überführt werden. Der bildbasierte Ansatz ist aber nicht neu. In der Arbeit von Quan, Tan, Zeng u. a. [19] wurden bereits 3D-Pflanzen-Modelle auf Basis von Fotografien erzeugt.

Abschließend betrachtet gibt es bis heute zwei grundsätzliche Herangehensweisen. Zum einen den Ansatz, Bäume und Pflanzen mithilfe von prozeduraler Content-Generierung zu erzeugen. Zum anderen anhand realer Bilder Modelle von Bäumen und Pflanzen zu generieren. Diese Arbeit lässt sich dem ersten Ansatz zuordnen.

4. Konzept

Das folgende Kapitel befasst sich mit dem Konzept zur Entwicklung des Low-Poly-Tree-Generator (LPTG).

4.1. Idee

Die Idee dieser Arbeit ist es, 3D-Modelle von Bäumen, ähnlich wie sie in der Natur zu finden sind, zu erzeugen. Der Detailgrad der Modelle soll dabei grob gehalten, im Low-poly-Stil sein. Dazu soll eine Anwendung entwickelt werden, die diese Aufgabe erfüllt.

Die Benutzeroberfläche des LPTG soll es dem Anwender ermöglichen, durch Änderung von Parametern und L-Systemen unterschiedliche Bäume zu erstellen. In Form eines Dreiecksnetzes sollen 3D-Modelle verschiedener Baumarten erzeugt werden, wie z.B. Tannen, Eichen oder auch Palmen. Andere Baumarten sind ebenfalls denkbar.

Um diese Aufgabe zu lösen, werden sich die in Kapitel 2 beschriebenen Konzepte zunutze gemacht, diese miteinander verknüpft und zu einer ”Baumsynthese” zusammengeführt, wie bereits in der Einleitung in Abbildung 1.2 gezeigt.

Die grundsätzliche Idee ist es, die folgend gezeigte Verarbeitungskette zu nutzen (siehe Abb. 4.1). Die Aufgabe wird so in kleinere Teilprobleme zerlegt und beherrschbarer (Teile-und-herrsche-Prinzip). Zudem lässt sich darauf aufbauend eine Architektur ableiten, die in Kapitel 4.2 vorgestellt wird.

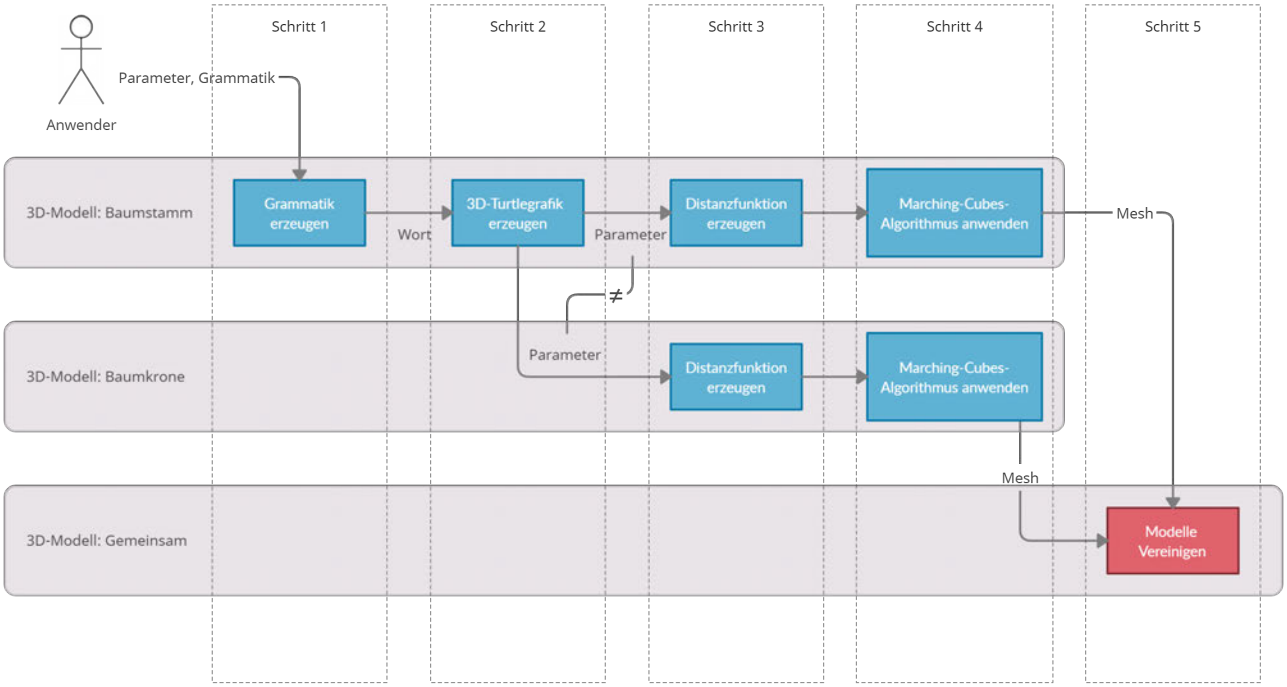


Abbildung 4.1.: Verarbeitungskette, um ein 3D-Modell eines Baumes zu erzeugen. (Quelle: Eigene Darstellung)

4.2. Architektur

Die Abbildung 4.2 zeigt die Architektur des LPTG. Wie zu erkennen ist, werden die einzelnen Funktionen in einzelne Komponenten mit klaren Zuständigkeiten gekapselt. Diese bilden untereinander Schnittstellen und Abhängigkeiten. Die Beschreibung der Abhängigkeiten, Komponenten und Schnittstellen wird nachfolgend aufgeführt.

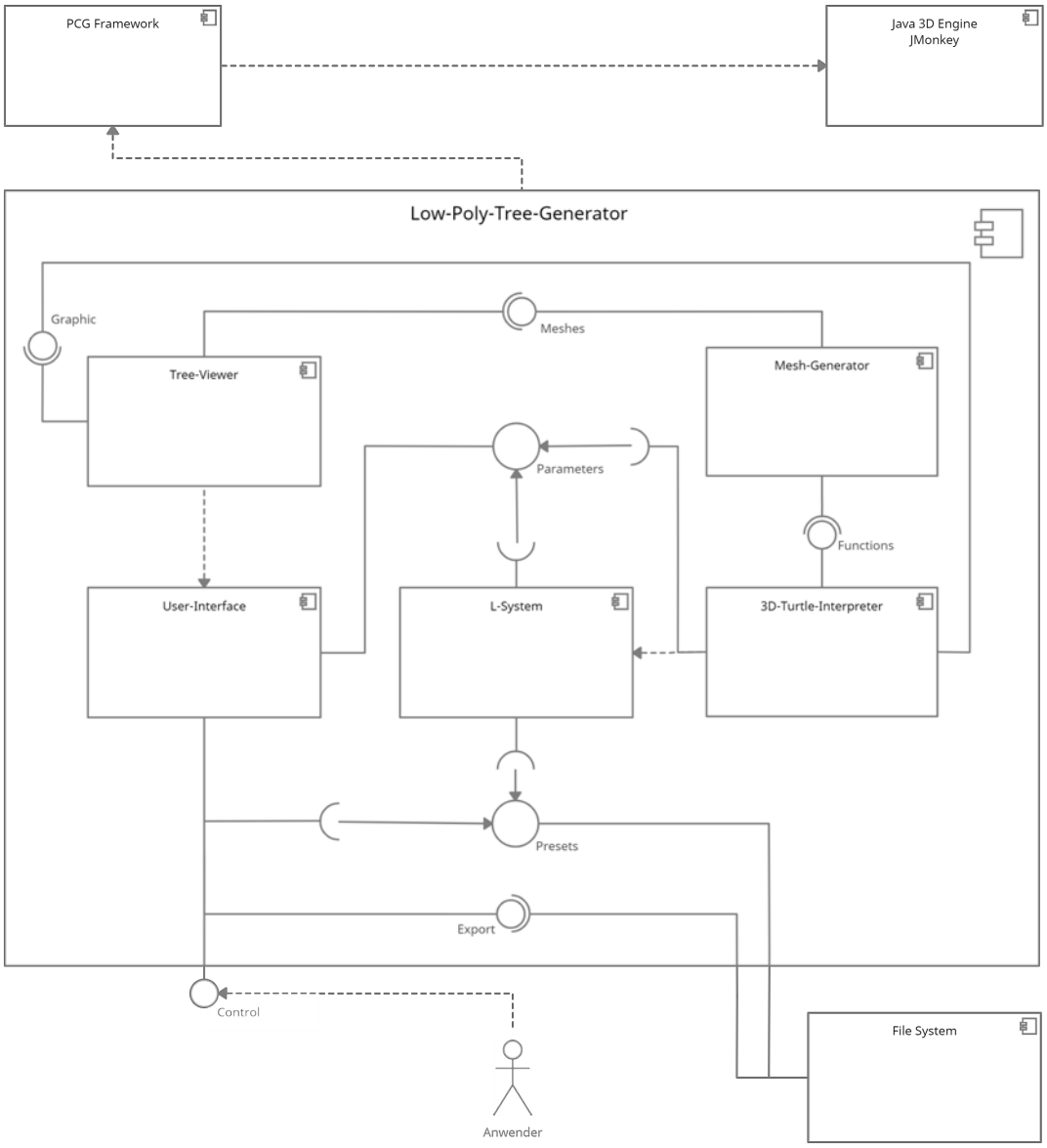


Abbildung 4.2.: Komponentendiagramm der Software-Architektur. (Quelle: Eigene Darstellung)

4.2.1. Komponenten

Komponente	Aufgabe
LPTG	Gesamte Anwendung, die es ermöglicht, 3D-Modelle von Bäumen zu erzeugen. Benutzung erfolgt durch den Anwender.
User-Interface (UI)	Bietet die Schnittstelle zur Verwendung durch den Anwender. Parameter können eingestellt, generierte Bäume angezeigt, Grammatiken verändert und Parameter-Presets geladen werden.
Tree-Viewer	Ist Teil des UI und ist für die Anzeige des generierten Baums verantwortlich. 3D-Modell des Baums, der Krone und der 3D-Turtlegrafik (3DTG) können wahlweise angezeigt werden.
L-System	Erzeugt mithilfe einer Grammatik ein abgeleitetes Wort, welches vom 3DTI benötigt wird.
3D-Turtle-Interpreter (3DTI)	Interpretiert das vom L-System übergebene Wort und Parameter aus der UI in eine 3DTG und Distanzfunktion (DF) für Baumkrone und Baumstamm.
Mesh Generator	Hier wird mithilfe der DF und dem MCA je ein Mesh für Baumkrone und Baumstamm erzeugt, die vom Tree-Viewer angezeigt werden können.
File System	Das File System ist eine Komponente des Betriebssystems und wird für das Laden von Grammatik-Dateien und UI-Presets benötigt.

Tabelle 4.1.: Aufgaben der Komponenten

4.2.2. Abhängigkeiten

Die Software basiert auf dem PCG-Framework, welches zu Beginn dieser Arbeit zur Verfügung gestellt wurde. Dieses bindet wiederum JMonkey ein (siehe Abb. 4.2). JMonkey bietet Funktionen, die es dem Anwender erleichtern, Grafiken, in dieser Arbeit insbesondere Dreiecksnetze, auf den Bildschirm zu zeichnen.

Weitere Abhängigkeiten bestehen innerhalb der Applikation sowie zwischen der Applikation und dem Anwender. Konkret zwischen der L-System-Komponente und dem 3D-Turtle-Interpreter (3DTI). Der 3DTI kann ohne das abgeleitete Wort der L-System-Komponente nicht arbeiten. Ebenso sind die Parameter-Einstellungen von dem Anwender abhängig, der das UI bedient.

4.2.3. Schnittstellen

Schnittstelle	**Aufgabe**	**Daten**
Control	Steuerung der UI vom Anwender	Menschlich gesteuerte Aktionen
Presets	Einlesen von Grammatik- und Preset-Dateien	Grammatikdateien, Presetdateien
Parameters	Parameteraustausch an beteiligte Komponente	Parameter, Grammatik
Functions	Bereitstellung von Distanzfunktionen, die den Baum beschreiben	Vereinigte Distanzfunktion (DF)
Graphic	Bereitstellung von Vektoren zur Erzeugung einer 3DTG	3D-Vektoren
Mesh	Bereitstellung von Dreiecksnetzen des Baumstamms und der Krone	Dreiecksnetz
Export	Exportieren der erzeugten Meshdaten in ein standardisiertes Dateiformat	Dreiecksnetz im OBJ-Format

Tabelle 4.2.: Schnittstellen der Komponenten

4.3. Parameter

Parameter bilden im Prozess der ”Baumsynthese” einen essentiellen Bestandteil und werden benötigt, um auf die Form und das Aussehen eines zu generierenden Baumes Einfluss zu nehmen. Die Parameter können über das UI vom Anwender verändert und eingestellt werden. Die Tabelle 4.3 listet alle Parameter auf, erläutert ihre jeweilige Funktion und den Effekt, den der jeweilige Parameter auf das Aussehen des Baumes auslöst.

Parameter	**Funktion**	**Effekt**	**Daten-Typ**	**Wertebereich**
P_{Tres}	Einstellung der Auflösung des MCA für den Baumstamm	Einfluss auf die Anzahl der Dreiecke im Mesh.	Integer	10 - 200
P_{Cres}	Einstellung der Auflösung des MCA für die Baumkrone	wie P_{Tres}	wie P_{Tres}	wie P_{Tres}
P_{Bl}	Astlänge, Länge einer Linie in der 3DTG	Einfluss auf die Größe des Baumes	Float	0 - 0.5
P_{Bw}	Astumfang	Einfluss auf den Durchmesser des Stamms und der Äste	Float	0 - 0.5
P_{r1}	Reduktionsfaktor des Stammumfangs	Stamm-verjüngung	Float	0 - 1
P_{r2}	Reduktionsfaktor des Astumfangs	Astverjüngung	Float	0 -1
P_{Sdia}	Durchmesser der Kugelgrundformen	Ändert Aussehen der Baumkrone	Float	0 - 0.5

P_{Soff}	Offset der Baumkrone in Y-Richtung	Ändert Aussehen der Baumkrone	Float	0 - 0.5
P_{Cfus}	Steuerung der Vereinigungsfunktion der Baumkrone	Ändert Aussehen der Baumkrone	Float	$10^{-3} - 10^{-1}$
P_{Tgen}	Baumgeneration, Ableitungen der Grammtik	Baumgröße, Anzahl der Verzweigungen	Integer	1 - 15
P_{α}	Rotationsschrittweite um X-Achse	Aussehen des Baumstamms und Verzweigungen	Integer	0 - 360
P_{β}	Rotationsschrittweite um Y-Achse	Aussehen des Baumstamms und Verzweigungen	Integer	0 - 360

Tabelle 4.3.: Parameter des Low-Poly-Tree-Generator (LPTG)

4.4. UI / Benutzeroberfläche

Die Benutzeroberfläche (User-Interface (UI)) der Anwendung gibt dem Anwender die Möglichkeit, einen Baum zu generieren und anschließend zu exportieren. Die Abbildung 4.3 zeigt die Benutzeroberfläche der Anwendung.

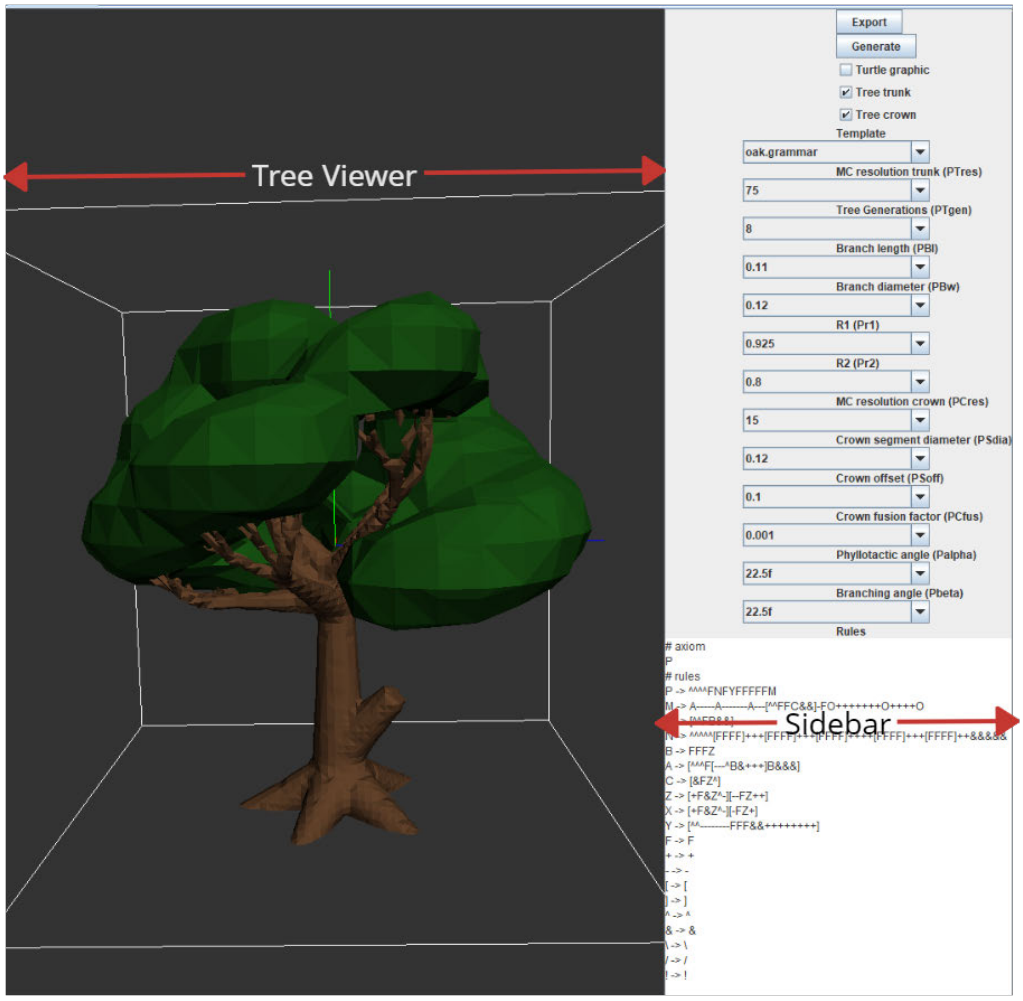


Abbildung 4.3.: Benutzeroberfläche der Anwendung. (Quelle: Eigene Darstellung)

Die Benutzeroberfläche teilt sich in zwei Bereiche: auf der linken Seite der ”Tree-Viewer” und auf der rechten Seite die ”Sidebar”. Der ”Tree-Viewer” zeigt den generierten Baum. Die Sidebar

enthält alle zuvor in Kapitel 4.3 beschriebenen Parameter und bietet per Dropdown-Menü die Möglichkeit, dessen Werte zu ändern. Zusätzlich befinden sich weitere Elemente des UI in der ”Sidebar”, die in der Tabelle 4.4 erläutert werden.

UI-Element	**Funktion**	**Typ**
Export	Der Export des generierten Baums in eine OBJ-Datei wird per Klick ausgeführt.	Button
Generate	Die Generierung des Baums wird per Klick gestartet.	Button
Template	Auswahl einer Baumvorlage	Button
Turtle graphic	Schaltet die Anzeige des Baumskeletts ein oder aus.	Checkbox
Tree trunk	Schaltet die Anzeige des Baumstamm-Meshes ein oder aus.	Checkbox
Tree crown	Schaltet die Anzeige des Baumkronen-Meshes ein oder aus.	Checkbox
Rules	Hierüber kann das geladene Grammatik-Preset verändert werden. Regeln können hinzugefügt, entfernt oder geändert werden.	Textfield

Tabelle 4.4.: Weitere Elemente des UI

4.5. Funktionsweise

Greift man die Abbildung 4.1 unter Berücksichtigung der Architektur erneut auf, lassen sich für die dort abgebildeten Schritte die vorgestellten Komponenten zuordnen.

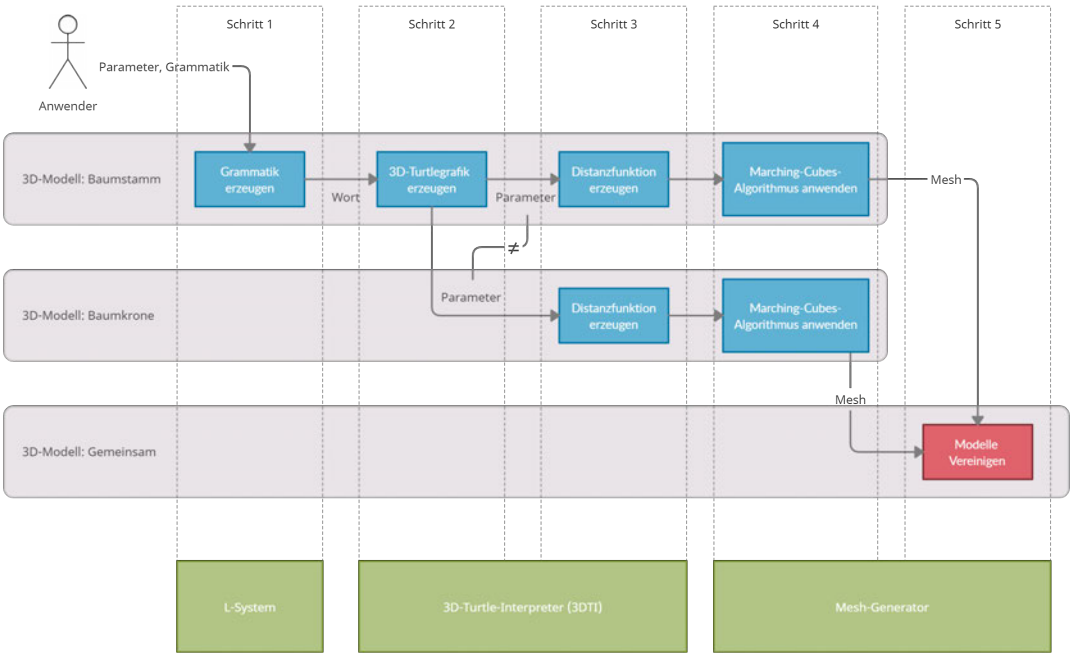


Abbildung 4.4.: Verarbeitungskette mit Komponenten. (Quelle: Eigene Darstellung)

Die in Abbildung 4.4 gezeigte Konzeptgrafik bildet die Basis für die in Kapitel 5 gezeigte Umsetzung. Nachdem Architektur und Zuständigkeiten näher beleuchtet wurden, wird die Funktionsweise der Verarbeitungskette näher beschrieben.

Grammatik erzeugen (Schritt 1)

Es wird ein L-System vom Anwender entworfen oder aus einem Baum-Template gewählt. Mit diesem L-System lässt sich ein Wort ableiten.

Grafik interpretieren (Schritt 2)

Auf Basis des Wortes aus Schritt 1 lässt sich mithilfe des 3DTI eine 3DTG, bestehend aus Linien, erzeugen. Parameter legen dabei Winkel und Länge der Linien fest. Das Resultat ist eine baumähnliche Grafik, ein “Skelett” des Baumes.

Distanzfunktionen erzeugen (Schritt 3)

Um aus dem “Skelett” ein Dreiecksnetz zu erzeugen, wird zunächst das 3D-Modell des Baumstamms mit einer Distanzfunktion (DF) beschrieben. Ziel ist, um jede Linie des “Skeletts” einen geometrischen Körper zu bilden und diesen durch eine Signierte Distanzfunktion zu beschreiben. So würde sich z.B. ein gekappter Kegel (siehe Kapitel 5.3) anbieten, da sich dieser ähnlich wie ein Ast von Anfang bis Ende verjüngt. Da das “Skelett” aus mehreren Linien besteht, resultieren ebenso viele DF. Aus allen resultierenden DF bildet man durch Vereinigung (siehe Kapitel 2.4.2) eine gemeinsame DF.

Für die Baumkrone geht man ähnlich vor, doch statt um jede Linie der 3DTG einen gekappten Kegel zu bilden, wird nun um den Endpunkt der äußeren Linien eine Kugel gebildet. Diese wird erneut durch eine Signierte Distanzfunktion beschrieben. Da das “Skelett” mehrere ”Astenden” besitzt, resultieren mehrere Kugeln und somit mehrere DF. Diese werden wieder zu einer gemeinsamen Funktion vereinigt, welche den Körper der Baumkrone beschreibt. Durch die Überlappungen einzelner Kugeln entsteht der Eindruck einer zusammenhängenden Krone. Um die Kugeln zusammenhängender und glatter wirken zu lassen, wird eine Verschmelzung durchgeführt.

Meshes erzeugen (Schritt 4)

Nach den vorangegangenen Schritten sind zwei DF entstanden, die den Baumstamm und die Baumkrone beschreiben. Der Marching-Cubes-Algorithmus arbeitet mit diesen Funktionen und erzeugt daraus jeweils ein Dreiecksnetz für Baumstamm und Baumkrone.

Meshes vereinigen (Schritt 5)

Beide Dreiecksnetze werden in diesem Schritt zu einem Dreiecksnetz vereinigt. Das gewünschte Endresultat, das 3D-Baummodell, ist erreicht.

5. Programmablauf

Nach der Vorstellung des Konzepts wird sich dieses Kapitel mit der Lösung und Umsetzung der oben vorgestellten Anwendung befassen. Dabei werden die für die Funktionalität der Anwendung wichtigsten Implementierungsdetails vorgestellt und erläutert. Die Umsetzung erfolgt in Java, weil das PCG-Framework und der zur Verfügung gestellte Code in dieser Sprache umgesetzt ist.

5.1. Wort ableiten

Ohne das von der L-System-Komponente (siehe Abb. 4.2) erzeugte Wort, funktioniert keiner der weiteren Verarbeitungsschritte (siehe Abb. 4.4) der Anwendung. Deshalb soll im Folgenden diese Komponente näher beleuchtet und deren Umsetzung gezeigt werden.

Die Grundlage für die Implementierung der L-System-Komponente (siehe Anhang Abb. A.1), welche für den 2D-Fall funktioniert, wurde aus dem zur Verfügung gestellten PCG-Framework übernommen. Das 2D-taugliche Modell musste nun erweitert werden, um es für den 3D-Turtle-Interpreter (3DTI) vorzubereiten. Dafür wurde die Grammar-Klasse der bestehenden Implementierung abgeleitet und das dort definierte Alphabet um zusätzliche Symbole erweitert.

Das erweiterte Alphabet (2.2) besteht aus den folgenden Symbolen.

$$V = \{F, -, +, /, \backslash, \wedge, \&, [,], A, B, C, M, N, O, P, Y, Z, \} \tag{5.1}$$

Die Produktionsregeln werden über das UI, genauer das Textfeld "Rules"(siehe Tabelle 4.1) bereitgestellt. Die Regeln werden in dem Textfeld nach dem folgenden Regelmuster 1 festgelegt, um korrekt verarbeitet werden zu können.

In Zeile 2 wird das Startsymbol ω, das Axiom, festgelegt. Ab Zeile 4 bis 14 folgen die Produktionregeln P. Das Muster 1 zeigt ein Beispiel für eine einfache Verästelungsstruktur. Botanisch

als sympodisch, genauer als Dichasium, bezeichnet. Damit lässt sich im Weiteren ein simpler Baum erzeugen. Dies ist aufgrund der Einfachheit ein geeignetes Beispiel für die folgenden Unterkapitel. Am Ende des Kapitels wird ein Baum basierend auf diesen Regeln entstanden sein.

Muster 1 Regelaufbau

```
1: #axiom
2: A
3: #rules
4: A -> ∧∧FB
5: B -> C+++C−−−
6: C -> [∧F&B]
7: F -> F
8: + -> +
9: - -> -
10: [ -> [
11: ] -> ]
12: ∧ -> ∧
13: & -> &
```

Nach dem Regelwerk aus Muster 1 lässt sich nun in mehreren Iterationen das folgende Wort ω' ableiten. Fünf Iterationen ($n = 5$) sollen genügen, damit das Wort für dieses Beispiel nicht unnötig lang und die resultierende 3D-Turtlegrafik (3DTG) nicht zu komplex wird.

$$\omega' = \{\wedge\wedge F[\wedge F\&[\wedge F\&B]+++[\wedge F\&B]---]+++[\wedge F\&[\wedge F\&B]+++[\wedge F\&B]---]---\} \tag{5.2}$$

Das in der Definition 5.2 gezeigte Wort besteht nach der 5. Ableitung aus 54 Symbolen. Diese werden im nächsten Schritt in Kapitel 5.2 mit der Komponente 3D-Turtle-Interpreter (3DTI) verarbeitet. Die Symbole werden dabei als Liste des Typs Symbol (siehe Klasse Symbol in Anhang A.1) übergeben. Das Ergebnis wird dann eine 3DTG sein.

5.2. Turtlegrafik interpretieren

Die Komponente 3DTI ist nach dem Klassendiagramm (siehe Anhang Abb. A.2) umgesetzt. Der Kern der Klasse ”TurtleInterpreter3D” besteht darin, dass das übergebene Wort ω' als Symbol-Array durchlaufen und für jedes terminale Symbol eine Aktion ausgeführt wird. Terminale Symbole sind alle Sonderzeichen und das Symbol F im Alphabet V. Nicht terminale Symbole sind übrigens die Buchstaben. Sie können gewöhnlich durch andere Regeln aus P ersetzt werden.

In Abbildung 5.1 bis 5.4 wird in vier Schritten erläutert, wie aus dem Wort ω' die 3DTG abgeleitet wird. Dabei wird nur die linke Seite des Baumes betrachtet, die von einem gestrichelten Kasten umrahmt ist. Da sich die rechte Seite nach dem gleichen Schema ableitet, kann so die Erläuterung des Beispiels kurz gehalten werden. Der ebenfalls eingerahmte Teil von ω' zeigt die dafür relevanten Symbole.

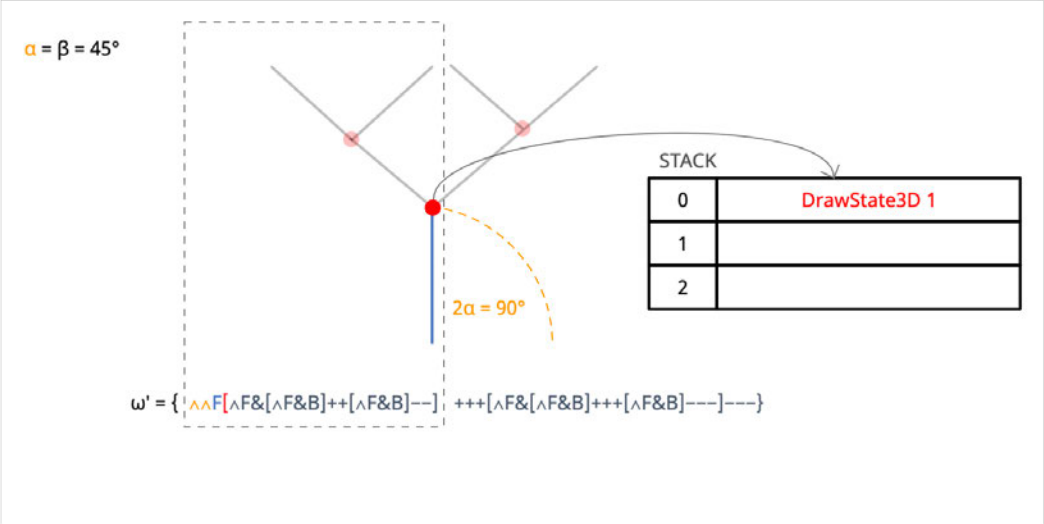


Abbildung 5.1.: Ableitung der Turtlegrafik. Schritt 1. (Quelle: Eigene Darstellung)

Die Symbole $\wedge$, orange eingefärbt, zu Beginn von ω' ändern den Wert der Klassenvariable ”branchingAngle” der Instanz des ”TurtleInterpreter3D” um $+\alpha$. Dieser Wert wird über das UI durch den Parameter P_α (siehe Tabelle 4.3) eingestellt. Der Initialwert ist 0. Als nächstes folgt das Symbol F in ω'. Das Symbol wird als Linie (blau in Abb. 5.1) interpretiert. Die Länge der Linie wird vom Parameter P_{Bl} (siehe Tabelle 4.3) bestimmt.

Die erste Linie der 3DTG, dargestellt in Abbildung 5.1 in Blau, kann nun zusammen mit dem Winkel α (P_α), als gestrichelter oranger Viertelkreis abgebildet, gezeichnet werden. Die Position, dargestellt als roter Punkt, wird nun in einer Instanz von ”DrawState3D” (siehe Anhang Abb. A.2) auf den Stack gelegt und so gespeichert.

Weiter in Abbildung 5.2 wird Schritt 2 gezeigt. Dazu betrachtet man die nächsten farblich markierten Zeichen von ω'. Die Verarbeitung folgt dem Schema aus Schritt 1. Neu in dieser Abbildung ist das Symbol ”&”. Dieses vermindert den Wert von ”branchingAngle” um α, dargestellt in Grün. Die aktuelle Position (roter Punkt) wird in einer neuen Instanz von ”DrawState3D” auf den Stack gelegt.

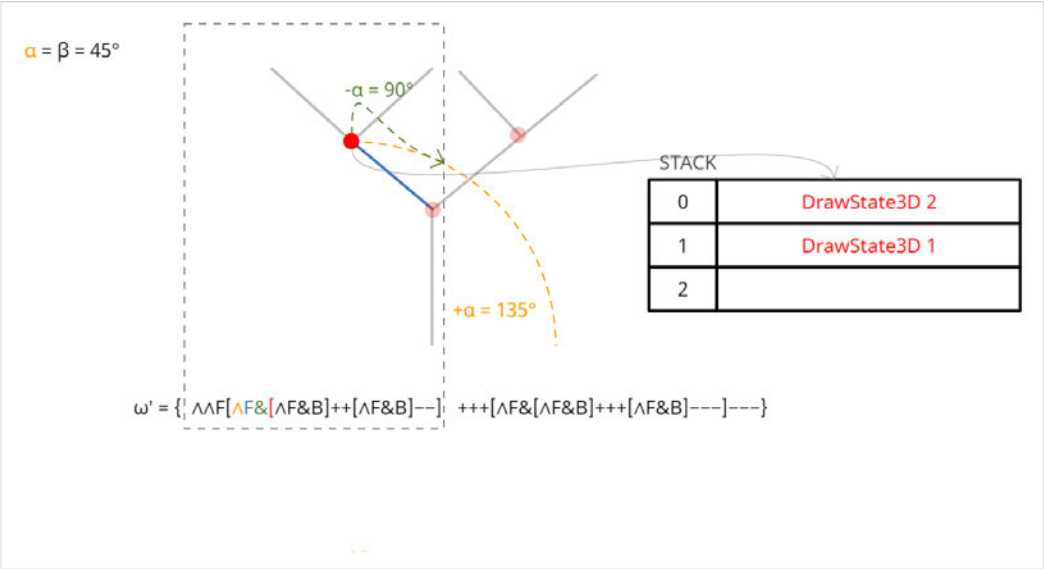


Abbildung 5.2.: Ableitung der Turtlegrafik. Schritt 2. (Quelle: Eigene Darstellung)

Im nächsten Schritt, gezeigt in Abbildung 5.3, wird der Stack verringert. Zuvor wurde eine neue Linie (Symbol F) und die neue Position (roter Punkt, ”DrawState3D” 3) auf den Stack gelegt.

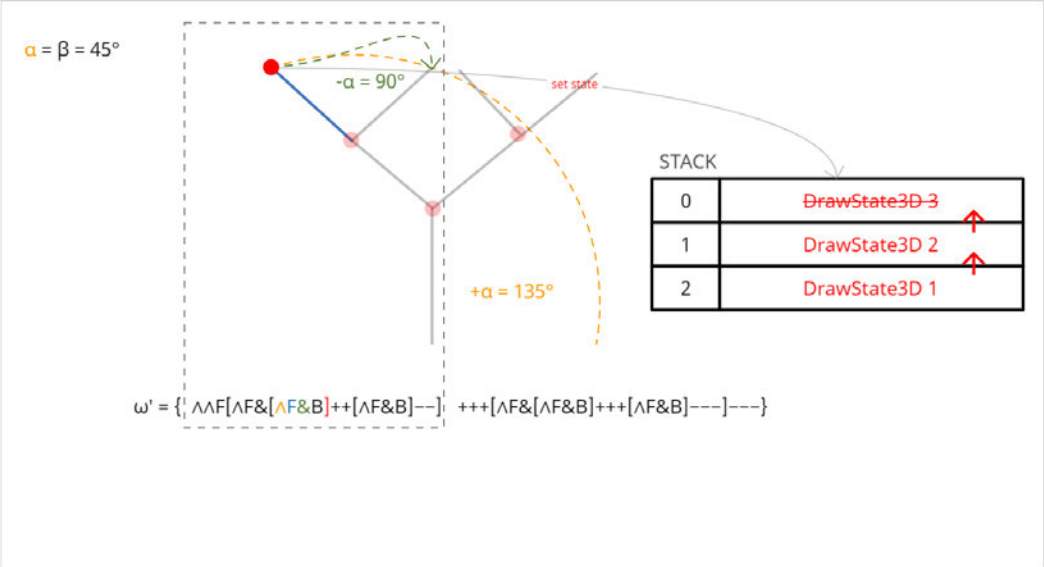


Abbildung 5.3.: Ableitung der Turtlegrafik. Schritt 3. (Quelle: Eigene Darstellung)

Der ”DrawState3D 3” wird vom Stack genommen. Der ”DrawState3D 2” rutscht nun wieder an die oberste Stelle und bildet im letzten Schritt 4 die Ausgangsposition für den 3DTG. Die weiteren farblich dargestellten Aktionen verhalten sich so wie in den vorherigen Schritten beschrieben.

In Schritt 4, dargestellt durch die nächste Abbildung 5.4, werden zwei neue Aktionen zusammen mit dem Winkel β eingeführt. Die Schrittweite des Winkels wird über den Parameter P_β (siehe Tabelle 4.3) bestimmt. Nun sind an der Rotation der Linie zwei Winkel beteiligt. Die Rotation ist durch den pink-gestrichelten Pfeil in der Abbildung dargestellt.

Das Symbol + in ω' bewirkt eine Erhöhung des Winkels β um 45°. Da das Symbol zweimal in ω' vorkommt, ist der Wert von β 90°. β wird in der Klasse "TurtleInterpreter3D" durch das Attribut "phyllotactingAngle" abgebildet.

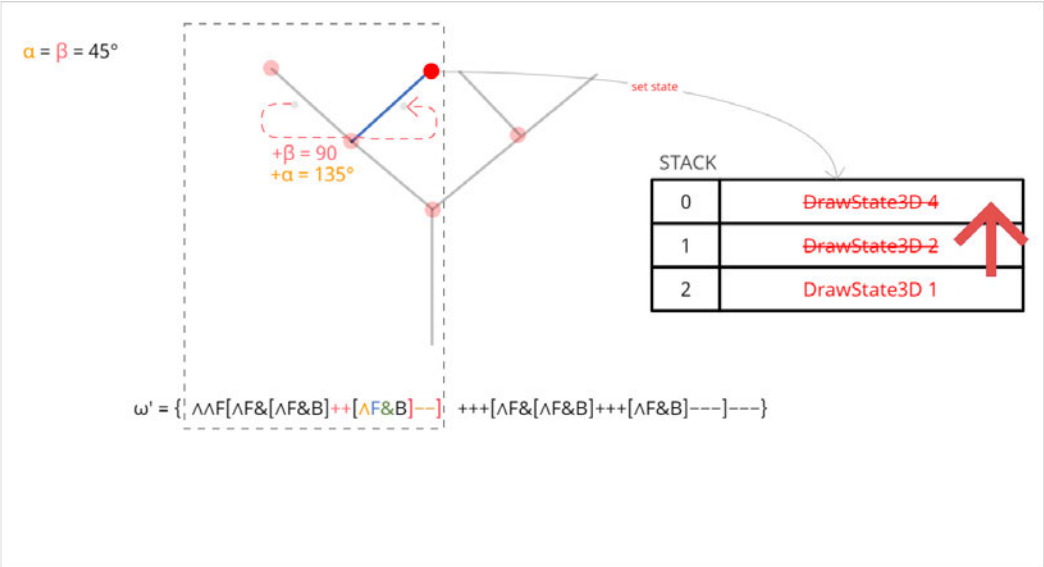


Abbildung 5.4.: Ableitung der Turtlegrafik. Schritt 4. (Quelle: Eigene Darstellung)

Abschließend wird der Stack abgebaut und β verringert. Die letzten vier Symbole im gestrichelten Rahmen bewirken diese Aktionen (Folge:] - -]). Zuvor wurde durch Symbol "F" eine Linie erzeugt. Die Position in "Drawstate3D 4" auf den Stack gelegt. Der Abbau erfolgt durch das Symbol "]". Da dies zweimal geschieht, rutscht der "DrawState3D 1" nach oben (symbolisiert durch den roten Pfeil in Abbildung 5.4).

Im Stack verbleibt nun noch der "DrawState3D 1". Die Symbole "- -" setzen den Wert von β auf 0°. Vom "DrawState3D 1", welcher als erstes auf den Stack gelegt wurde (siehe Schritt 1, Abb. 5.1), wird nun der rechte Teil des Baumes gebildet und die 3D-Turtlegrafik (3DTG) komplettiert. Dies geschieht nach dem zuvor beschriebenen Schema.

Die komplettierte Turtlegrafik abgeleitet von ω' ist in Abbildung 5.5 zu erkennen. Das ist die Basis für die Bildung eines Körpers für das ”Baumskelett”. Wie dies umgesetzt wird, behandelt das nächste Unterkapitel 5.3.

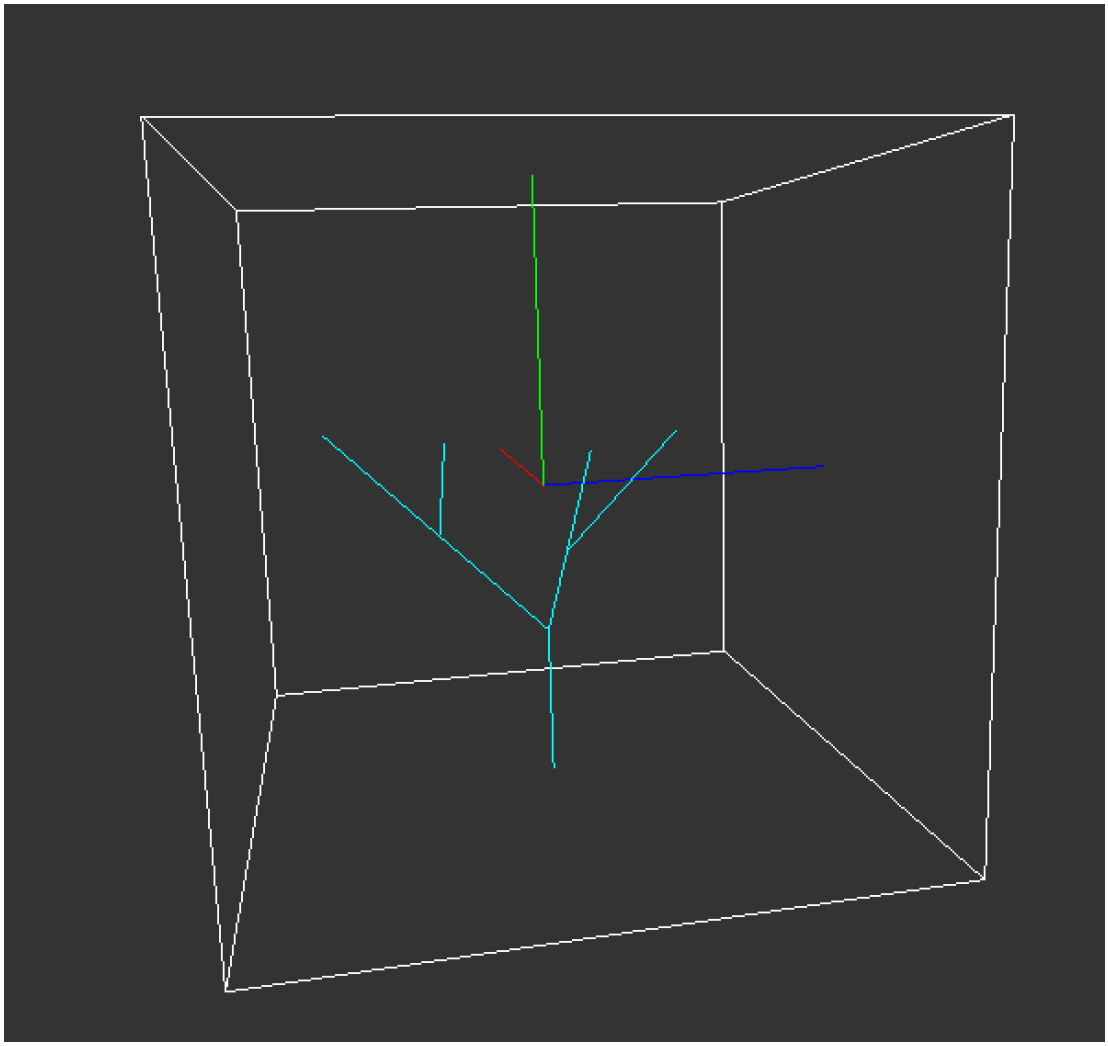


Abbildung 5.5.: Turtlegrafik resultierend aus ω' (Quelle: Eigene Darstellung)

5.3. Körper konstruieren

Wie in Kapitel 4 erläutert, soll der Körper des Baumes mit Distanzfunktionen beschrieben werden. Die Abbildung 5.6 visualisiert dazu die grundlegende Idee, wie aus der 3D-Turtlegrafik (3DTG) zunächst der Körper des Baumstamms konstruiert wird.

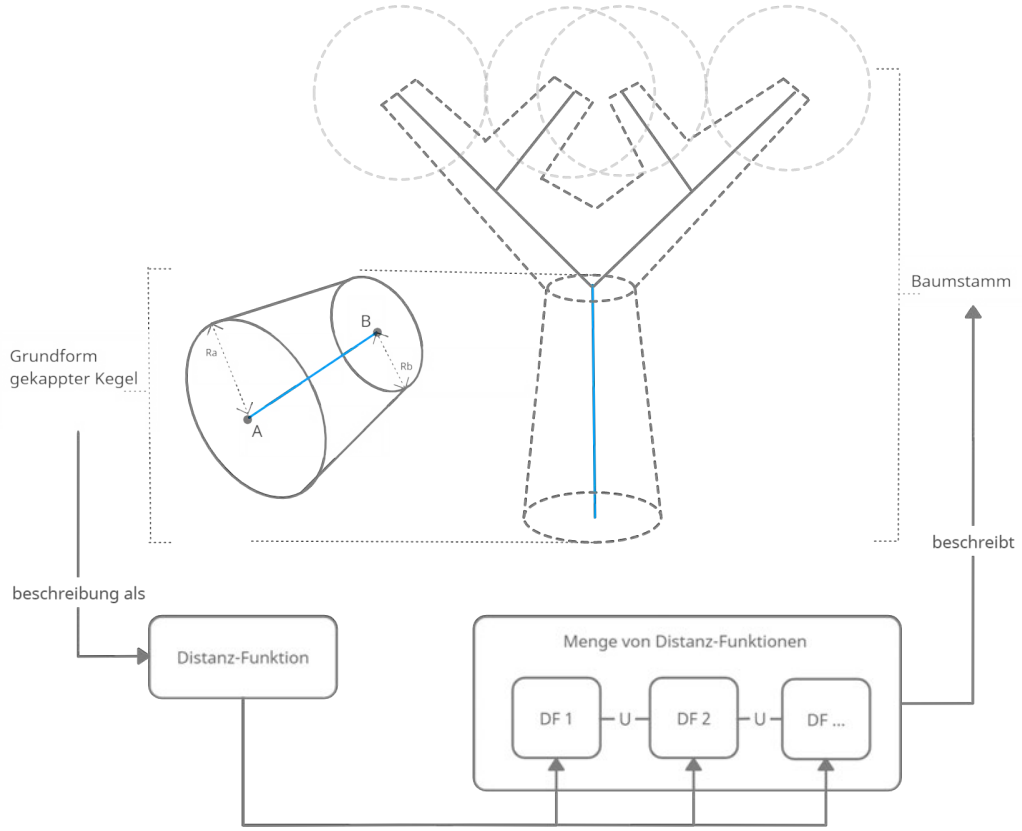


Abbildung 5.6.: Baumstamm bestehend aus vereinigten Kegeln (Quelle: Eigene Darstellung)

Betrachtet man die blaue Linie, ist diese ein Segment der 3DTG. Um dieses Segment wird ein Hüllkörper gelegt, sodass aus der Linie ein Volumen erzeugt werden kann. Der Hüllkörper ist, wie in Abbildung 5.6 gezeigt, ein gekappter Kegel. Der Kegel wird so positioniert, dass die blaue Linie die zentrale Achse des Kegels bildet. Die Mittelpunkte A und B der Kegelkreisflächen sind 3D-Vektoren. Zusammen mit den Radien R_a und R_b legen sie die Parameter der DF des Kegels fest, die ein Segment beschreibt. Die DF des Kegels wird durch die Javaklasse "ImplicitFunction3DCappedCone" umgesetzt (siehe Anhang Abb. A.3).

Nach diesem Schema werden alle Segmente der 3DTG mit einem Kegel umhüllt. Die Vereinigung aller Kegel zu einer gemeinsamen DF wird durch die Klasse "ImplicitFunction3DUnion" (siehe Anhang Abb. A.3) implementiert. Das Interface "ImplicitFuction3D" gibt die Funktion $f(v)|v \in R^3$ vor . Diese wird später vom Marching-Cubes-Algorithmus (MCA) aufgerufen, um zu prüfen, ob v im Körper liegt oder nicht.

Die konkreten Implementierungen ("ImplicitFunction3DCappedCone", "ImplicitFunction3DUnion") von $f(v)$ sind nach dem Vorbild von Inigo Inquilez umgesetzt, der verschiedene Implementierungen von geometrischen Körpern und Operationen auf seiner Website[1] zur Verfügung stellt.

Die Erzeugung der DF findet ebenfalls im 3D-Turtle-Interpreter (3DTI) statt. Genauer im Fall von Symbol "F" aus ω'. Hier werden die Vektoren A und B (siehe Abb. 5.6) aus dem Segment der 3DTG (blaue Linie) erzeugt. Die weiteren Parameter R_a und R_b berechnen sich wie folgt.

$$R_a = w_B \cdot r^{c_i} | r = \begin{cases} P_{r_1} & wenn \quad n = 1 \\ P_{r_2} & sonst \end{cases} \tag{5.3}$$

$$R_b = w_B \cdot r^{c_i+1} \tag{5.4}$$

Wobei w_B initial durch P_{Bw} gesetzt wird und sich w_B wie folgt berechnet:

$$w_B = \prod_{i}^{n} w_B \cdot r^{c_i} \tag{5.5}$$

Dabei ist n die aktuelle Stackgröße des 3DTI und c eine Zählervariable, welche die Anzahl der gezeichneten Linien (Baumäste) zur jeweiligen Stackgröße speichert. Durch die gezeigten Berechnungen in 5.3 bis 5.5 wird eine Verjüngung der Verästelungen im Baum bewirkt.

[1](2021). Distance functions, Adresse: https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm (besucht am 10. 12. 2021).

Nachdem die Umsetzungsdetails für die DF des Stamms erläutert wurden, zeigt der nächste Abschnitt, wie die Krone eines Baumes durch eine DF beschrieben wird. Als ein Überblick dient die nächste Abbildung 5.7.

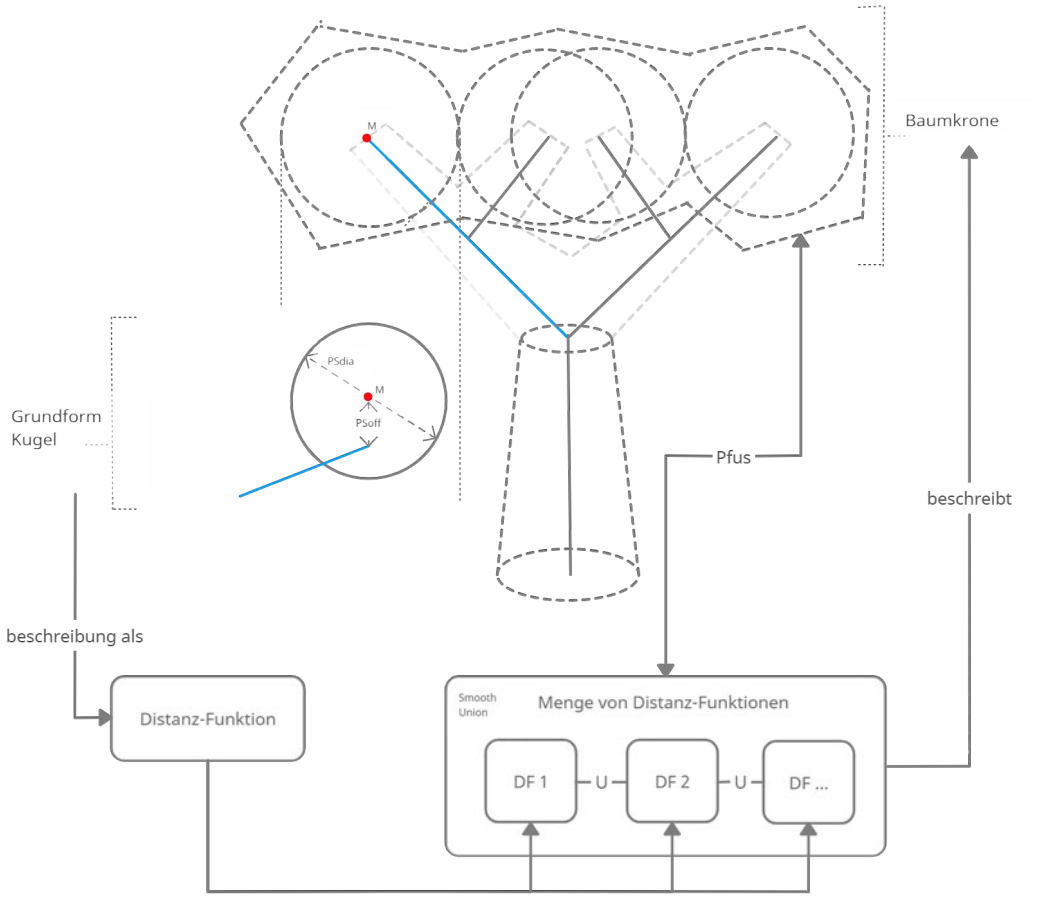


Abbildung 5.7.: Baumkrone bestehend aus Kugeln. (Quelle: Eigene Darstellung)

Grundsätzlich wird bei der Krone verfahren wie zuvor beim Stamm. Der geometrische Grundkörper wird durch eine Kugel ersetzt. Dieser wird nun mit seinem Mittelpunkt M (roter Punkt) an den Segmenten (blaue Linie) platziert. Der Parameter P_{Sdia} (siehe Tabelle 4.3) bestimmt dabei den Kugeldurchmesser, P_{Soff} (siehe Tabelle 4.3) den Offset zum Kugelmittelpunkt. Dieser kann genutzt werden, um die Krone in Y-Richtung zu verschieben und so die darunter liegenden Äste sichtbarer zu machen.

Ein weiterer Parameter ist P_{fus} (siehe Tabelle 4.3). Dieser nimmt Einfluss auf die DF und bestimmt dabei wie stark die Kugeln verschmelzen und einen gemeinsamen Körper bilden. Umgesetzt ist dies in der Klasse "ImplicitFunction3DSmoothUnion" (siehe Anhang Abb. A.2).

Die grundlegende Operation ist eine Verschmelzung (siehe Kapitel 2.4.2, "root smooth min" nach Inigo Quilez[2]), mit der die einzelnen DF der Kugeln zu einer komplexen DF verschmolzen werden. Je größer P_{fus} desto stärker gehen die Kugeln in einen gemeinsamen Körper über. So lässt sich weniger auf die Grundform der beteiligten DF zurückführen. Der Eindruck eines einzelnen komplexen Körpers entsteht.

5.4. Dreiecksnetz generieren

Diese Komponente ist auf Basis einer Implementierung des MCA aus dem PCG-Framework aufgebaut. Der Algorithmus ist in der Klasse "MarchingCubes" gekapselt (siehe Anhang Abb. A.3) und arbeitet wie in Kapitel 2.5 beschrieben. Der Algorithmus benutzt die zuvor gebildeten Distanzfunktion (DF), wie in Unterkapitel 2.4 erklärt, für Krone und Stamm des Baumes und wird jeweils einmal durchlaufen.

Das Ergebnis ist je ein Dreiecksnetz für Baumkrone und Baumstamm. Diese beiden Dreiecksnetze werden im Anschluss über Funktionen des PCG-Frameworks zusammengeführt. Das Ergebnis ist in den Abbildungen 5.8 und 5.9 zu sehen. Dort ebenfalls zu sehen ist ein Würfel (weiße Linien), welcher den Baum umhüllt. Dies ist der Bereich, in dem der MCA abläuft. Für die Generierung des Dreiecksnetzes wird eine Auflösung (P_{Tres}, P_{Cres}, siehe Tabelle 4.3) von 30 gewählt. Das bedeutet, dass der Würfel in $30^3 = 900$ kleinere Würfel vom MCA aufgeteilt wird. Je höher die Auflösung desto feiner ist das resultierende Dreiecksnetz, je kleiner desto gröber ist das Netz.

[2](2021). Smooth minimum, Adresse: https://www.iquilezles.org/www/articles/smin/smin.htm (besucht am 10. 12. 2021).

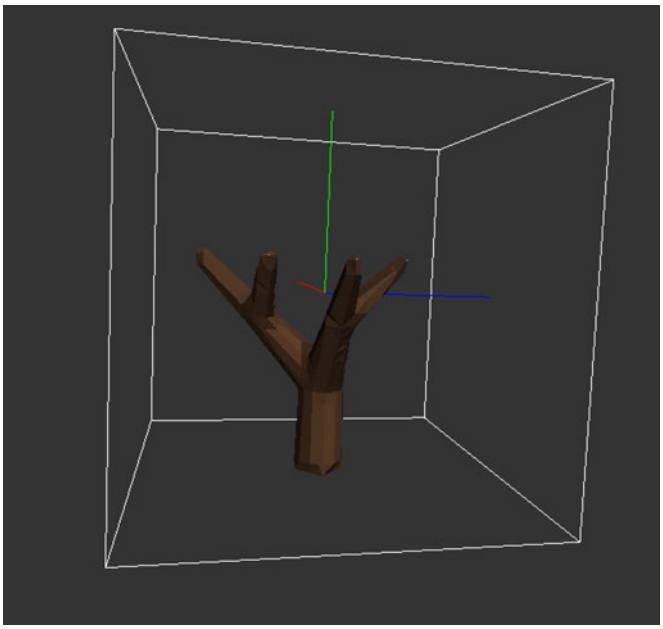


Abbildung 5.8.: Fertiger 3D-Baumstamm (Quelle: Eigene Darstellung)

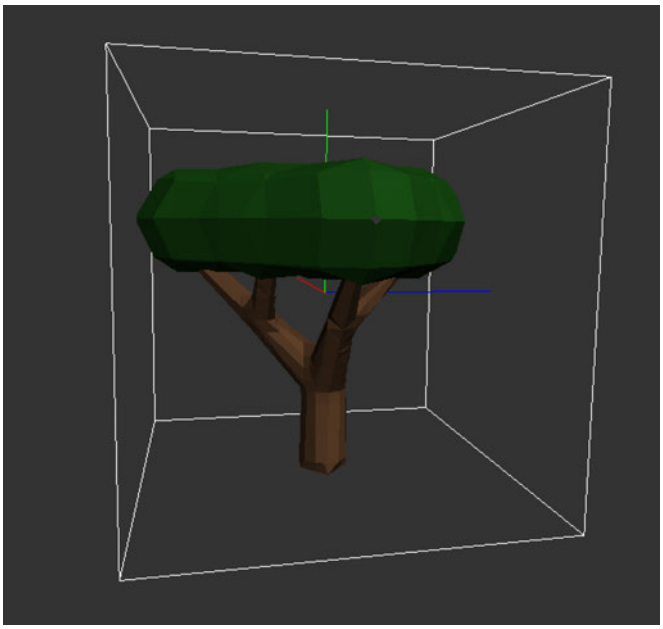


Abbildung 5.9.: Fertiger 3D-Baum (Quelle: Eigene Darstellung)

Wie sich während der Entwicklung schon abzeichnete, ist der MCA in Verbindung mit vielen Generationen P_{Tgen} (siehe Tabelle 4.3) und dem zufolge einer komplexen DF eine rechenintensive Aufgabe (siehe Kapitel 6.2). Deshalb wurde der MCA parallelisiert.

Das Marschieren durch die Würfel muss in keiner bestimmten Reihenfolge geschehen. Das bedeutet, dass die Berechnungen für jeden Würfel parallelisiert werden können. Genauer: Pro Prozessorkern könnten Berechnungen für einen Würfel durchgeführt werden. In der Praxis stehen allerdings nicht so viele Kerne zur Verfügung.

Die Idee ist deshalb, in der Implementierung die Anzahl an Prozessorkernen, die das System zur Verfügung stellt, abzufragen und alle zu besuchenden Würfel durch diese Anzahl zu teilen. So rechnet jeder Kern parallel an einer unterschiedlichen Liste an Würfeln. Konkret werden also im verbesserten Algorithmus so viele Threads erstellt, wie Prozessorkerne zur Verfügung stehen. Die Umsetzung ist mit der Klasse ”AcceleratedMarchingCubes” erfolgt (siehe Anhang A.3).

5.5. Erweiterung (Blätter)

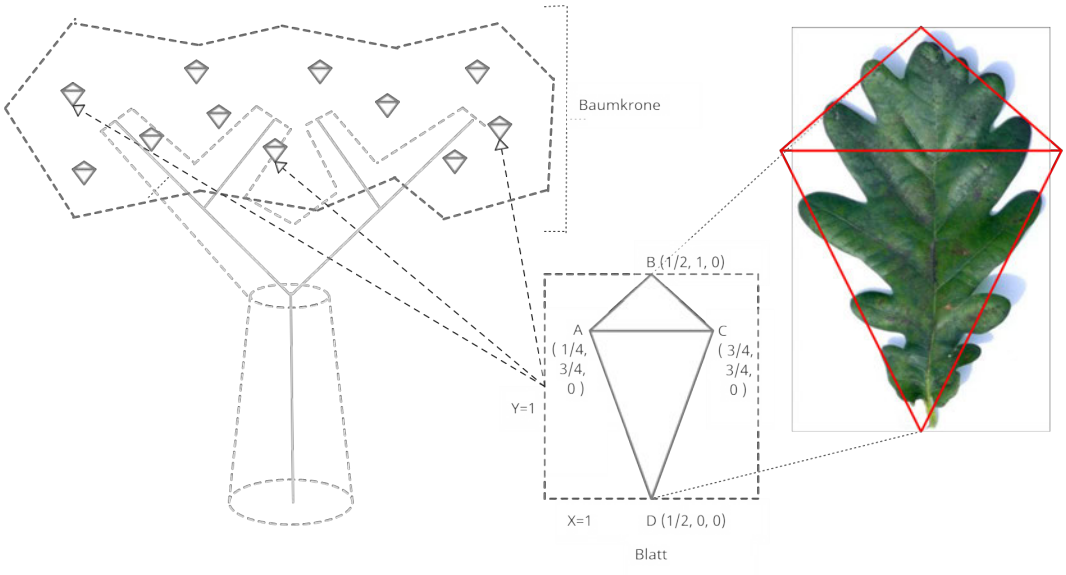


Abbildung 5.10.: Konzept für Blattdarstellung (Quelle: Eigene Darstellung)

Neben der Variante, die Baumkrone durch ein geschlossenes Dreiecksnetz abzubilden, wurde die Anwendung um eine zusätzliche Variante erweitert. Diese ermöglicht es, einzelne Blätter des Baumes darzustellen. Die Abbildung 5.10 zeigt dafür ein Konzept.

Dazu wird ein einzelnes Blatt wie in Abbildung 5.10 durch ein Dreiecksnetz dargestellt. Dieses besteht aus je zwei Dreiecken für Ober- und Unterseite. Durch Transformation, Rotation und Skalierung wird das einzelne Blatt im Raum positioniert. Konkret wurde dazu ein Interface ”Leaf” erstellt. Das in Abbildung 5.10 dargestellte Blatt ist eine simple und abstrakte Darstellung eines Eichenblattes und durch die Klasse ”OakLeaf” implementiert, welche wiederum das Interface ”Leaf” implementiert.

Um das einzelne Blatt plausibel am Baum zu positionieren, wird sich die Distanzfunktion (DF) der Baumkrone aus Kapitel 5.3 zunutze gemacht. Ähnlich wie beim MCA wird der Raum innerhalb des Würfels (weiße Linien, siehe Abbildung 5.9) diskretisiert. Einzelne Punkte im Raum werden nun durchschritten. An jedem Punkt wird mit der DF der Krone geprüft, ob der Punkt innerhalb der Krone liegt. Ist dies der Fall, wird an dieser Position eine Instanz von ”OakLeaf” erzeugt. Die Orientierung im Raum erfolgt per Zufall, damit die Blätter nicht in die gleiche Richtung zeigen. Ein so erzeugter Baum ist in Abbildung B.6 zu sehen.

6. Evaluation

Die in dieser Arbeit vorgestellte Software soll im folgenden Kapitel überprüft werden. Dazu wurden nach eigenem Ermessen sinnvolle Kriterien festgelegt, anhand derer die Überprüfung stattfinden soll.

Die Software wird in Form eines Praxistests geprüft. Dafür wird diese auf einem aktuellen Desktop-PC mit Windows 11 Betriebssystem ausgeführt und bewertet. Die verwendete CPU ist der AMD 5800X (8 physische und 16 logische Kerne).

6.1. Funktionalität und Zuverlässigkeit

In diesem Test soll die Anwendung als Gesamtes betrachtet werden. Es wird überprüft, ob sich alle Parameter im UI korrekt einstellen lassen und ob das damit erzeugte Ergebnis, das Mesh des Baumes, plausibel erscheint. Des Weiteren soll auf weitere Auffälligkeiten, wie z.B. Programmabstürze und Einfrieren der UI geprüft werden. In diesem Test sollen drei verschiedene Bäume (Tanne, Palme, Eiche) erzeugt werden. Die Mindestanfordungen zum Bestehen des Tests sollen die Folgenden sein.

- Baum 1 bis 3 erfolgreich erstellt
- Parameter können eingestellt und verändert werden und haben Einfluss auf das Ergebnis
- Keine Auffälligkeiten vorhanden

Ein Baum gilt als erfolgreich erstellt, wenn das Ergebnis, das erzeugte Mesh, nach Sichtprüfung plausibel erscheint.

Testergebnis

Alle Bäume konnten erfolgreich erstellt werden. Die Ergebnisse sind im Anhang in den Abbildungen B.4, B.5 und B.3 gezeigt. Die Parameter konnten erfolgreich geändert werden und haben auf das Aussehen des Dreiecksnetzes Einfluss genommen. Abstürze sind nicht aufgetreten.

Die Anwendung kann für große Auflösungen (P_{Tres} und P_{Cres} > 100) einfrieren, da der MCA durch mehr als $100^3 = 1.000.000$ Würfel iterieren muss. Ebenso kann die Funktionsauswertung der DF mit Generationen von P_{Tgen} > 10 für einen Baum lange dauern (> 50ms). Da diese Funktion in jedem Würfel achtmal aufgerufen wird, kann dies selbst bei niedrigen Auflösungen zeitintensiv sein.

Da Auffälligkeiten aufgetreten sind, hat die Anwendung den Test nicht bestanden.

6.2. Performance

In diesem Abschnitt soll die Performance der Anwendung, insbesondere die Berechungsdauer des MCA, überprüft werden. Genügt diese z.B. einen Baum in einer angemessenen Zeit zu berechnen? Ist die Dauer der ”Baumsynthese” abhängig von den Parametern?

Während der Entwicklung fiel auf, dass der MCA teilweise sehr lange (> 10s) rechnete, um ein Dreiecksnetz zu erzeugen. Deshalb wurde eine Optimierung bzgl. der Rechenzeit vorgenommen, wie in Kapitel 5.4 beschrieben. Um festzustellen, wie viel Performancegewinn diese Optimierung gebracht hat, sollen der optimierte (MCA_{opt}) und der normale MCA im Folgenden miteinander verglichen werden.

6.2.1. Vergleich der MCA-Varianten

Beide Varianten werden den gleichen Baum erzeugen, um eine Vergleichbarkeit herzustellen. Das bedeutet, dass alle Parametereinstellungen gleich sind sowie dieselbe Grammatik verwendet wurde. Die Einstellungen sind in der Testkonfiguration im Anhang B.1 festgehalten. Die erzeugten Bäume sind im Anhang unter Abbildungen B.1 und B.2 zu sehen.

Um zusätzlich den Rechenaufwand stufenweise zu erhöhen, sollen zwei Parameter geändert werden: P_{Cres} oder P_{Tres}. Diese Parameter wurden gewählt, weil sie die Schleifendurchläufe im MCA (MCA und MCA_{opt}) stark erhöhen. Eine Verdoppelung der Auflösung (P_{Cres} oder P_{Tres}) bedeutet eine Verachtfachung der Schleifendurchläufe im Algorithmus. Der MCA hat eine Komplexität von $O(n)$ [17]. Die folgende Tabelle 6.1 zeigt die Laufzeiten der Algorithmen.

Auflösung	MCA **(Stamm)**	MCA **(Krone)**	MCA_{opt} **(Stamm)**	MCA_{opt} **(Krone)**	**Anz. △ (Stamm)**	**Anz. △ (Krone)**
25	0,3s	0,2s	0,05s	0,02s	942	3.848
50	1,4s	1,4s	0,4s	0,2s	6.452	15.992
75	8,7s	4,6s	1,1s	0,43s	14.196	36.268
100	20,8s	10,9s	2,5s	1,0s	27.100	64.476
125	39,9s	21,3s	4,4s	1,9s	40.738	101.668
150	68,5s	36,9s	7,8s	3,3s	60.472	146.604
200	-	-	17,9s	7,8s	111.026	259.556

Tabelle 6.1.: Laufzeiten der MCA-Varianten

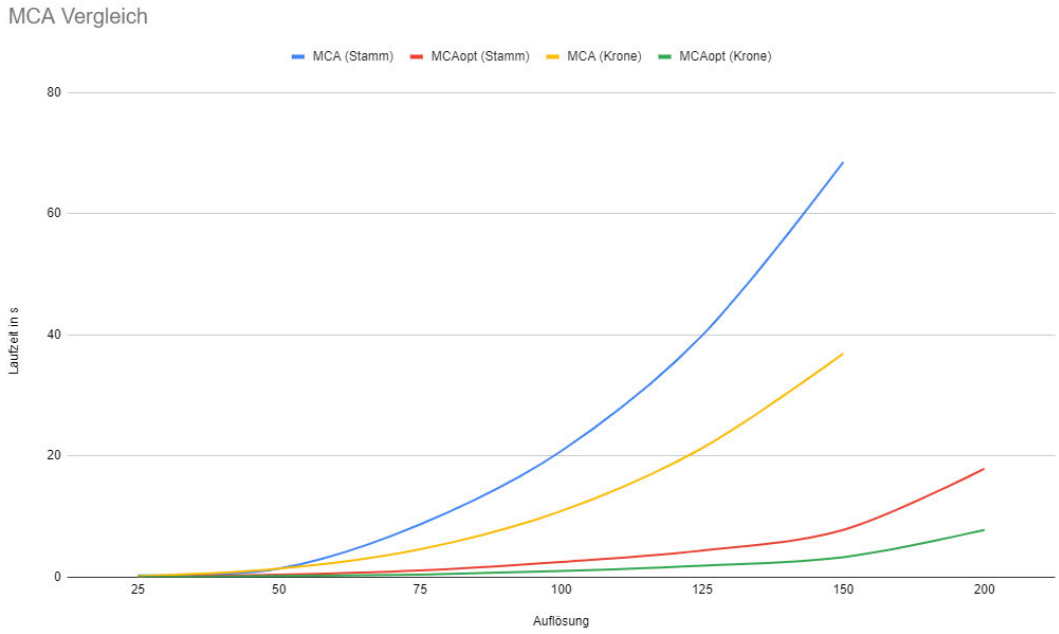


Abbildung 6.1.: Laufzeiten der MCA-Varianten (Quelle: Eigene Darstellung)

6.2.2. Testergebnis

Aus Abbildung 6.1 ist eine deutliche Reduzierung der Laufzeiten des MCA_{opt} feststellbar. Dennoch sind die Laufzeiten von Auflösungen > 125 in dieser Konfiguration in Form von Einfrieren der UI noch deutlich spürbar. Für Auflösungen < 100 ist die Laufzeit in einem akzeptablen Maß. Bäume sind so nahezu ohne spürbare Wartezeit generierbar. Für das Ziel, einen Baum in einem Low-poly-Stil zu generieren, genügen Auflösungen < 100. Somit ist die erreichte Laufzeit befriedigend.

6.3. Form und Varietät

Im Folgenden sollen die in 6.1 generierten Bäume visuell geprüft werden. Dadurch soll ermittelt werden, ob die generierten Bäume in Form, Aussehen und Varietät den Ansprüchen, festgelegt durch die folgenden Merkmale, genügen.

- Low-poly-Stil (Wirken die Dreiecksnetze der Bäume grob und kantig? Wie hoch ist die Anzahl der Dreiecke im Dreiecksnetz?)
- Varietät (Sind die generierten Bäume unterscheidbar? Ist die Anwendung fähig, unterschiedliche Baumarten zu erzeugen?)
- Qualität (Ist das Dreiecksnetz geschlossen oder weist es Fehler auf, wie z.B. Löcher oder unplausibel wirkende Polygone?)

Testergebnis

Festzustellen ist, dass die in Kapitel 6.1 erzeugten Bäume (B.4, B.5 und B.3) gut zu unterscheiden sind. Durch unterschiedliche Grammatiken und Parametereinstellungen ist es möglich, diverse Bäume zu generieren. Wegen der simplen Grundkörper (Kegel und Kugel) in Verbindung mit einer niedrigen Auflösung des MCA (P_{Cres} und P_{Tres} <= 75) ist ein polygonaler Look darstellbar. Feinere Auflösungen sind möglich, erhöhen aber deutlich den Rechenaufwand, wie bereits im vorherigen Kapitel 6.2 erläutert.

Des Weiteren sind Limitierungen und Fehler zu bemerken. Die Generierung der Dreiecksnetze ist ausschließlich in dem vom Würfel (weiße Linien) umschlossenen Bereich generierbar (siehe Anhang Abb. B.8). Auch kommt es vor, dass das Dreiecksnetz in einigen Fällen Löcher aufweist (siehe Abb. B.9).

7. Zusammenfassung

Ziel dieser Arbeit war es, eine Möglichkeit zu finden, 3D-Modelle von Bäumen im Low-poly-Stil zu generieren. Dafür wurde eine Anwendung auf Basis des PCG-Framework und JMonkey in Java entwickelt.

Die Software ermöglicht es dem Anwender, verschiedene Bäume zu generieren, zu exportieren und diese in anderen Projekten zu verwenden.

Dazu wurde eine Verarbeitungskette vorgestellt, bezeichnet als ”Baumsynthese” (siehe Abb. 1.2), die auf Basis von L-Systemen, eines Turtleinterpreters, von Distanzfeldern und dem Marching-Cubes-Algorithmus diese Aufgabe erfüllt. Durch eine geringe Auflösung des MCA wird ein Low-poly-Stil erzielt. Das L-System erzeugt dabei ein Wort (ω'), dessen Symbole durch den 3DTI in bilderzeugende Befehle umgesetzt werden. Das Resultat ist eine 3D-Turtlegrafik (3DTG), bestehend aus Linien im 3D-Raum. Vereinfacht ausgedrückt, stellt diese Grafik das ”Skelett” des Baumes dar. Um dieses ”Skelett” wird anschließend eine Hülle konstruiert und durch eine vereinte Distanzfunktion (DF) beschrieben. Der MCA kann mithilfe dieser Funktion ein Dreiecksnetz generieren, das fertige 3D-Baummodell, welches die Grafikkarte mit der Renderingpipeline (siehe Abb. 2.1) in einem Bild ausgibt.

In der Evaluation wurde die Software auf Funktionalität, Zuverlässigkeit und Performanz geprüft. Zudem wurden unterschiedliche Bäume generiert und die resultierenden Dreiecksnetze anhand festgelegter Kriterien überprüft, ob sie den visuellen Ansprüchen genügen. Die Evaluation ergab, dass die Software mit Einschränkungen funktionsfähig ist und verschiedene Bäume erzeugen kann. Es treten Einfrierungen des User-Interface (UI) für hohe MCA-Auflösungen auf. Des Weiteren können sporadisch Löcher im Dreiecksnetz eines Baummodells auftreten. Da der Anwender über das UI ein L-System vorgeben, verändern und diverse Parameter einstellen kann, ist es möglich, unterschiedliche Bäume zu erzeugen. Die Herausforderung für den Anwender ist hierbei, das nicht triviale Schreiben der Produktionsregeln P für das L-System.

Die Software erzeugt für Auflösungen (P_{Tres} und P_{PCres}) < 100 einen Low-poly-Stil. Noch geeigneter sind Auflösungen < 50. Feinere Auflösungen sind ebenfalls möglich, erhöhen aber den Rechenaufwand für das Erstellen des Baummodells. Unterhalb einer Auflösung von 75 und Generationen (P_{Tgen}) kleiner als 8 ist es möglich, nahezu verzögerungsfrei Modelle zu erzeugen. Die Baummodelle müssen in dem als Würfel (siehe Abb. B.8) gekennzeichneten Bereich erzeugt werden, da nur dort der MCA aus Performanzgründen angewendet wird.

Nach Erweiterung des Baumkronen-Algorithmus ist es möglich, einzelne Blätter statt eines geschlossenen Kronenkörpers darzustellen. Dazu wird die zuvor erstellte DF der Krone genutzt, die den Körper der Baumkrone definiert. Innerhalb dieser Krone werden einzelne Blätter platziert. Die Orientierung der Blätter wird per Zufall bestimmt, die Blattdichte anhand einer Auflösung wie beim MCA. Die so erzeugten Bäume sind visuell überzeugender (siehe Abb. B.6). Der Low-poly-Stil wird gewahrt, da die Blätter selbst in diesem Stil gehalten sind. In Abhängigkeit vom Parameter P_{Tgen} sind auch z.B. verschiedene Entwicklungsstadien eines Baums generierbar (siehe Abb. B.7).

Betrachtet man die Tabelle 6.1, so sieht man, dass die Dreiecksnetze selbst für niedrige Auflösungen < 50 eine hohe Anzahl von Dreiecken aufweisen. Subjektiv betrachtet, erzeugt der Low-Poly-Tree-Generator (LPTG) optisch ansprechende Bäume im Low-poly-Stil, das darunter liegende Dreiecksnetz kommt dem jedoch nicht ganz nach. Deswegen gibt es noch Bedarf, dies zu verbessern. Im nächsten Kapitel werden Möglichkeiten aufgezeigt, die unter anderem dieses Problem lösen können.

8. Ausblick

Aufgrund der vielschichtigen Arbeiten zu diesem Thema, kann man sich von zahlreichen Arbeiten für künftige Erweiterungen inspirieren lassen. Der zeitliche Rahmen der vorliegenden Arbeit ist begrenzt, sodass sich nicht alle Ideen umsetzen ließen. Es bleiben aber noch offene Probleme, welche zuvor angedeutet wurden und deren Lösung lohnenswert wäre. Im Folgenden soll ein Ausblick gegeben werden, was möglich ist.

Die Arbeit von Shekhar, Fayyad, Yagel u. a. [11] befasst sich mit dem Thema, wie man den MCA optimieren kann und erzielte damit gute Ergebnisse. Sowohl Ausführungszeit und Anzahl der Dreiecke im Dreiecksnetz konnten stark reduziert werden.

Eine weitere Möglichkeit, die Anzahl der Dreiecke im Netz zu reduzieren, ist der Ansatz von Garland und Heckbert [12], die es mit dem in dieser Arbeit vorgestellten Algorithmus ermöglichen Dreiecksnetze zu minimieren.

Des Weiteren sind Verbesserungen am Algorithmus der Baumkrone empfehlenswert. So können z.B. mit relativ geringem Aufwand neue Blattformen erstellt werden. Dafür ist es auf Implementierungsseite nötig, das Leaf-Interface in eine neue Blatt-Klasse zu implementieren. Eine naheliegende Erweiterung diesbezüglich wäre eine Nadel-Klasse, um die Darstellung von Nadelbäumen zu verbessern.

Ebenfalls denkbar wären Früchte oder Accessoires, z.B. Tannenzapfen, Kokosnüsse oder Weihnachtsbaumschmuck. Die Arbeit von De Reffye, Edelin, Françon u. a. [7] könnte hier hilfreich sein, da diese unter anderem ein Konzept zur Erstellung von Früchten und Knospen zeigt.

Denkbar wäre ebenso, sich vom Ansatz mit dem Marching-Cubes-Algorithmus (MCA) zur Generierung einer Oberfläche zu trennen und ein ähnliches Konzept wie in der Arbeit von Bloomenthal [5] zu nutzen. Auf eine implizite Beschreibung des Baumkörpers wird hier ver-

zichtet. Das Dreiecksnetz wird explizit erzeugt.

Es wird ebenfalls ein ”Baumskelett” benutzt. Anders als in dieser Arbeit wird das ”Skelett” nicht durch Linien sondern mithilfe von Splines repräsentiert, was eine denkbare Erweiterung für den 3D-Turtle-Interpreter (3DTI) sein könnte. Dies hat den Vorteil, dass die Äste optisch geschwungener und natürlicher wirken.

Arbeiten an der Nutzbarkeit und Bedienung der Software sind empfehlenswert, da das Schreiben von Grammatiken zur Beschreibung eines Baumes nicht sehr eingängig ist, und es viel Einarbeitungszeit benötigt, bis man gewünschte Ergebnisse erzielt. Ein naheliegender Ansatz wäre eine Auswahl an vorgegebener Grammatik anzubieten, welche in einem Baukastensystem kombiniert werden könnte. Zudem könnte man Parameter zusammenführen und diese auf nur einem oder wenigen Parametern abbilden. Dankbar wäre hier z.B. der Parameter Baumalter, welcher Einfluss auf z.B. Astdicke, Grammatik, Blattdichte usw. hätte.

Man könnte dies aber ebenso mit Beteiligung einer KI lösen. In der Arbeit von Goodfellow, Pouget-Abadie, Mirza u. a. [25] wurde ein Konzept vorgestellt, welches aus Generator und Diskriminator besteht. Der Generator könnte hier mit Justierungen von Parametern und Änderungen an der Grammatik Bäume erstellen. Der Diskriminator, der anhand realer Bilder von Bäumen trainiert worden sein könnte, würde diese Bäume bewerten und an den Generator das Feedback geben, ob der erstellte Baum plausibel erscheint oder nicht. So würde der Generator lernen, Parameter und Grammatik nur in bestimmten Kombinationen anzupassen. Das Ergebnis wären KI-generierte Bäume, nach dem Vorbild der Natur. Ein Mensch zur Steuerung des UI wäre so nicht mehr notwendig.

Zudem sollte das Einfrieren der UI behoben werden. Dies ließe sich durch eine Auslagerung der Methode zur Generierung des Baums in einen weiteren Thread ermöglichen. Eine Einbindung eins Indikators, der Hintergrundberechnungen anzeigt, schafft zusätzliche Transparenz für den Anwender.

A. Klassendiagramme

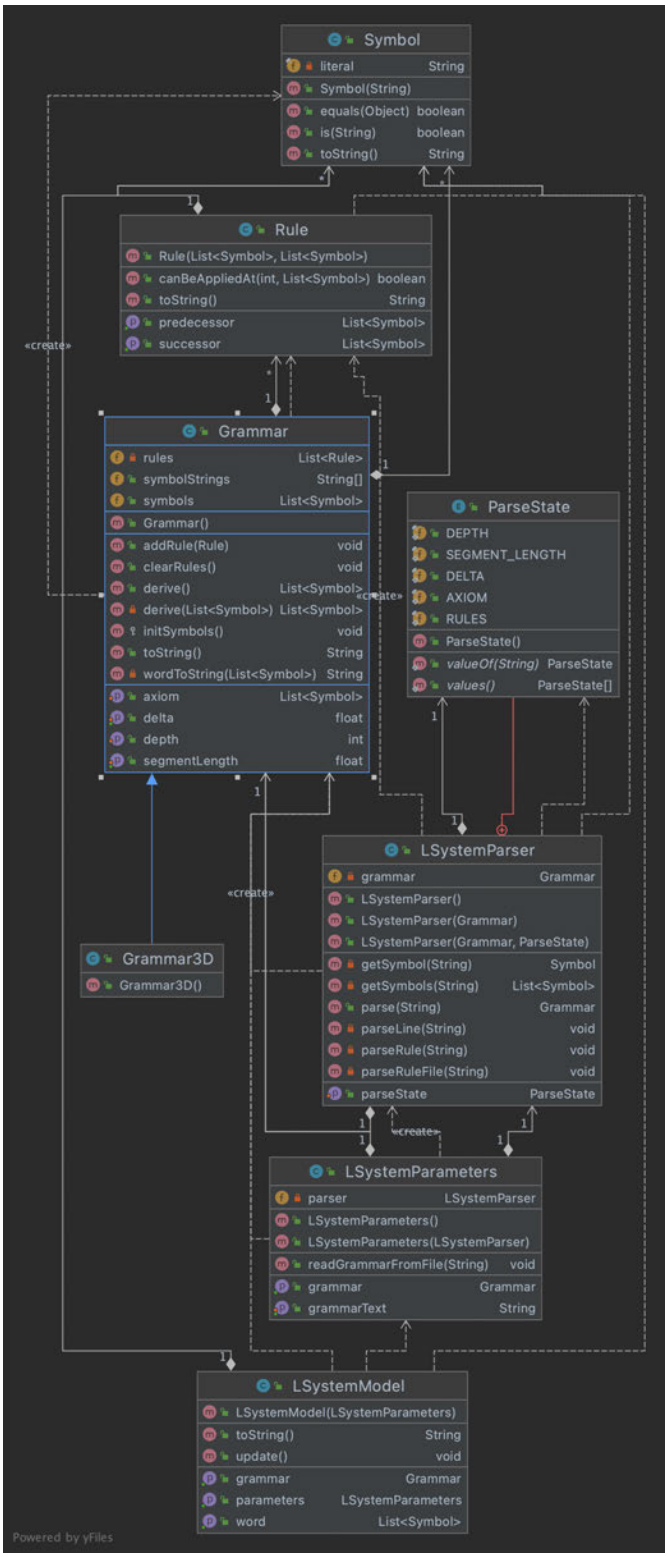


Abbildung A.1.: Klassendiagramm des L-Systems nach Erweiterung der Basisimplementierung

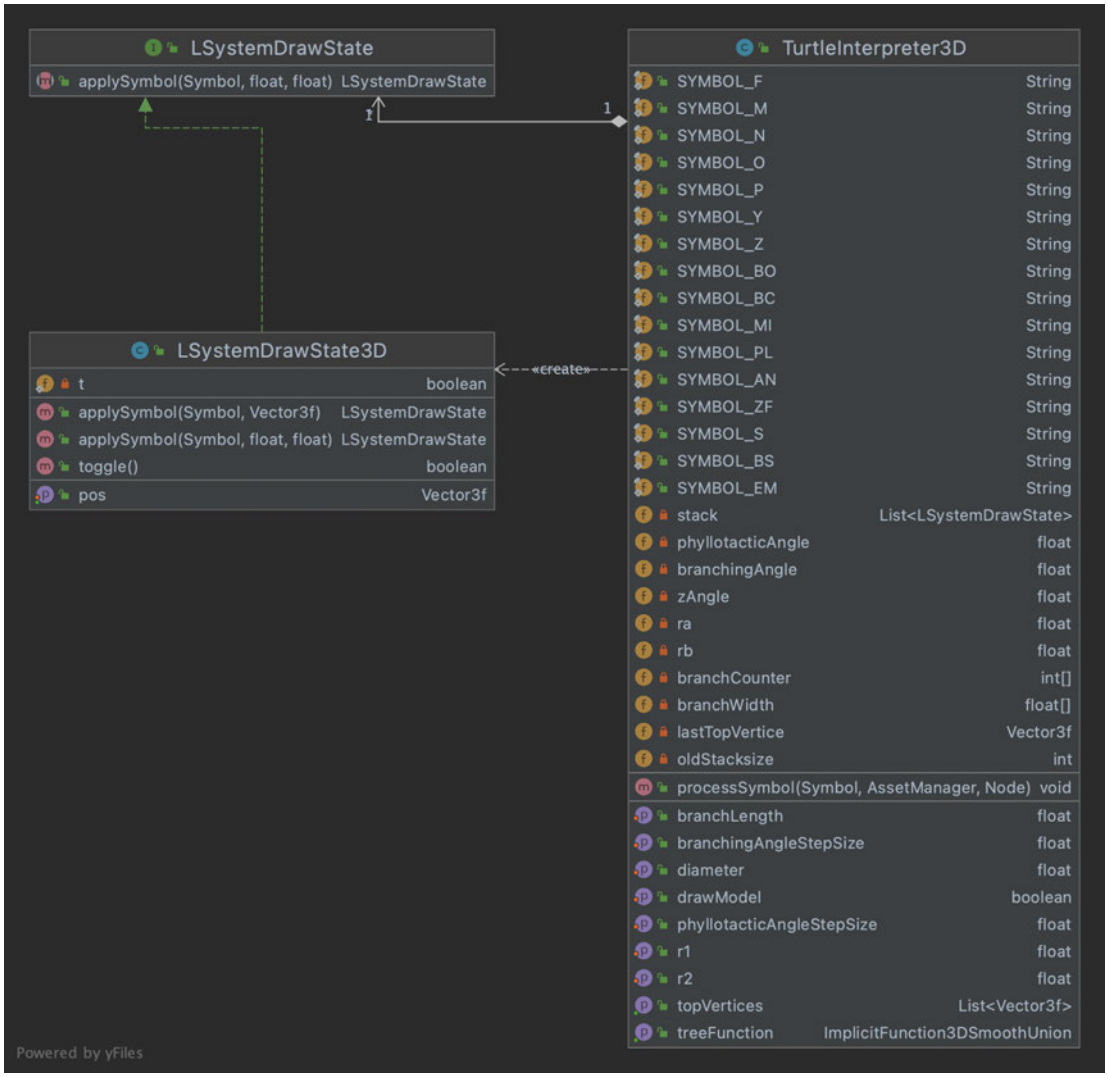


Abbildung A.2.: Klassendiagramm des 3D-Turtle-Interpreter (3DTI)

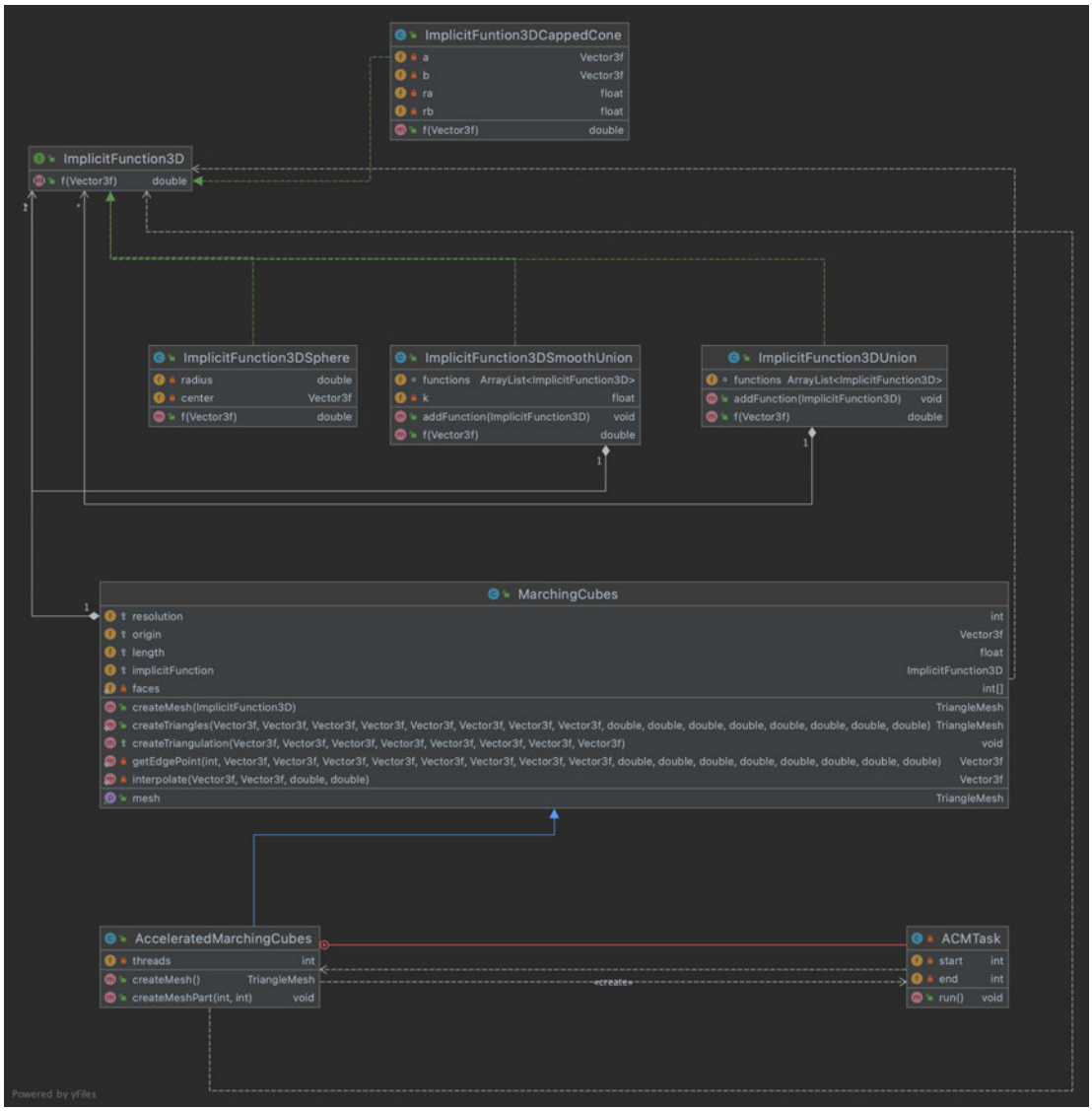


Abbildung A.3.: Klassendiagramm des Mesh-Generators

B. Evaluation

B.1. Parameter (Fichte)

P_{Cres} = 75 P_{Tres} = 75 P_{Tgen} = 6 P_{Blen} = 0.11 P_{Bdia} = 0.1 P_{r1} = 0.95

P_{r2} = 0.825 P_{Sdia} = 0.08 P_{Soff} = 0.06 P_{Fus} = 0.01 P_{α} = 22.5 P_{β} = 22.5

$$P_{rules} = \begin{cases} P & \rightarrow & \wedge\wedge\wedge\wedge FFFFM \\ M & \rightarrow & \text{FFF[]AAAAAM} \\ A & \rightarrow & +++[\&\&\&\&\&\&F[]F[] \wedge F\&[]O \wedge\wedge\wedge\wedge\wedge\wedge] \\ N & \rightarrow & \wedge\wedge \&\&\&[]N \\ O & \rightarrow & [--F[]N++]++[F[]N]--\&[] \wedge \&\&[]N \\ F & \rightarrow & F \\ + & \rightarrow & + \\ - & \rightarrow & - \\ [& \rightarrow & [\\ ] & \rightarrow &] \\ \wedge & \rightarrow & \wedge \\ \& & \rightarrow & \& \end{cases} \tag{B.1}$$

B.2. Grafiken

B.2.1. Fichte

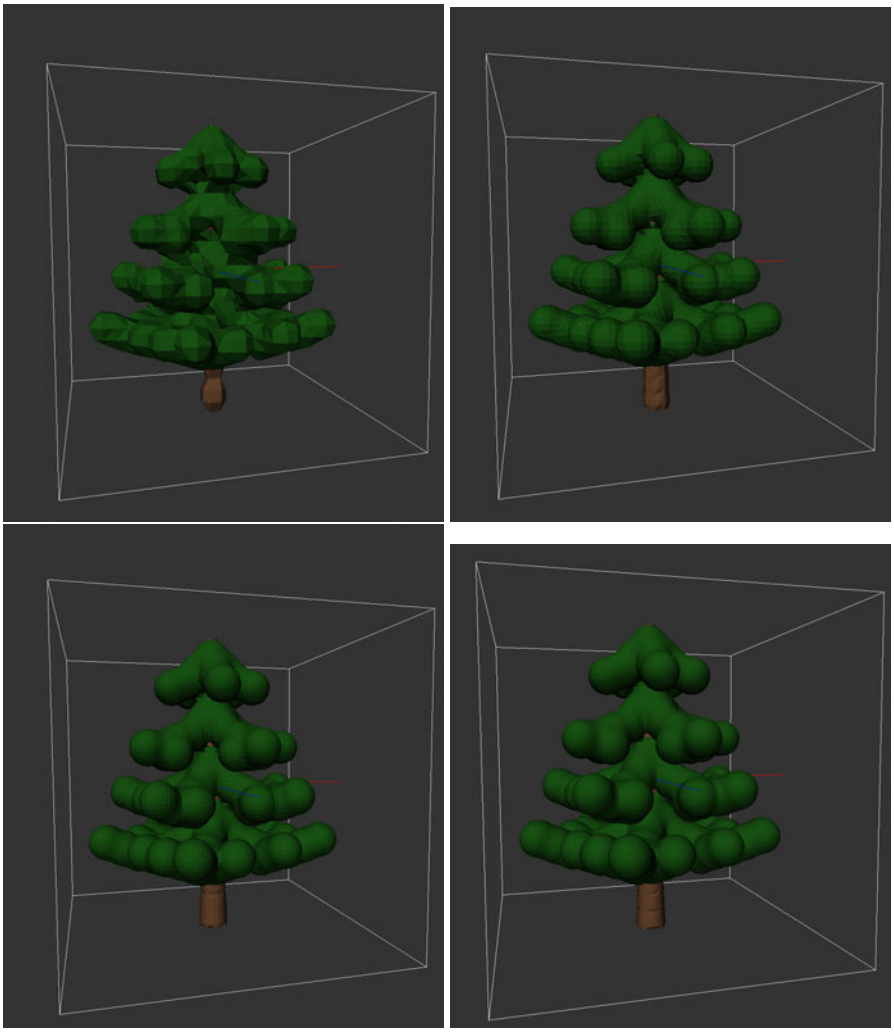


Abbildung B.1.: Fichten in diversen Auflösungen 25 (o.l.) bis 100 (u.r.)

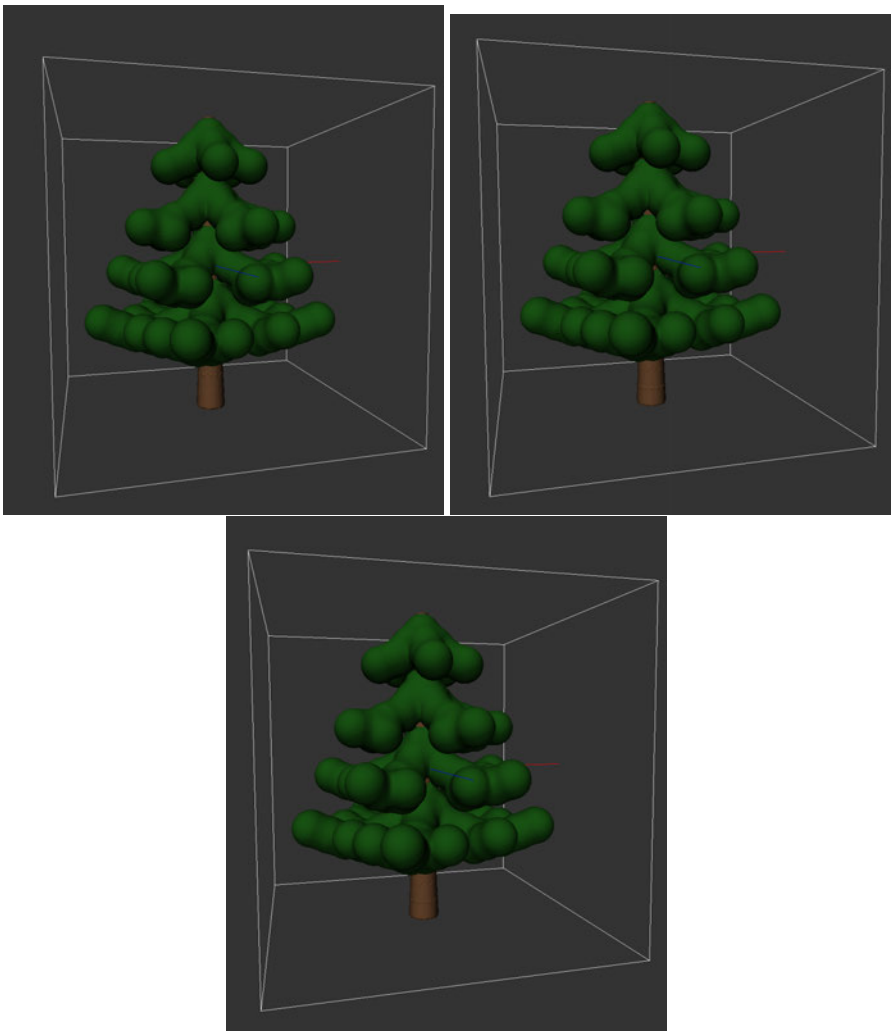


Abbildung B.2.: Fichten in diversen Auflösungen 125 (o.l.) bis 200 (u.m.)

B.2.2. Palme

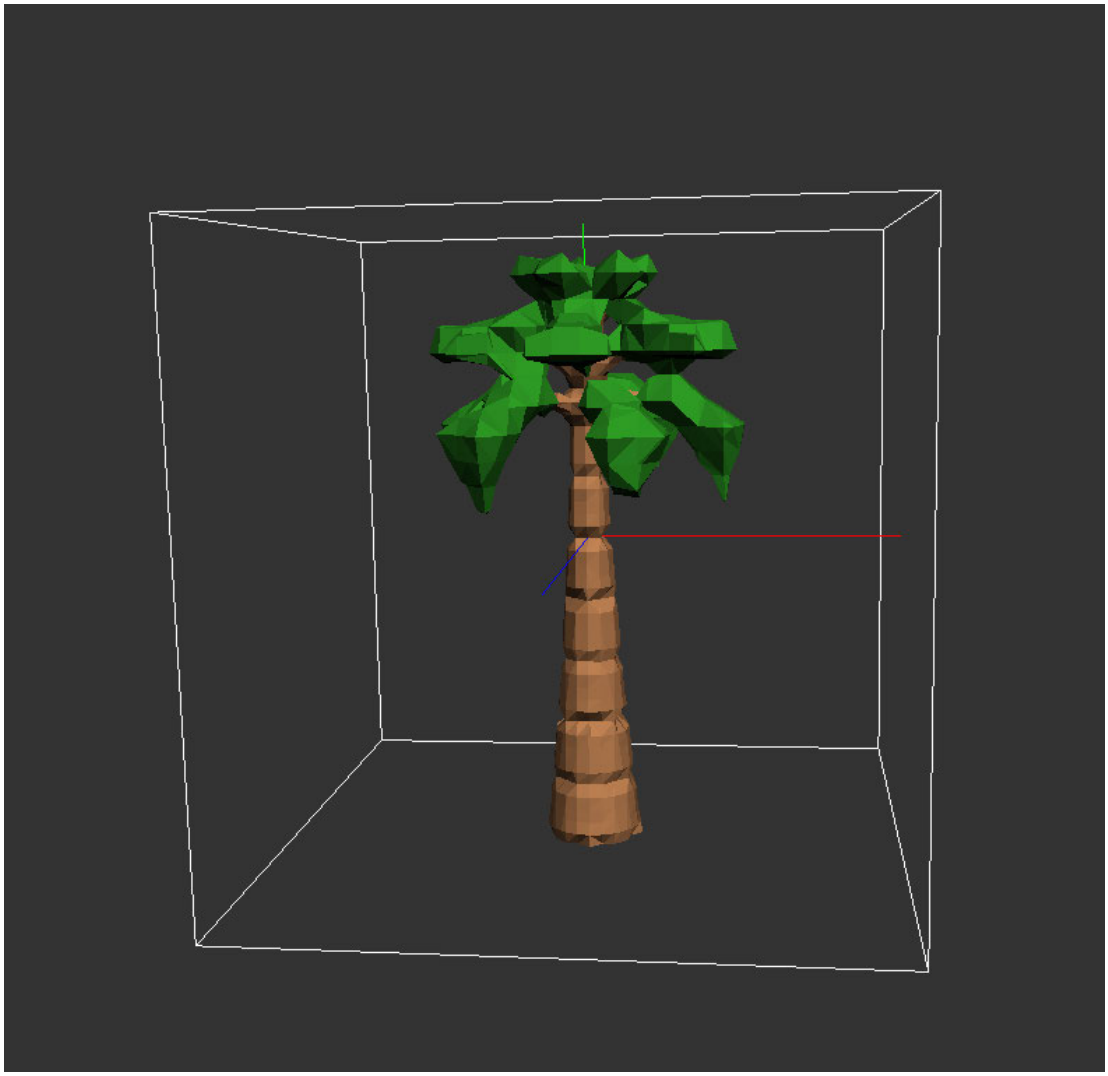


Abbildung B.3.: Palme mit einer Auflösung von 50 beim Stamm und 25 bei der Krone

B.2.3. Eiche

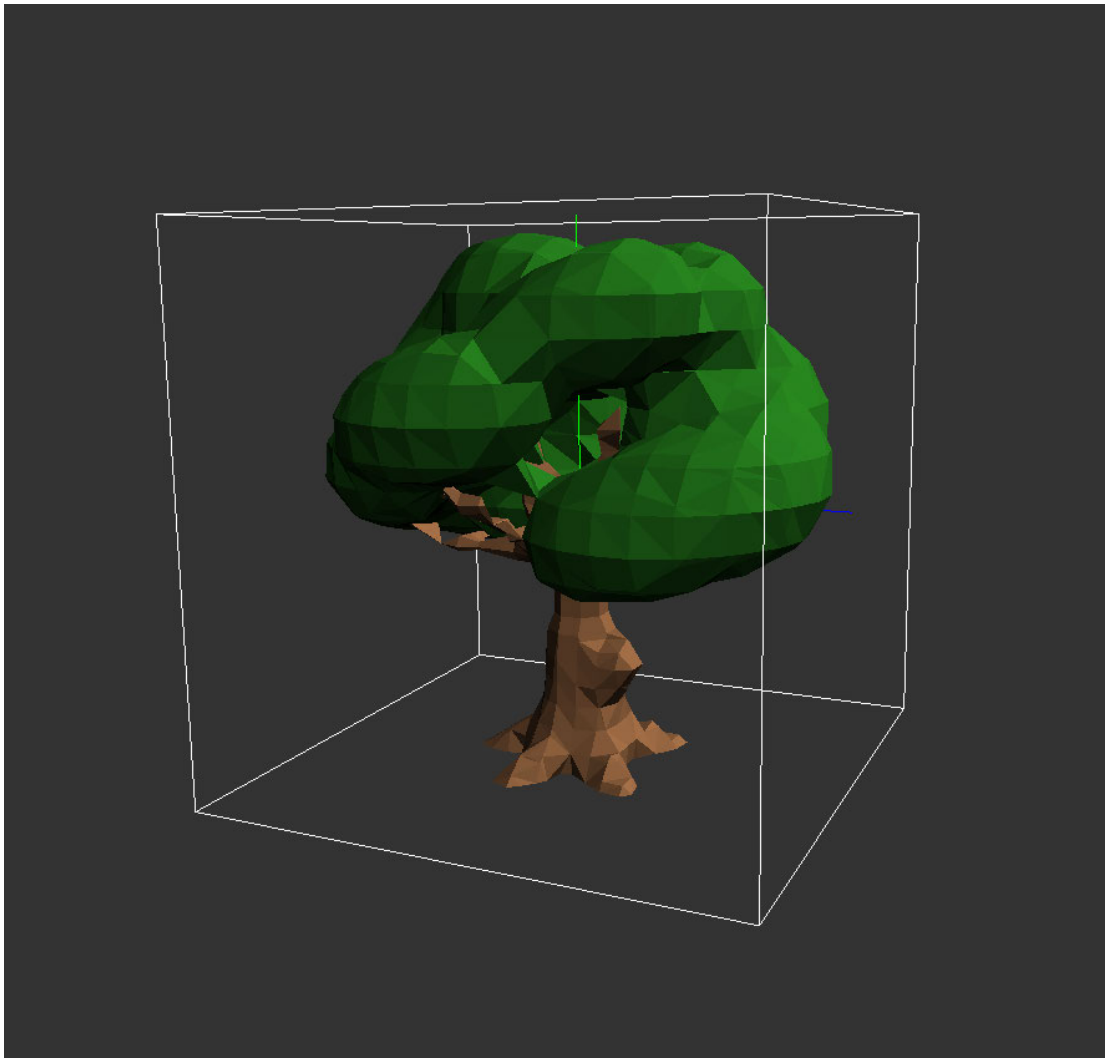


Abbildung B.4.: Eiche mit einer Auflösung von 25 beim Stamm und 15 bei der Krone

B.2.4. Tanne

Abbildung B.5.: Tanne mit einer Auflösung von 35 beim Stamm und 20 bei der Krone

B.2.5. Eiche nach Erweiterung

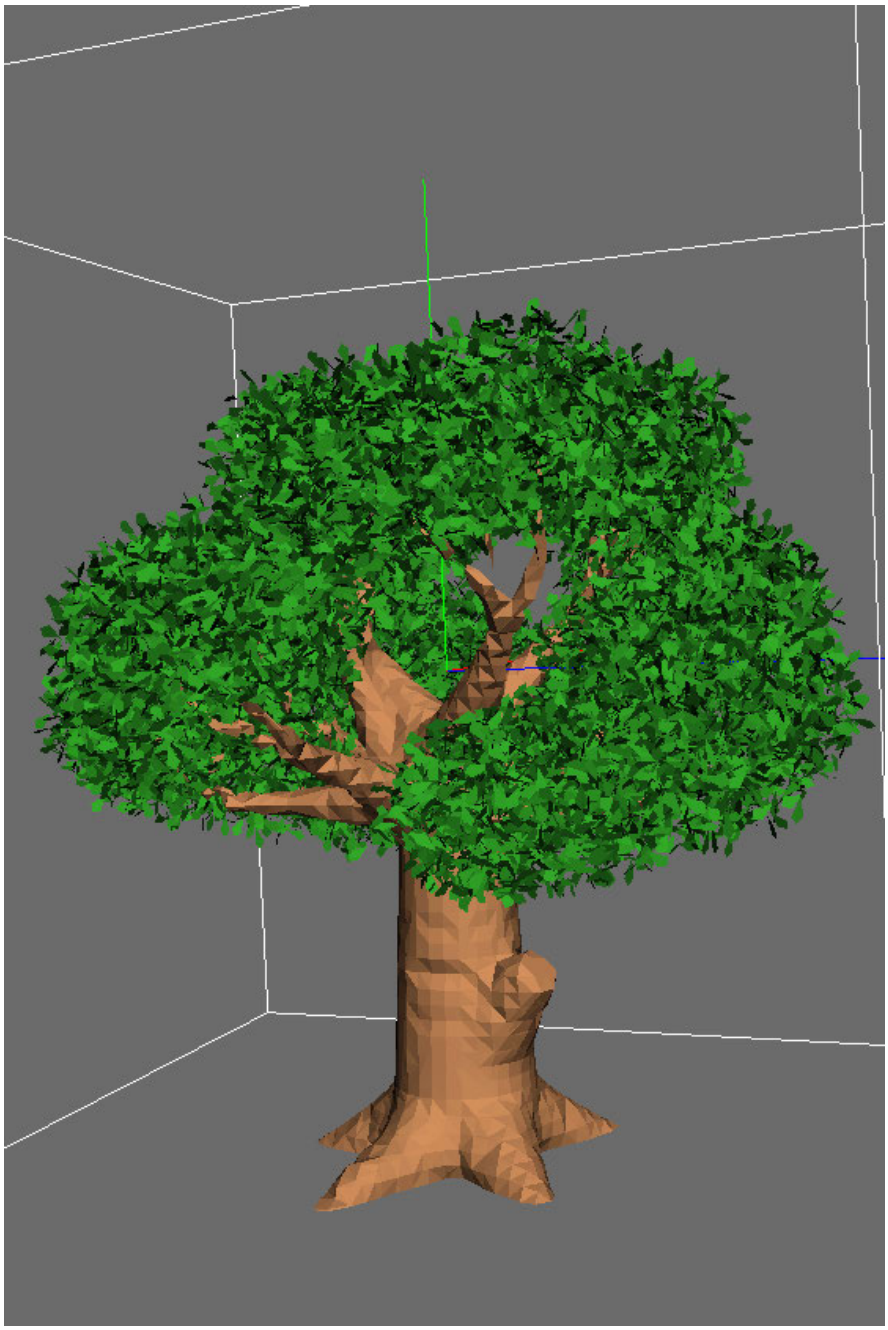


Abbildung B.6.: Eiche nach Erweiterung

B.2.6. Wachstum einer Eiche

Abbildung B.7.: Verschiedene Entwicklungsstadien einer Low-poly-Eiche

B.2.7. Meshfehler

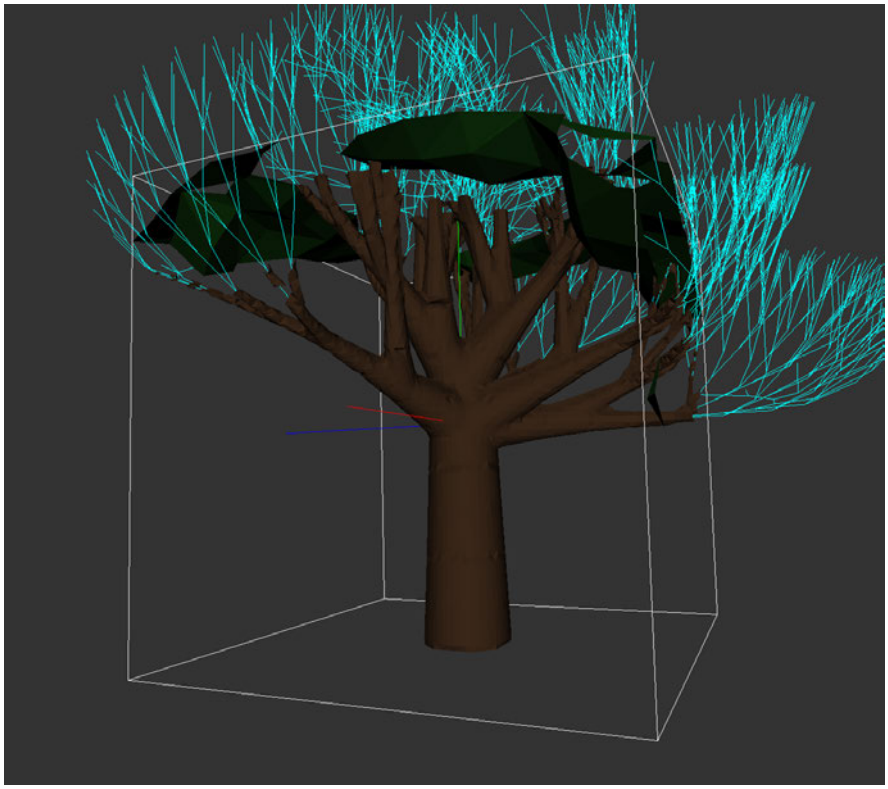


Abbildung B.8.: Generierung innerhalb des Würfels

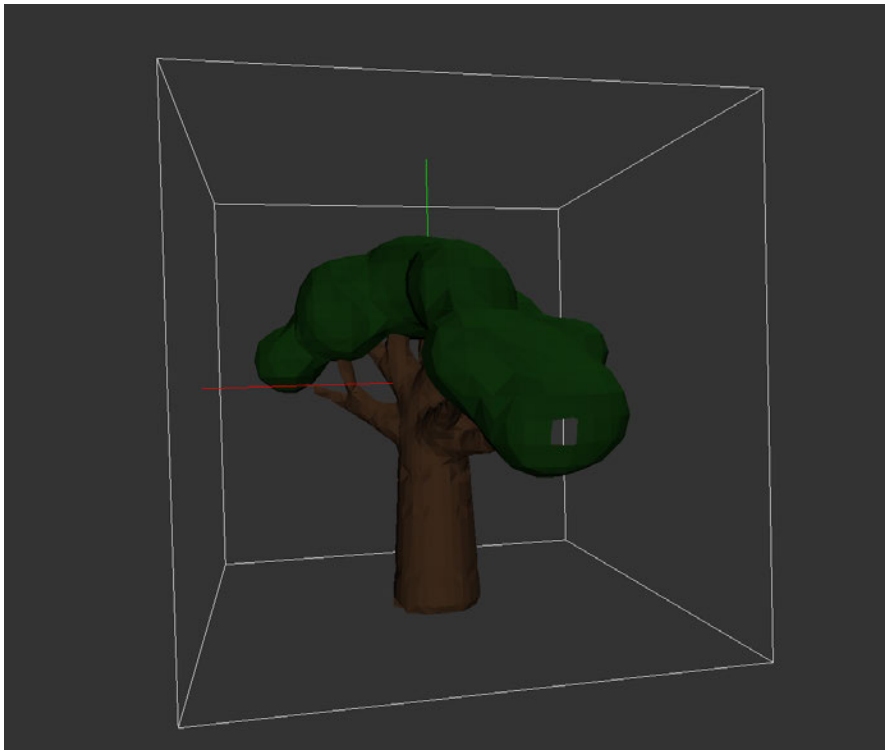


Abbildung B.9.: Beispiel für ein Fehler im Dreiecksnetz

Literaturverzeichnis

[1] N. Chomsky, *Three Models for the Description of Language* -. Institute of Radio Engineers, Professional Group on Information Theory, 1956.

[2] A. Lindenmayer, "Mathematical models for cellular interaction in development: Parts i and ii.", *Journal of Theoretical Biology*, Jg. 18, 1968.

[3] H. Honda, "Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body", *Journal of Theoretical Biology*, Jg. 31, Nr. 2, S. 331–338, 1971.

[4] C. De Boor und C. De Boor, *A practical guide to splines*. springer-verlag New York, 1978, Bd. 27.

[5] J. Bloomenthal, "Modeling the mighty maple", *ACM SIGGRAPH Computer Graphics*, Jg. 19, Nr. 3, S. 305–311, 1985.

[6] W. Lorensen und H. Cline, "Marching cubes: A high resolution 3d surface construction algorithm", *ACM SIGGRAPH Computer Graphics*, Jg. 21, S. 163–, Aug. 1987. DOI: 10.1145/37401.37422.

[7] P. De Reffye, C. Edelin, J. Françon, M. Jaeger und C. Puech, "Plant models faithful to botanical structure and development", *ACM Siggraph Computer Graphics*, Jg. 22, Nr. 4, S. 151–158, 1988.

[8] P. Prusinkiewicz, A. Lindenmayer, J. Hanan, F. Fracchia, M. Cutter, D. Fowler, M. de Boer und L. Mercer, *The Algorithmic Beauty of Plants*, Ser. The Virtual Laboratory. Springer New York, 1990, ISBN: 9783540972976.

[9] A. A. Pasko und V. V. Savchenko, "Blending operations for the functionally based constructive geometry", 1994.

[10] J. Weber und J. Penn, "Creation and rendering of realistic trees", in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, Ser. SIGGRAPH '95, New York, NY, USA: Association for Computing Machinery, 1995, 119–128, ISBN: 0897917014. DOI: 10.1145/218380.218427. Adresse: https://doi.org/10.1145/218380.218427.

[11] R. Shekhar, E. Fayyad, R. Yagel und J. Cornhill, "Octree-based decimation of marching cubes surfaces", in *Proceedings of Seventh Annual IEEE Visualization '96*, 1996, S. 335–342. DOI: 10.1109/VISUAL.1996.568127.

[12] M. Garland und P. S. Heckbert, "Surface simplification using quadric error metrics", in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, S. 209–216.

[13] m. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica, Second Edition* -. Boca Raton, Fla: CRC Press, 1997, ISBN: 978-0-849-37164-6.

[14] S. Campagna, L. Kobbelt und H.-P. Seidel, "A scalable representation for triangle meshes", 1998.

[15] B. Lintermann und O. Deussen, "Interactive modeling of plants", *IEEE Computer Graphics and Applications*, Jg. 19, Nr. 1, S. 56–65, 1999.

[16] M. Bender und M. Brill, "Computergrafik: Ein anwendungsorientiertes lehrbuch", in. Hanser, 2006, Kap. 6.1.1, S. 145, ISBN: 9783446404342. Adresse: https://books.google.de/books?id=VBogNNs10zEC.

[17] T. S. Newman und H. Yi, "A survey of the marching cubes algorithm", *Computers & Graphics*, Jg. 30, Nr. 5, S. 854–879, 2006.

[18] S. Osher und R. Fedkiw, *Level set methods and dynamic implicit surfaces*. Springer Science & Business Media, 2006, Bd. 153.

[19] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang und S. B. Kang, "Image-based plant modeling", in *ACM SIGGRAPH 2006 Papers*, 2006, S. 599–604.

[20] A. Runions, B. Lane und P. Prusinkiewicz, "Modeling trees with a space colonization algorithm.", *NPH*, Jg. 7, S. 63–70, 2007.

[21] B. Beneš, N. Andrysco und O. Šťava, "Interactive modeling of virtual ecosystems", in *Proceedings of the Fifth Eurographics conference on Natural Phenomena*, 2009, S. 9–16.

[23] S. Pirk, O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. Měch, B. Benes und O. Deussen, "Plastic trees: Interactive self-adapting botanical tree models", *ACM Transactions on Graphics (TOG)*, Jg. 31, Nr. 4, S. 1–10, 2012.

[24] E. H. Spanier, *Algebraic Topology* -. Berlin Heidelberg: Springer Science Business Media, 2012, ISBN: 978-1-468-49322-1.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio, "Generative adversarial nets", *Advances in neural information processing systems*, Jg. 27, 2014.

[26] J. Hughes, A. Van Dam, J. Foley, M. McGuire, D. Sklar, S. Feiner und K. Akeley, "Computer graphics: Principles and practice", in, Ser. The systems programming series. Addison-Wesley, 2014, S. 187, ISBN: 9780321399526. Adresse: https://books.google.de/books?id=OVpsAQAAQBAJ.

[27] W. Zhang, S. Xiao und X. Shi, "Low-poly style image and video processing", in *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2015, S. 97–100. DOI: 10.1109/IWSSIP.2015.7314186.

[28] B. Wassermann, S. Kollmannsberger, T. Bog und E. Rank, "From geometric design to numerical analysis: A direct approach using the finite cell method on constructive solid geometry", *Computers Mathematics with Applications*, Jg. 74, Nr. 7, S. 1703–1726, 2017, High-Order Finite Element and Isogeometric Methods 2016, ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2017.01.027. Adresse: https://www.sciencedirect.com/science/article/pii/S0898122117300780.

[29] L. Priese und K. Erk, "Theoretische informatik - eine umfassende einführung", in. Berlin Heidelberg New York: Springer-Verlag, 2018, Kap. 4, S. 53–61, ISBN: 978-3-662-57409-6.

[30] L. Yi, H. Li, J. Guo, O. Deussen und X. Zhang, "Tree growth modelling constrained by growth equations", in *Computer Graphics Forum*, Wiley Online Library, Bd. 37, 2018, S. 239–253.

[31] A. Nischwitz, M. Fischer, P. Haberäcker und G. Socher, "Computergrafik - band i des standardwerks computergrafik und bildverarbeitung", in. Berlin Heidelberg New York: Springer-Verlag, 2019, Kap. 16.2, S. 578–588, ISBN: 978-3-658-25384-4.

[32] ——, "Computergrafik - band i des standardwerks computergrafik und bildverarbeitung", in. Berlin Heidelberg New York: Springer-Verlag, 2019, Kap. 12, S. 317–368, ISBN: 978-3-658-25384-4.

[37] Y. Liu, J. Guo, B. Benes, O. Deussen, X. Zhang und H. Huang, "Treepartnet: Neural decomposition of point clouds for 3d tree reconstruction", *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, Jg. 40, Nr. 6, 232:1–232:16, 2021.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. Januar 2022 Florian Heuer