

Interaktive Pflanzenmodellierung

Abschlussarbeit
zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der
Hochschule für Technik und Wirtschaft Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang Angewandte Informatik

vorgelegt von

Donat Deva Brzoska
Matrikelnummer: 553655

1. Prüfer: Prof. Dr.-Ing. Thomas Jung
2. Prüfer: Julien Letellier, M.Sc.

Berlin, 05.09.2019



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Vorwort

Die grenzenlose Schönheit der Natur fasziniert mich schon mein Leben lang. Während meines Bachelorstudiums an der Hochschule für Technik und Wirtschaft in Berlin lernte ich die algorithmischen Möglichkeiten kennen, um Objekte im dreidimensionalen Raum zu modellieren. So widme ich nun meine Bachelorarbeit dem Thema der algorithmischen Pflanzenerzeugung.

Bevor die Schilderung der technischen Themen beginnt, möchte ich mich bei meinen Betreuern Prof. Dr.-Ing. Thomas Jung und Julien Letellier bedanken. Die zahlreichen Anregungen haben mir sehr dabei geholfen, den Prozess des Software Engineerings zu durchlaufen und der Arbeit eine passende Struktur zu geben. Ein Dank gilt auch meinem Kommilitonen Jonas Grabber, der sich trotz Urlaub bereiterklärt hat, die Arbeit Korrektur zu lesen.

Berlin, 05.09.2019
Donat Deva Brzoska

Zusammenfassung

Diese Arbeit beschäftigt sich mit der algorithmischen Erzeugung von Pflanzenstrukturen, insbesondere Bäumen und Sträuchern. Dazu wird ein Überblick zu einigen Verfahren gegeben und anschließend eine prototypische Software entwickelt. Ziel der Software ist es, eine interaktive Modellierung vieler verschiedener Bäume und Sträucher zu ermöglichen, wobei nur wenige anpassbare Parameter verwendet werden. Dazu wird der *Space Colonization Algorithmus* von Runions et al. genutzt, um eine prozedurale Komponente ergänzt und seine Laufzeit für die interaktive Verwendung optimiert. Herunterhängende Äste werden mit einem selbstentwickelten Verfahrens modelliert. Die Zielstellung wird so zu einem Großteil erreicht, für eine zukünftige Version der Software sollte allerdings ein erweitertes Verfahren benutzt werden.

Abstract

This thesis focuses on the algorithmic generation of plant like structures, especially trees and shrubs. First an overview of some popular methods is given, afterwards a prototypical software is developed. Its goal is to make an interactive modeling of various trees and shrubs possible, while using only a small amount of adjustable parameters. To achieve this, the *Space Colonization Algorithm* proposed by Runions et al. is used, slightly complemented with a procedural component and optimized for usage at runtime. Pendulous branches are modeled with a self-developed method. The results mostly fulfill the requirements, though for a future version of the software, an extended method should be used.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation und Zielstellung	1
1.2 Aufbau der Arbeit	2
2 Grundlagen	3
2.1 Botanische Grundlagen	3
2.1.1 Knospung	3
2.1.2 Verzweigung	3
2.1.3 Wachstumszeitpunkte	4
2.1.4 Baumformen	5
2.1.5 Tropismen	5
2.2 Generalisierte Zylinder	6
2.3 Usability	8
3 Anforderungserhebung und -analyse	9
3.1 Bisherige Softwarelösungen	9
3.2 Analyse der Problemstellung	11
3.2.1 Parameterwahl	11
3.2.2 Usability	13
3.2.3 Modellierprozess	13
3.2.4 Export	14
3.3 Anforderungsdefinition	14
3.3.1 Funktionale Anforderungen	14
3.3.2 Nicht-funktionale Anforderungen	15
3.4 Modellierung von Verzweigungsstrukturen	15
3.4.1 Prozedurale Ansätze	16
3.4.2 Lindenmayer-Systeme	17
3.4.3 Regelbasierte Objekterzeugung	22
3.4.4 Wachstumssimulation mittels Punktwolken	23
3.4.5 Herunterhänge Äste	26
3.5 Modellierung von Blättern	28
3.6 Kamerasteuerung	29
3.7 Entwicklungsumgebungen	29

Inhaltsverzeichnis

4 Entwurf	31
4.1 Auswahl der Modellierungsansätze	31
4.2 Benutzeroberfläche	34
4.3 Softwarearchitektur	34
5 Implementierung	36
5.1 Datenstruktur und Geometriedaten	36
5.2 Erzeugung der Verzweigungsstruktur	39
5.2.1 Grundlegende Klassen	39
5.2.2 Optimierung der Distanzberechnungen	43
5.3 Kern	45
5.4 Benutzeroberfläche und Controller	46
5.5 Abläufe	49
5.5.1 Änderung von GrowthProperties	49
5.5.2 Änderung von GeometryProperties	50
5.5.3 Neuer Seed	51
5.5.4 Export	51
6 Evaluierung	52
6.1 Erfüllung der Anforderungen	52
6.1.1 Funktionale Anforderungen	52
6.1.2 Nicht-funktionale Anforderungen	58
6.2 Limitationen	58
7 Zusammenfassung und Ausblick	61
A Anhang	63
Abbildungsverzeichnis	66
Quellenverzeichnis	67

1 Einleitung

Schon seit einem halben Jahrhundert beschäftigen sich Botaniker und Informatiker mit der synthetischen Erzeugung von Bildern natürlicher Objekte [1, S. 1]. Dabei handelt es sich um eine komplexe Thematik mit vielen Teilbereichen. Denn Pflanzen sind Lebewesen und unterliegen einem kontinuierlichen Wachstumsprozess, dessen Verlauf neben der Pflanzenart von einer Vielzahl weiterer Faktoren abhängt. Dazu gehören die Lichtverhältnisse, die Menge an verfügbarem Platz in der Umgebung, die Verteilung der Feuchtigkeit im Boden und die Gravitation. Das Resultat des Wachstumsprozesses sind in der Regel komplexe Gebilde, dessen geometrische Repräsentation für die Darstellung noch einmal eine eigene Herausforderung darstellt. Denn anders als bei vielen menschengemachten Objekten wie Häusern, Stühlen und Tischen finden sich in der Pflanzenwelt vorrangig runde Strukturen, die auch ineinander übergehen. Trotz allem ist das Gebiet der algorithmischen Pflanzenerzeugung inzwischen gut erforscht und es können ausgesprochen detailgetreue Darstellungen ermöglicht werden. Anwendung finden diese in Animationsfilmen, Computerspielen und Software für Landschaftsplaner, Botaniker und Architekten [1, S. 1].

1.1 Motivation und Zielstellung

Es existieren zahlreiche Softwaresysteme, um Pflanzen im dreidimensionalen Raum zu modellieren. Die allermeisten davon sind jedoch eher 3D-Modelliertools mit Spezialisierung auf Pflanzen. Über eine Vielzahl von anpassbaren Parametern können damit sehr präzise alle möglichen Strukturen erstellt werden. Der Modellieraufwand ist dabei relativ hoch und eine Einarbeitung in die oft komplexe grafische Benutzeroberfläche nötig. Viele Systeme bieten hier vorgefertigte Modelle an, die dann modifiziert werden können. Das Problem der komplexen Benutzeroberfläche bleibt allerdings bestehen und dient im Rahmen dieser Arbeit als Motivation dafür, eine einfachere Lösung zu schaffen. Ziel ist es, eine Software zu entwickeln mit deren Hilfe unter Verwendung von möglichst wenigen anpassbaren Parametern, eine dennoch breite Masse an Bäumen und Sträuchern erzeugt werden kann. Für die Weiterverwendung in der Spieleentwicklung soll außerdem ein Export möglich sein. Um die Komplexität der Aufgabe im Rahmen zu halten, wird ein minimalistisches Design der erzeugten Pflanzen angestrebt. Die Form der Pflanzen soll zwar realistisch aussehen aber Details bezüglich der Texturierung und geometrischen Repräsentation der Modelle spielen vorerst keine Rolle.

1.2 Aufbau der Arbeit

Im nächsten Kapitel werden kurz wichtige Begriffe aus der Botanik erläutert und ein Ansatz zur geometrischen Repräsentation botanischer Objekte beschrieben. In [Kapitel 3](#) wird eine Auswahl bekannter Modelliertools für Pflanzen vorgestellt. Anschließend erfolgt die Erhebung konkreter Anforderungen an die geplante Software und es wird ein Überblick zu einigen Ansätzen für die algorithmische Pflanzenerzeugung gegeben. In [Kapitel 4](#) werden passende Ansätze zur Modellierung ausgewählt und die grundlegende Architektur der Software entworfen. [Kapitel 5](#) beschreibt die konkrete Implementierung, indem die zentralen Klassen und Abläufe erläutert werden. [Kapitel 6](#) gibt eine Übersicht zu den erzielbaren Ergebnissen und evaluiert die Software auf Basis der zuvor definierten Anforderungen. Den Abschluss bilden eine Zusammenfassung und ein Ausblick zu Verbesserungsmöglichkeiten in zukünftigen Versionen der Software.

2 Grundlagen

2.1 Botanische Grundlagen

Um den visuellen Aspekt verschiedener Bäume und Sträucher modellieren zu können, ist es sinnvoll ihren Wachstumsprozess zu verstehen. Dieser wird durch ihr Verzweigungsverhalten und äußere Einflüsse bestimmt und definiert zusammen mit der Anordnung und Form der Blätter das äußere Erscheinungsbild einer Pflanze [1, S. 9f.].¹ Der folgende Abschnitt gibt einen kleinen Einblick in einige zentrale Fachbegriffe aus der Botanik.

2.1.1 Knospung

Jeder neue Trieb einer Pflanze beginnt sein Wachstum in einer Knospe. Unterschieden wird hierbei zwischen Gipfelknospen und Seitenknospen. Gipfelknospen wachsen am Ende eines Triebes und Seitenknospen (auch Achselknospen genannt) entstehen in der Achsel von Tragblättern. Eine Knospe kann sich entweder zu einem Ast mit Blättern oder zu einer Blüte entwickeln. Bereiche der Sprossachse aus denen neue Blätter und Knospen hervorgehen, werden als Knoten bezeichnet, der Bereich zwischen zwei Knoten als Internodium. Die Anzahl und Anordnung der Blätter, die aus einem Knoten wachsen kann ebenfalls variieren, wobei die Winkel zwischen benachbarten Blättern stets gleich bleiben (Äquidistanzregel). [1, S. 13f.]

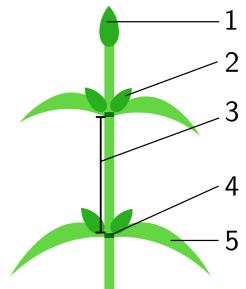


Abbildung 1:

- 1: Gipfelknospe
- 2: Seitenknospe
- 3: Internodium
- 4: Knoten
- 5: Tragblatt

2.1.2 Verzweigung

Die Art der Verzweigung wirkt sich direkt auf das Aussehen von Pflanzen aus. Es wird zwischen *monopodialer* und *sympodialer* Verzweigung unterschieden.

Bei der monopodialen Verzweigung dominiert das Wachstum der Hauptachse gegenüber dem der Seitenzweige. Allgemein formuliert wachsen Zweige niedrigerer Ordnung stärker als Zweige höherer Ordnung - in Abbildung 2 links wird dies veranschaulicht. [1, S. 14] Das Ergebnis ist ein *Monopodium*. [4, S. 3] Viele Nadelbäume wie beispielsweise die Fichte sind

¹Dies ist nur oberflächlich betrachtet korrekt. Denn eigentlich ist die Form einer Pflanze und ihrer Teile durch deren Funktion bestimmt. Im Kontext der visuellen Darstellung genügt es jedoch zunächst, sich auf die rein morphologische Sichtweise der Botanik zu konzentrieren. [1, S. 9]

2 Grundlagen

so aufgebaut. Der Austrieb von Seitenknospen wird bei diesen Bäumen hormonell gehemmt - dies wird als *Apikaldominanz* bezeichnet.² [5][6], S. 75]

Bei der sympodialen Verzweigung stellen die Knospen (bzw. die daraus entstehenden Äste) nach einer Wachstumsperiode ihr Wachstum ein oder bringen noch eine Blüte hervor. [7, S. 3] Die weitere Verzweigungsstruktur wird ausschließlich von den Zweigen höherer Ordnung bestimmt. Daraus resultiert entweder ein *Monochasium* oder ein *Dichasium*. [1, S. 14] Wie auch in Abbildung 2 (Mitte) zu sehen, wird bei einem Monochasium die Hauptachse durch einen dominanten Seitentrieb weitergeführt. Beispiele für Bäume mit monochasialem Wachstum sind Birken und Linden [8]. Bei einem Dichasium geht das Verzweigungssystem aus zwei gleichstark wachsenden Seitenachsen hervor, die unterhalb der Gipfelknospe austreiben. [4, S. 3f.] Ein Beispiel für eine dichasial wachsende Pflanze ist der Flieder. [5]

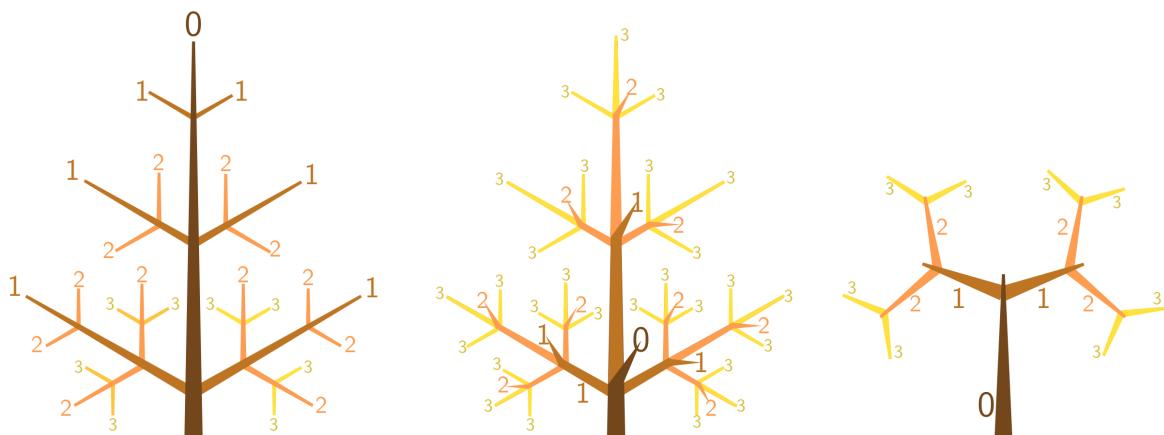


Abbildung 2:
Links: Monopodium; Mitte: Monochasium; Rechts: Dichasium

2.1.3 Wachstumszeitpunkte

Bezüglich der Wachstumszeitpunkte von Knospen wird zwischen *sylleptischem* und *proleptischem* Wachstum unterschieden. Proleptisches Wachstum bedeutet, dass eine Knospe nach ihrer Entstehung eine Ruhephase beginnt und erst im darauf folgenden Jahr weiter wächst. Bei sylleptischem Wachstum können neu entstandene Knospen noch in der Wachstumsperiode weiter wachsen, in der auch ihr übergeordneter Zweig entstanden ist (siehe Abbildung 3). [7, S. 3]

²Es existiert noch ein ähnlicher Begriff - Apikalkontrolle (Übersetzung des Autors aus dem englischen: "apical control"). Hierbei wird nicht das Austreiben von Knospen, sondern das Wachstum bereits existierender Seitentriebe gehemmt. [6, S. 75] Für die Betrachtungen dieser Arbeit ist die genauere Differenzierung jedoch nicht wichtig, sondern nur, ob generell das Wachstum seitlicher Zweige gehemmt wird oder nicht.

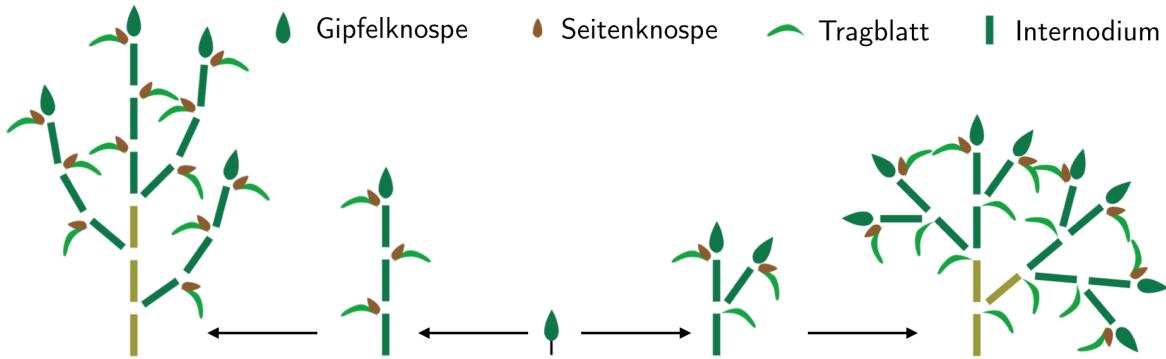


Abbildung 3: Darstellung von drei Wachstumsperioden;
Nach links: Proleptisches Wachstum; Nach rechts: Sylleptisches Wachstum

2.1.4 Baumformen

Im Rahmen dieser Arbeit ist vor allem entscheidend, wie sich die eben beschriebenen Wachstumsverhaltensweisen auf die Baumform auswirken.

Wenn der Haupttrieb stärker als die Seitentriebe wächst, entsteht eine schmale, konisch geformte Baumkrone mit einem klar erkennbaren Hauptstamm. Diese Struktur wird als *exkurrent* bezeichnet und resultiert aus proleptischem Wachstum mit monopodialer Verzweigung.

Wachsen die Seitentriebe genauso schnell oder schneller als der Haupttrieb, entsteht eine breitere Baumkrone mit mehreren Hauptästen. Diese Struktur wird als *dekurrent* bezeichnet und wird durch alle anderen Wachstumsarten erreicht. [9, S. 2] Junge Bäume weisen hierbei oftmals noch eine klar erkennbare Hauptachse auf (sind also eher exkurrent geformt) und entwickeln erst mit zunehmendem Alter eine dekurrente Form. [10, nach [7, S. 5]]

2.1.5 Tropismen

Die bevorzugte Wuchsrichtung der Triebe einer Pflanze wird neben der Verzweigungsart auch durch die Umgebungsbedingungen beeinflusst. Durch äußere Einflüsse ausgelöste Verformungen der Zweige werden als *Tropismen* bezeichnet. [1, S. 16f.] Die Pflanze empfängt hierbei Reize aus einer bestimmten Richtung und abhängig von der Reizrichtung und Reizart erfolgt lokal eine Anpassung der Wachstumsrichtung. Orientiert sich das Wachstum parallel zur Reizrichtung, spricht man von *Orthotropismus*. Eine Orientierung schräg zur Reizrichtung (in einem Winkel, der von der Reizrichtung abweicht [11]), wird als *Plagiotropismus* bezeichnet. Beträgt der Winkel zwischen Wachstumsrichtung und Reizrichtung 90° , ist die Rede von *Transversaltropismus* oder *Diatropismus*. Eine Hinwendung zur Reizquelle wird als *positiver Tropismus* bezeichnet und eine Abwendung als *negativer Tropismus*. [12]

Im Kontext dieser Arbeit sind vor allem Gravitation und Licht entscheidende Einflussfaktoren. Die Haupttriebe von Pflanzen streben meist in negativ gravitrope Richtung, wachsen also entgegen der Schwerkraft, und ermöglichen ihren Seitentrieben somit eine gute Raumausnutzung. Hauptwurzeln hingegen wachsen positiv gravitrop in den Boden hinein und Blätter richten sich transversal gravitrop, also im rechten Winkel zum Schwerkraftvektor aus. Die

Wachstumsrichtung von Seitentrieben orientiert sich am einfallenden Licht - es erfolgt eine transversal fototrope Ausrichtung. [1, S. 17]

2.2 Generalisierte Zylinder

Bei der geometrischen Repräsentation von Pflanzen handelt es sich wegen ihrer oftmals komplexen Verzweigungsstruktur um komplexe Thematik. Aufgrund des begrenzten zeitlichen Rahmens wird im Folgenden für die Repräsentation der Äste der erzeugten Bäume und Sträucher auf das Konzept der generalisierten Zylinder zurückgegriffen. Generalisierte Zylinder sind prinzipiell dreidimensionale Linien und werden nach Agin [13] nach [14, S. 306]] durch eine beliebige Kurve im Raum definiert. Senkrecht dazu werden Querschnitte ebenfalls beliebiger Form angeordnet. Im Fall von Bäumen und Sträuchern ist es sinnvoll, Kreise als Querschnitte zu verwenden (siehe Abbildung 4 links). Die Kreise können dann mittels Dreiecken zu einer geschlossenen Oberfläche verbunden werden - ein Beispiel ist in Abbildung 4 rechts zu sehen.

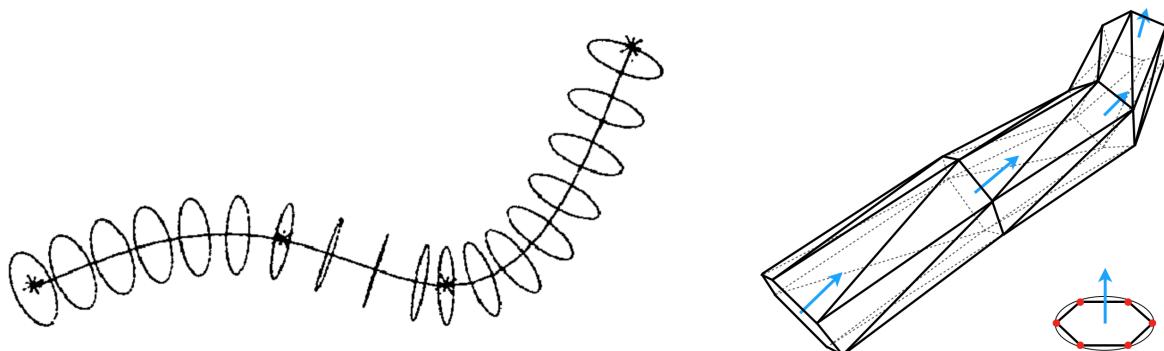


Abbildung 4: Generalisierte Zylinder

Die Anzahl der sich ergebenden Vertices und Dreiecke hängt davon ab, wie viele Querschnitte für die Kurve gewählt werden und wie hoch die Auflösung der Kreise ist. Verzweigungen können modelliert werden, indem einzelne Kreise nicht nur einen, sondern mehrere Nachfolger haben und zu allen diesen verbunden werden.

Die Dreiecke in Abbildung 4 rechts sind auf Basis von vier Kreisen gezeichnet worden. Diese werden hier jeweils durch sechs Punkte im dreidimensionalen Raum repräsentiert (siehe Abbildung 4 rechts unten in rot). Zur Berechnung der x-, y- und z-Koordinaten der Eckpunkte eines solchen Sechsecks müssen drei Parameter bekannt sein:

1. Die Position K des Sechsecks, repräsentiert durch seinen Mittelpunkt
2. Der Radius r des Kreises, der das Sechseck aufspannt
3. Die Ausrichtung des Sechsecks, repräsentiert durch eine Normale \vec{n} , in Abbildung 4 rechts als blaue Pfeile dargestellt

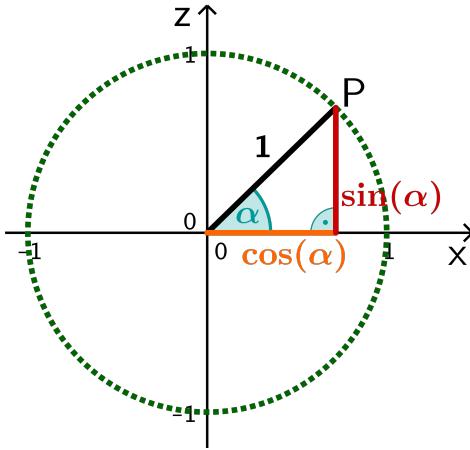


Abbildung 5: Einheitskreis

Die Berechnung der Koordinaten der Eckpunkte des Sechsecks erfolgt nun in vier Schritten:

1. Abbildung 5 zeigt, wie sich mittels Sinus und Kosinus unter Angabe eines Winkels α die Koordinaten beliebiger Punkte auf dem Einheitskreis berechnen lassen. Im dreidimensionalen Raum gilt:

$$P = \begin{pmatrix} \cos(\alpha) \\ 0 \\ \sin(\alpha) \end{pmatrix} \quad (1)$$

Im Fall eines Sechsecks werden nacheinander Koordinaten für sechs Werte von Alpha berechnet, wobei Alpha in jedem Schritt um 60° vergrößert wird ($\frac{360}{6} = 60$). Für einen Kreis beliebiger Auflösung w , ergeben sich w Punkte P_i deren Koordinaten wie folgt berechnet werden können:

$$P_i = \begin{pmatrix} \cos\left(\frac{i}{w} \cdot 360\right) \\ 0 \\ \sin\left(\frac{i}{w} \cdot 360\right) \end{pmatrix}, \quad i = 0, \dots, w - 1 \quad (2)$$

2. Die berechneten Koordinaten werden nun mittels Rotation anhand der definierten Normale \vec{n} ausgerichtet. Die Normale des Einheitskreises zeigt in positive y-Richtung. Der Winkel β , um den rotiert werden muss, wird also von der Normalen \vec{u} des Einheitskreises und der definierten Zielnormale \vec{n} aufgespannt. Die Achse \vec{s} , um die rotiert werden muss, steht senkrecht zu den beiden Normalen.

$$\vec{s} = \vec{u} \times \vec{n} \quad \text{und} \quad \beta = \angle(\vec{u}, \vec{n}) \quad , \quad \vec{u} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (3)$$

3. Die rotierten Kreiskoordinaten werden um den definierten Radius r skaliert. Jeder Punkt P_i auf dem Kreis wird dazu mit r multipliziert.
4. Die skalierten Kreiskoordinaten werden an die gewünschte Position K verschoben. Dafür wird zu jedem Punkt P_i auf dem Kreis K addiert.

2.3 Usability

Bei der Entwicklung interaktiver Anwendungen sollten grundlegende Aspekte der Usability einfließen. Teil 110 der internationalen Norm DIN EN ISO 9241 definiert dazu sieben Grundsätze zur Gestaltung interaktiver Systeme: [16], S. 61ff.]

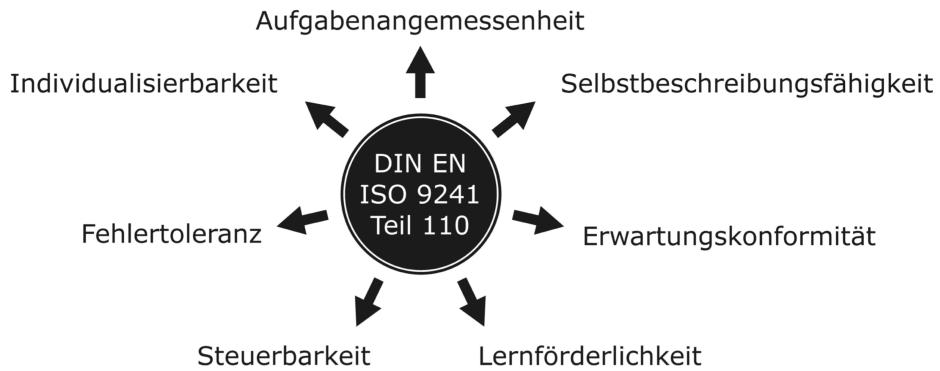


Abbildung 6: Grundsätze der Dialoggestaltung nach DIN EN ISO 9241-110

Aufgabenangemessenheit “Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe zu erledigen, d.h., wenn Funktionalität und Dialog auf den charakteristischen Eigenschaften der Arbeitsaufgabe basieren anstatt auf der zur Aufgabenerledigung eingesetzten Technologie.”

Selbstbeschreibungsfähigkeit “Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.”

Erwartungskonformität “Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht.”

Lernförderlichkeit “Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet.”

Steuerbarkeit “Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.”

Fehlertoleranz “Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.”

Individualisierbarkeit “Ein Dialog ist individualisierbar, wenn Benutzer die Mensch-System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen.”

3 Anforderungserhebung und -analyse

Ziel dieses Kapitels ist es, konkret umsetzbare Anforderungen zu definieren und anschließend mögliche Lösungsansätze zu analysieren. So wird die Basis für den Softwareentwurf sowie die Implementierung geschaffen und es kann in [Kapitel 6](#) objektiv bestimmt werden, ob die Software den Anforderungen genügt.

3.1 Bisherige Softwarelösungen

In diesem Abschnitt wird eine Auswahl der bereits existierenden Modelliertools für Pflanzen vorgestellt. Dabei wird deutlich, dass diese sich in ihrem Konzept ähneln und, dass für eine Software mit weniger Parametern vermutlich ein grundsätzlich anderer Ansatz benötigt wird.

SpeedTree

Ein bekanntes Tool zur Pflanzenmodellierung ist SpeedTree. Durch eine Art Baukastensystem ist es in wenigen Schritten möglich, gut aussehende Bäume zu erstellen. So wird immer mit einer Stammkomponente gestartet und an diese kann dann beispielsweise eine Verzweigungskomponente mit Ästen gehangen werden. Hängt man an die erste Verzweigungskomponente eine weitere, fügt dort noch Blätter hinzu und texturiert die Komponenten in SpeedTree, so ist das Ergebnis eine Baumstruktur wie sie in [Abbildung 7](#) links dargestellt ist. Rechts ist der zugehörige Graph abgebildet.



Abbildung 7: In SpeedTree erstellter Baum mit zugehörigem Graphen

Die Verzweigungskomponenten können vielfältig modifiziert werden. So ist es beispielsweise möglich, die Verzweigungswinkel oder Länge der jeweils untergeordneten Zweige zu ändern. Es können auch einzelne Äste ausgewählt und modifiziert oder entfernt werden. [17] Zweige können außerdem "per Hand" gezeichnet, abgebrochen oder gebogen werden und das Hinzufügen von Windeffekten ist möglich. [18] SpeedTree bietet noch unzählige weitere Features, die eine sehr präzise Modellierung beliebiger Bäume ermöglichen, an dieser Stelle wird darauf jedoch nicht weiter eingegangen, da die Details im Kontext dieser Arbeit nicht relevant sind.

Unity Tree Editor

Unity ist eine 3D-Entwicklungsumgebung und kann beispielsweise zur Entwicklung von Spielen oder Animationsfilmen genutzt werden. [19] In Unity enthalten ist unter anderem ein Tool, mit dem Bäume modelliert werden können. Der Modellierprozess erinnert dabei an den von SpeedTree - es wird mit einem Graphen aus Komponenten gearbeitet, die über eine Reihe von Parametern konfiguriert werden können. Windeffekte sind ebenfalls erzeugbar. [20] Im Gegensatz zu SpeedTree ist der Tree Editor in Unity kostenlos, der Funktionsumfang dafür jedoch um einiges geringer.

Tree It

Ein weiteres kostenloses Tool zur Baummodellierung ist das Tool "Tree It". Anders als bei SpeedTree und Unitys Tree Editor wird kein variabler Graph aus Komponenten verwendet. Stattdessen ist vorgegeben, dass es einen Hauptstamm, daran angebrachte Hauptäste und wiederum daran angebrachte kleine Äste gibt. Die drei Hauptkomponenten können wieder über verschiedene Parameter angepasst werden. Das Hinzufügen von Wurzeln und Blättern ist wie auch in SpeedTree und Unitys Tree Editor möglich. [21]

3.2 Analyse der Problemstellung

Verglichen mit einer manuellen Modellierung mittels beispielsweise Blender¹, erleichtern die bisherigen Tools das Erstellen von Bäumen bedeutend. Dabei müssen jedoch für zunehmend realistischere Ergebnisse auch immer mehr Parameter kontrolliert werden, so dass sich die Benutzeroberfläche entsprechend komplex gestaltet. Der erste Schritt zur Lösung dieses Problems ist es zu analysieren, welche Merkmale für die äußere Erscheinung von Bäumen und Sträuchern am bedeutendsten sind. In den Unterabschnitten danach wird auf zentrale Kriterien zur Usability und dem Modellierprozess, sowie den Export der erzeugten Modelle eingegangen.

3.2.1 Parameterwahl

Die anpassbaren Parameter müssen so gewählt sein, dass möglichst wenige und vor allem intuitiv verständliche Parameter eine breite Masse an Ergebnissen liefern. Dazu muss zunächst verstanden werden, welche Merkmale die äußere Erscheinung von Bäumen und Sträuchern ausmachen. Die wichtigsten Merkmale können im Idealfall später direkt in anpassbare Parameter überführt werden. So wird vom Nutzer nicht verlangt, zu definieren, wie das Ergebnis entstehen soll, sondern nur welches Ergebnis gewünscht ist. Die Komplexität der Benutzeroberfläche könnte auf diesem Weg verringert werden.

Wurzeln Alle Bäume und Sträucher besitzen Wurzeln, über die sie im Boden verankert sind und Nährsalze sowie Wasser beziehen. Typischerweise befindet sich der Großteil der Wurzeln unter der Erde und ist somit nicht sichtbar. Bei älteren Bäumen sind oft noch die Wurzelansätze zu sehen. Im weiteren Verlauf dieser Arbeit werden die Wurzeln eine zweitrangige Bedeutung haben, da sie - wenn überhaupt - nur einen kleinen Teil des Gesamteindrucks ausmachen.

Stammdicke Die Stammdicke eines jeden Baumes und Strauches nimmt "nach oben hin" ab, oder anders formuliert, jüngere Zweige sind dünner als ältere. Zudem besitzen gerade ältere Bäume einen dicken Stamm. Bei der Stamm- und Zweigdicke handelt es sich also um ein essentielles Merkmal, das für eine realistische Darstellung im weiteren Verlauf bedeutend sein wird.

Stammlänge Die Länge des initialen Stammes einer Pflanze bestimmt ebenfalls maßgeblich ihre äußere Erscheinung. Strauchförmige Pflanzen besitzen tendenziell keinen initialen Stamm - ihre Baumkrone beginnt sozusagen direkt am Boden. Bei Bäumen ist dies anders, eine Konfigurationsmöglichkeit der initialen Stammlänge ist also sinnvoll.

Borke Die Borke eines jeden Baumes und Strauches bildet ein weiteres optisch auffallendes Merkmal. Insbesondere bei Bäumen ist es häufig möglich, anhand der Borkenfarbe und -oberflächenstruktur eine Baumart zuordnen, ohne die Blätter oder Verzweigungsstruktur zu betrachten. Im Rahmen dieser Arbeit wird eher ein minimalistisches Design der Pflanzen angestrebt. Aus diesem Grund wird die Borkenfarbe im weiteren Verlauf im Vordergrund stehen, während die Oberflächenstruktur eine zweitrangige Rolle einnimmt.

¹Blender ist ein kostenloses Open Source Tool mit dem unter anderem beliebig komplexe dreidimensionale Objekte modelliert werden können. [22]

3 Anforderungserhebung und -analyse

Form Eine weitere bedeutende Eigenschaft ist die Form der Baumkrone. In [Kapitel 2](#) sind in diesem Zusammenhang bereits die Begriffe *dekkurrent* und *exkurrent* erläutert worden. Insbesondere bei dekkurrent geformten Bäumen, also denen mit mehreren Hauptästen, ist eine weitere Differenzierung sinnvoll. Eichen beispielsweise haben häufig eine annähernd kugelförmige Krone, Pappeln hingegen wachsen meist stärker in die Höhe als in die Breite (siehe [Abbildung 8](#) und [9](#)).



Abbildung 8: Eiche



Abbildung 9: Pappel

Verzweigungsdichte Die Anzahl der Verzweigungen von Sträuchern und vor allem Bäumen nimmt aufgrund der zunehmend besseren Lichtverhältnisse nach außen hin zu [\[25\]](#). Diese Eigenschaft prägt das Aussehen der Verzweigungsstruktur maßgeblich und wird deshalb im weiteren Verlauf bedeutend sein.

Herunterhängende Äste Manche Bäume wie die Birke oder Trauerweide zeichnen sich durch besonders stark herunterhängende dünne Äste aus. So ist es oft schon von weitem erkennbar um welchen Baum es sich handelt. Wie sehr die Äste eines Baumes herunterhängen, spielt demzufolge eine wichtige Rolle für die äußere Erscheinung.

Astkrümmung Bäume wie die Korkenzieherweide sind gut daran erkennbar, dass ihre Äste stark gekrümmt, also nicht gerade wachsen. Da es sich um ein Merkmal handelt, dass mit dieser Intensität nicht bei vielen Bäumen vorkommt, erhält die Astkrümmung zunächst eine zweitrangige Bedeutung.

Blätter Neben der Form der Äste und deren Art zu verzweigen, lassen sich Bäume häufig am einfachsten bestimmen, indem die Form der Blätter betrachtet wird. Für den optischen Gesamteindruck aus etwas Abstand ist jedoch oftmals viel entscheidender, wie hoch die Blattdichte ist und wie die Blätter gefärbt sind. Aus diesem Grund - und auch zur Vereinfachung - werden im Folgenden vor allem die beiden letztgenannten Merkmale Bedeutung bekommen.

Blüten und Früchte In der Blütezeit fallen Bäume und Sträucher stark durch ihre jeweilige Blütenfarbe auf. Wenn die Blüten sich zu Früchten entwickeln, genügt häufig ein Blick auf die Frucht, um die Pflanzenart zu bestimmen. Da die Zeit, in der Bäume und Sträucher

3 Anforderungserhebung und -analyse

blühen und Früchte tragen, eher begrenzt ist, wird deren Bedeutung zunächst eine zweitrangige Rolle spielen. Dies reduziert zudem die Komplexität der Software, wobei eine spätere Erweiterung diesbezüglich vermutlich einfach umzusetzen ist.

Alter Ein letztes entscheidendes Merkmal ist das Alter einer Pflanze - jüngere Bäume sind klein und besitzen wenige Äste. Ältere Bäume hingegen fallen durch ihre Größe auf und die Vielzahl der Verzweigungen führt mit zunehmendem Alter zu einer starken physischen Präsenz. Eine Anpassbarkeit dieses Parameters wird für den weiteren Verlauf in jedem Fall bedeutend sein.

3.2.2 Usability

Der Fokus der Arbeit liegt zwar primär auf der Reduktion der verwendeten Parameter, da es sich jedoch um eine interaktive Anwendung handelt, sollten die sieben Grundsätze zur Dialoggestaltung ([Abschnitt 2.3](#)) in den Entwicklungsprozess einfließen. Für den spezifischen Einsatzbereich der Software werden zusätzlich einige Details ergänzt. So sollte eine Interaktion mit dem Programm jederzeit möglich sein und Feedback an den Nutzer geben. Wenn beispielsweise das Programm mit Berechnungen beschäftigt ist, sollte der Nutzer trotzdem währenddessen die Möglichkeit haben, es zu beenden oder Anpassungen an den Parametern vorzunehmen. Dabei sollten die Anpassungen sofort in die Berechnungen einfließen, d.h. es muss nicht erst auf das Ende der gerade laufenden Berechnungen gewartet werden. Weiterhin ist es sinnvoll, dem Nutzer den derzeitigen Fortschritt der Berechnungen mitzuteilen. Im eben genannten Beispiel würde das bedeuten, dass ein fertig berechneter Teil der Pflanzenstruktur auch sofort dargestellt wird.

Für ein flüssiges Arbeiten sollte das Programm außerdem schnell arbeiten, so dass keine unnötig langen Wartezeiten entstehen. Eine stabile Software fördert ebenfalls ein angenehmes Benutzererlebnis, da es nicht zu unerwünschten Unterbrechungen kommt. Eine Garantie für letzteres kann nur erreicht werden, indem umfassende Tests geschrieben werden. Da dies den Entwicklungsaufwand signifikant erhöhen würde, spielt die Stabilität im Folgenden eine zweitrangige Rolle.

3.2.3 Modellierprozess

Der Aufwand für die Modellierung soll möglichst gering sein. Um dies zu ermöglichen, ist es sinnvoll, eine initiale Struktur bereitzustellen, die dann bezüglich der bereits genannten Parameter beliebig modifiziert werden kann. Es sollte außerdem möglich sein, auf Basis der schon definierten Einstellungen eine neue zufällige Baumstruktur zu generieren. So kann beispielsweise eine gewünschte Form und ein Alter definiert und dann beliebig viele verschiedene Pflanzenstrukturen mit diesen Eigenschaften erzeugt werden.

In einem 3D-Spiel werden die erzeugten Bäume und Sträucher vermutlich von allen Seiten betrachtet werden können. Deshalb sollte dies auch während der Modellierung möglich sein, was eine entsprechende Kamerasteuerung erfordert.

Programmierung ist prinzipiell auf allen Betriebssystemen möglich, gleiches gilt also auch für die Spieleentwicklung. Aus diesem Grund sollte die Software für die gängigsten Betriebssysteme Linux, macOS und Windows verfügbar sein.

3.2.4 Export

Damit die erzeugte Pflanzenstruktur weiterverwendet werden kann, muss ein Export in einem geeigneten Format möglich sein. 3D-Objekte werden typischerweise über Vertices und Dreiecke definiert. Mittels Normalen können die Lichtverhältnisse am Objekt genauer spezifiziert werden und UV-Koordinaten ermöglichen eine Texturierung. Die eben genannten Informationen (im Folgenden als Geometriedaten bezeichnet) können in Wavefront Technologies OBJ-Format kodiert werden. Das Format wird von den meisten bekannten Tools, die mit 3D-Modellen arbeiten, unterstützt [26] und die Kodierung in ASCII ist einfach [27]. Da Bäume in Spielen meist zur Umgebung gehören, sollte darauf geachtet werden, die Anzahl der Vertices und Dreiecke gering zu halten und gegebenenfalls anpassbar zu machen.

3.3 Anforderungsdefinition

Dieser Abschnitt dient einer strukturierten Zusammenfassung der im letzten Abschnitt beschriebenen Anforderungen. Dazu wird zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden, welche wiederum in die Kategorien *obligatorisch* (erforderlich) und *fakultativ* (optional) eingeteilt werden.

3.3.1 Funktionale Anforderungen

Obligatorisch:

FAO01 Die erzeugten Verzweigungsstrukturen müssen realistisch aussehen, also an echte Pflanzen erinnern. Der Gesamteindruck ist hierbei wichtiger als botanische Korrektheit.

FAO02 Dekurrente Strukturen mit modellierbarer Form müssen erzeugt werden können.

FAO03 Exkurrente Strukturen müssen erzeugt werden können.

FAO04 Es soll eingestellt werden können, wie sehr die Äste der erzeugten Pflanzenstrukturen herunterhängen.

FAO05 Die Farbe der Borke muss einstellbar sein.

FAO06 Die Anzahl und Farbe der Blätter muss einstellbar sein.

FAO07 Das Alter der Pflanzenstruktur muss einstellbar sein.

FAO08 Bei Programmstart muss eine zufällig erzeugte dreidimensionale Baumstruktur mit Blättern dargestellt werden.

FAO09 Die Generierung eines neuen Seeds muss möglich sein, wobei alle eingestellten Parameter ihre Werte behalten.

3 Anforderungserhebung und -analyse

FAO10 Die Benutzeroberfläche muss es ermöglichen, die derzeit erstellte Struktur von allen Seiten zu betrachten.

FAO11 Erzeugte Pflanzenstrukturen müssen im OBJ-Format inklusive ihrer Textur exportiert werden können.

FAO12 Die Stammdicke muss einstellbar sein.

FAO13 Die Stammlänge muss einstellbar sein.

FAO14 Der Rechenfortschritt der Software muss angezeigt werden.

FAO15 Die Anzahl der Vertices und Dreiecke soll möglichst niedrig sein.

Fakultativ:

FAF1 Es soll gewählt werden können, ob die erzeugte Pflanzenstruktur Wurzelansätze hat oder nicht.

FAF2 Der Grad der Astkrümmung soll eingestellt werden können.

FAF3 Die Textur der Borke sowie der Blätter soll einstellbar sein.

FAF4 Blüten und Früchte sollen hinzugefügt werden können und bezüglich ihrer Anzahl, Farbe und Form anpassbar sein.

FAF5 Für den Export sollen verschiedene Auflösungen wählbar sein, um die Anzahl der Vertices und Dreiecke präziser regulieren zu können.

FAF6 Die Software sollte sich an den sieben Grundsätzen zur Dialoggestaltung orientieren.

3.3.2 Nicht-funktionale Anforderungen

Obligatorisch:

NFAO1 Eine Interaktion mit der Software muss jederzeit möglich sein und dem Nutzer Feedback geben.

NFAO2 Die Software muss auf den Betriebssystemen Linux, macOS und Windows ausführbar sein.

Fakultativ:

NFAO1 Die zugrunde liegenden Algorithmen sollen so optimiert sein, dass die Wartezeiten für den Nutzer möglichst gering sind.

NFAF2 Die Software sollte so stabil laufen, dass es nicht zu Abstürzen kommt.

3.4 Modellierung von Verzweigungsstrukturen

In den letzten Jahrzehnten wurden zahlreiche Verfahren entwickelt, um Pflanzenstrukturen algorithmisch zu erzeugen. Der folgende Abschnitt gibt einen Überblick zu einigen bedeutenden Ansätzen.

3.4.1 Prozedurale Ansätze

Die Bezeichnung “Prozedurale Ansätze” ist nicht wirklich akkurat, da Prozeduren bei allen möglichen Arten von Verfahren eine Rolle spielen. Im Folgenden sind damit Ansätze gemeint, die parametrisierte Algorithmen für die Pflanzenerzeugung verwenden. Das Verfahren von Oppenheimer [28] soll repräsentativ für prozedurale Ansätze erklärt werden. Sehr viele Pflanzen besitzen eine rekursive Verzweigungsstruktur, das heißt die Äste eines Baumes sehen, für sich stehend, ebenfalls einem Baum ähnlich, gleiches gilt auch für deren Unteräste und so weiter. Ein rekursiver Algorithmus kann somit genutzt werden, um Verzweigungsstrukturen nach folgender Definition zu erzeugen:

```
tree:= {
    Draw Branch Segment

    if (too small) {
        Draw leaf
    } else {
        #Continue to Branch

        Transform Stem
        "tree"

        repeat n times {
            Transform Branch
            "tree"
        }
    }
}
```

Listing 1: Definition eines rekursiv erzeugten Baumes, nach [28], S. 56]

Nach der Definition beginnt ein Baum immer mit einem Astsegment. Unterschreitet dieses eine definierte Größe, wird ein Blatt angehangen und es erfolgt keine weitere Verzweigung mehr. Andernfalls wird das Wachstum rekursiv fortgesetzt, indem weitere Bäume angehangen werden. Die Wachstumsrichtung und Größe der untergeordneten Bäume werden mittels einer linearen Transformation bestimmt. Oppenheimer verwendet dabei unterschiedliche Transformationen für die Erzeugung des Hauptstamms und der Seitenäste. Neben der Anzahl der Seitenäste können für die Transformation verschiedene Parameter eingestellt und somit unterschiedliche Verzweigungsstrukturen erreicht werden. Beispiele für Parameter sind:

- Der Winkel zwischen Hauptstamm und den Seitenästen
- Das Verhältnis der Größe zwischen Hauptstamm-Segmenten und Seitenast-Segmenten
- Die Rate mit der der Hauptstamm verjüngt wird
- Die Menge an helikaler Verdrehung zwischen den Segmenten

Damit keine zu regelmäßigen Bäume entstehen, wird die Transformationsmatrix während des Erzeugungsprozesses verändert. Dazu erhalten die Parameter einen Mittelwert und eine Standardabweichung. [28, S. 55ff.] Abbildung 10 zeigt eine so erzeugte Verzweigungsstruktur.

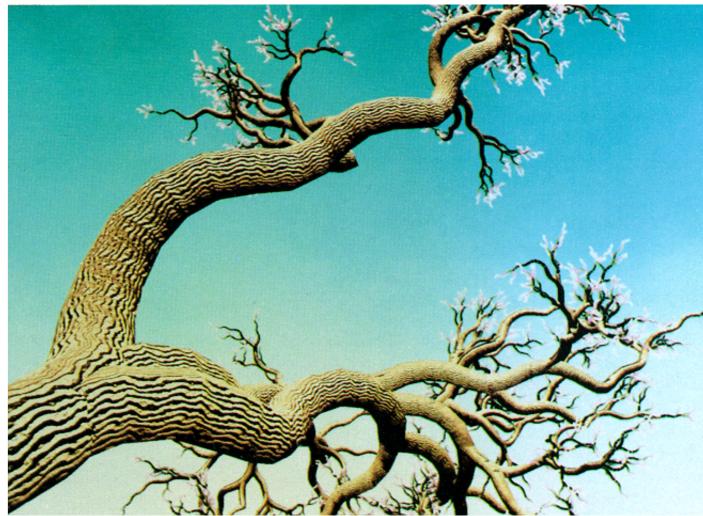


Abbildung 10: Baum nach Oppenheimer

Oppenheimer beschreibt in seinem Paper zwar nicht alle verwendeten Parameter, die abgebildeten Pflanzen lassen jedoch darauf schließen, dass sein Verfahren bezüglich der möglichen Vielfalt noch ausbaufähig ist. Weber und Penn [29] präsentierten 1995 ein prozedurales Verfahren, mit dem eine realistische Modellierung von sehr vielen verschiedenen Pflanzen möglich wird. Der Nachteil an dem Verfahren ist, dass es über 60 Parameter verwendet und somit für eine effektive Verwendung gut verstanden werden muss.

3.4.2 Lindenmayer-Systeme

Ein völlig anderer Ansatz zur Pflanzenmodellierung ist die Verwendung von Lindenmayer-Systemen, im Folgenden als “L-Systeme” bezeichnet. Aristid Lindenmayer entwickelte diese ursprünglich, um die Entwicklung von mehrzelligen Organismen zu modellieren, später wurde das Potential von L-Systemen auch auf die Modellierung von komplexen Verzweigungsstrukturen ausgeweitet. [30, S. vi] L-Systeme basieren auf dem Konzept der Ersetzungssysteme. Zeichenketten werden dabei nach bestimmten Vorschriften erstellt und anschließend grafisch interpretiert. [30, S. 1ff.]

DOL-Systeme

Die einfachste Form von L-Systemen sind DOL-Systeme und zeichnen sich dadurch aus, dass sie deterministisch und kontextfrei sind. Abbildung 11 veranschaulicht zunächst einmal das grundlegende Verhalten von L-Systemen. Am Anfang steht ein initiales Zeichen b und es existieren zwei Produktionsregeln: $b \rightarrow a$ und $a \rightarrow ab$. In jedem Ableitungsschritt werden alle Produktionsregeln gleichzeitig auf die derzeitige Zeichenkette angewandt. Aus b wird somit im ersten Schritt a . Das neue entstandene a wiederum wird im nächsten Schritt zu ab und so weiter.



Abbildung 11: Ableitung eines einfachen L-Systems

Ein DOL-System ist die deterministische Ausprägung eines OL-Systems. Allgemein formuliert ist ein OL-System definiert durch ein Tripel $G = \langle V, \omega, P \rangle$. Dabei ist V das Alphabet der zulässigen Zeichen, wobei V^* die Menge aller möglichen Wörter, und V^+ die Menge aller nichtleeren möglichen Wörter aus V repräsentieren. ω steht für eine nichtleere initiale Zeichenkette, auch *Axiom* genannt, es gilt: $\omega \in V^+$. P ist eine endliche Menge von Produktionsregeln der Form $a \rightarrow \chi$, wobei a als *Vorgänger* und χ als *Nachfolger* bezeichnet werden. a ist ein Zeichen aus V , und χ ein Zeichen oder eine Zeichenkette aus V^* .

Voraussetzung für die sinnvolle Ableitung eines L-Systems ist, dass für jedes Zeichen $a \in V$ eine Produktionsregel $a \rightarrow \chi$ existiert. Ist letzteres für ein (oder mehrere) $a \in V$ nicht der Fall, wird implizit die Identitäts-Produktionsregel $a \rightarrow a$ definiert, d.h. das Zeichen bleibt wie es ist. Damit ein OL-System als DOL-System bezeichnet werden kann, darf für ein konkretes $a \in V$ immer nur *eine* Ableitungsregel definiert sein. Dies führt dazu, dass ein OL-System stets zum gleichen vorhersehbaren Ergebnis führt - es ist deterministisch. OL-Systeme sind kontextfrei, da für die Ersetzung nur der Vorgänger bedeutend ist und seine Nachbarn keinerlei Einfluss auf die Anwendung der Produktionsregeln haben. [30, S. 3ff.]

Das Beispiel aus Abbildung 11 basiert nach den Definitionen auf folgendem L-System:

$$\begin{aligned}
 V &: \{a, b\} \\
 \omega &: b \\
 p_1 &: b \rightarrow a \\
 p_2 &: a \rightarrow ab
 \end{aligned} \tag{4}$$

Turtle-Interpretation

Damit Textersetzungssysteme für die Modellierung von Pflanzen sinnvoll genutzt werden können, ist eine anschließende graphische Interpretation nötig. Dazu wird eine Art Zeichenroboter verwendet, im Folgenden auch Turtle genannt. Eine Turtle nimmt verschiedene Befehle entgegen und ändert abhängig davon ihren Zustand. Der Zustand einer Turtle wird definiert durch ein Tripel $T = \langle x, y, \alpha \rangle$, wobei x und y für die Position der Turtle im zweidimensionalen Raum stehen und α für die Richtung in die sie schaut - Werte für α von 0-360° erlauben jede

3 Anforderungserhebung und -analyse

mögliche Ausrichtung im \mathbb{R}^2 . Zusätzlich wird eine *Schrittweite* d und eine *Winkeländerungsrate* δ definiert. Nun können folgende Befehle benutzt werden:

- F Die Turtle bewegt sich um die Schrittweite d vorwärts (d.h. in die Richtung in die sie schaut) und zeichnet dabei eine Linie
- f Die Turtle bewegt sich um die Schrittweite d vorwärts, ohne eine Linie zu zeichnen
- Die Turtle dreht sich im Uhrzeigersinn um die Winkeländerungsrate δ
- + Die Turtle dreht sich gegen den Uhrzeigersinn um die Winkeländerungsrate δ

Folgende Abbildung veranschaulicht drei der Befehle anhand eines Beispiels:

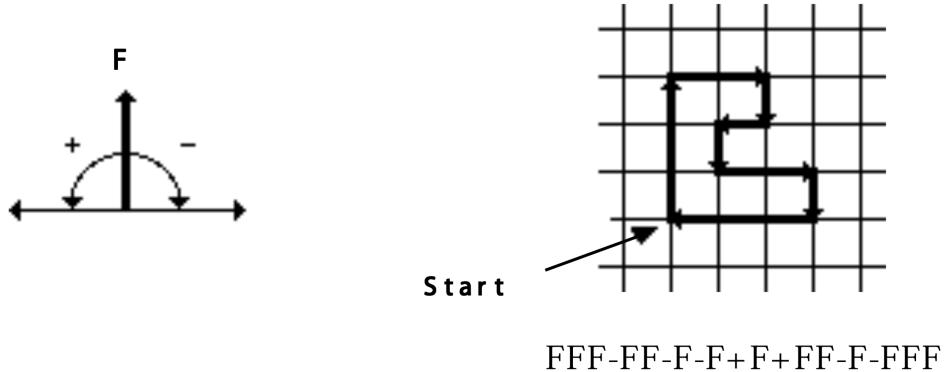


Abbildung 12: Funktionsweise einer Turtle

Links: Verhaltensdefinition; Rechts: Beispiel mit $\delta = 90^\circ$

Durch die Definition von L-Systemen, die Anweisungen der Turtle enthalten, können selbstähnliche Strukturen produziert werden. [30, S. 6f.] Ein Beispiel hierfür ist das folgende L-System:

$$\begin{aligned} V &: \{F, -, +\} \\ \omega &: F \\ p &: F \rightarrow F + F - -F + F \end{aligned} \tag{5}$$

Mit $\delta = 60^\circ$ nähert es, wie auch in Abbildung 13 dargestellt, die Koch-Kurve an.



Abbildung 13: Konstruktion der Koch-Kurve, ganz rechts das Ergebnis

Die Koch-Kurve besteht prinzipiell aus einer einzigen durchgehenden Linie, da immer ausgehend von der zuletzt erreichten Position gezeichnet wird. Pflanzliche Verzweigungsstrukturen lassen sich jedoch besser modellieren, indem auch Sprünge möglich sind. Zu diesem Zweck nimmt die Turtle zwei weitere Befehle entgegen:

3 Anforderungserhebung und -analyse

- [] Der Zustand der Turtle wird auf einem Stack gespeichert
- [] Der zuletzt gespeicherte Zustand der Turtle wird wiederhergestellt und vom Stack entfernt

So kann beispielsweise zu einem Knotenpunkt zurückgekehrt werden, nachdem eine daran angebrachte Verzweigungsstruktur gezeichnet wurde. [30, S. 24] Ein etwas komplexeres L-System, das davon Gebrauch macht, lautet:

$$\begin{aligned}
 V &: \{F, -, +, [,]\} \\
 \omega &: X \\
 p_1 &: F \rightarrow FF \\
 p_2 &: X \rightarrow F[+++FX]F[---FX]F[++F-FX][F-FX]
 \end{aligned} \tag{6}$$

Interpretiert durch eine Turtle mit $\delta = 10^\circ$ ergibt sich folgende Struktur:

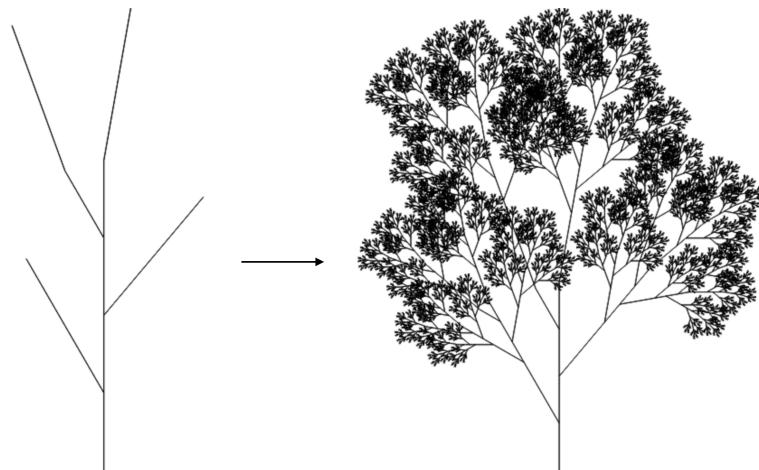


Abbildung 14:

Links: "Initiale" Struktur; Rechts: Pflanzenstruktur nach 7 Ersetzungsschritten

Modellierung im dreidimensionalen Raum

Die bisher erläuterten L-Systeme und ihre Turtle-Interpretation arbeiteten im zweidimensionalen Raum. Im dreidimensionalen Raum erfolgt die Ausrichtung der Turtle anhand von drei Vektoren:

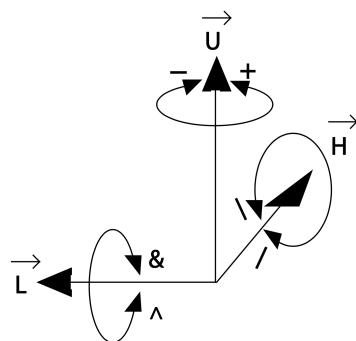


Abbildung 15: Ausrichtung und Befehle der Turtle im dreidimensionalen Raum

\vec{H} zeigt die Blickrichtung der Turtle an, \vec{U} wo oben und \vec{L} wo links ist. Die Vektoren stehen senkrecht aufeinander, so dass $\vec{H} \times \vec{L} = \vec{U}$ gilt, und haben eine Länge von 1. Zudem werden neue Befehle für die Ausrichtung der Turtle definiert: [30, S. 18f.]

- + Drehung um δ gegen den Uhrzeigersinn
- Drehung um δ im Uhrzeigersinn
- | Drehung um 180°
- & Neigung um δ nach unten
- \wedge Neigung um δ nach oben
- \ Seitliche Drehung um δ gegen den Uhrzeigersinn
- / Seitliche Drehung um δ im Uhrzeigersinn

Abbildung 16 zeigt ein Beispiel für eine dreidimensionale Pflanzenstruktur, die mit den eben erläuterten Befehlen erzeugt wurde.



Abbildung 16: Dreidimensionale Pflanzenstruktur nach 7 Ersetzungsschritten, $\delta = 22.5^\circ$

$$\begin{aligned}
 V &: \{F, f, A, S, L, -, +, |, \&, \wedge, \backslash, /, [,], ', !, \{, \}\} \\
 \omega &: A \\
 p_1 &: A \rightarrow [\&FL!A]// // /'[\&FL!A]// // // /'[\&FL!A] \\
 p_2 &: F \rightarrow S // // /F \\
 p_3 &: S \rightarrow FL \\
 p_4 &: L \rightarrow ['' \wedge \wedge \{-f + f + f - | - f + f + f\}]
 \end{aligned} \tag{7}$$

Produktionsregel p_1 generiert neue Äste ausgehend von der Spitze des übergeordneten Zweigs. Ein Ast besteht dabei immer aus einem Internodium F, einem Blatt L und einer Spitze A. Die Produktionsregeln p_2 und p_3 führen dazu, dass die Internodien wachsen und dabei neue Blätter produzieren, die um den Zweig herum angeordnet werden. Produktionsregel p_4 definiert ein Blatt, indem die Turtle sich zuerst vom Zweig wegdreht und dann eine sechseckige Form definiert.

3 Anforderungserhebung und -analyse

Zum Zweck einer realistischeren optischen Erscheinung werden hier weitere Befehle verwendet. Die geschweiften Klammern ({}{}) indizieren, dass alle Bewegungen der Turtle, die innerhalb der Klammern stattfinden die Umrandung einer Form definieren, die anschließend mit einer Farbe gefüllt wird. So wird das Zeichnen der Blätter ermöglicht. Ein Ausrufezeichen (!) verringert den Durchmesser der nachfolgend gezeichneten Segmente und das einfache Anführungszeichen oben (') inkrementiert den Index auf einer Farbtabelle, so dass nahe der Wurzel ein brauner und nahe der Zweigenden ein grüner Farbton verwendet wird. [30, S. 24ff.]

Stochastische, kontextsensitive und parametrische L-Systeme

An diesem Punkt wird bereits ein klarer Nachteil von L-Systemen deutlich: Dadurch, dass es sich um sehr abstrakte Repräsentationen der Pflanzen handelt, ist die Herstellung der Produktionsregeln nur wenig intuitiv [32, S. 2] und die Eignung für die interaktive Pflanzenerzeugung eher begrenzt. Deshalb soll auf detailliertere Ausführungen der Erweiterungen von L-Systemen verzichtet werden, es wird jedoch noch ein kurzer Ausblick gegeben.

Stochastische L-Systeme erlauben für einen Vorgänger verschiedene Nachfolger, die dann anhand von Wahrscheinlichkeiten für die Textersetzung ausgewählt werden. Dies ermöglicht die Erzeugung verschiedener Pflanzenstrukturen mit einem einzigen L-System. [30, S. 28]

Mit *kontextsensitiven L-Systemen* kann die Anwendung der Produktionsregeln von der Umgebung des Vorgängers abhängig gemacht werden. Die Produktionsregel wird dann jeweils nur angewandt, wenn die definierten linken und/oder rechten Nachbarn des Vorgängers vorhanden sind. Damit kann beispielsweise Hormonfluss innerhalb von Pflanzen simuliert werden. [30, S. 30]

Parametrische L-Systeme ermöglichen es zusätzlich, den Zeichen numerische Werte zuzuweisen, die dann über Produktionsregeln auch verändert werden können. Die dadurch gewonnene Flexibilität löst eine Reihe von Problemen, auf die hier jedoch nicht näher eingegangen wird. [30, S. 40]

Eine Kombination aus den eben beschriebenen Arten von L-Systemen ist ebenfalls möglich. Mittels der Modellersprache L+C können die genannten Arten von L-Systemen definiert und anschließend über das Simulationsprogramm lpfg dargestellt werden. L+C basiert auf C++ und erweitert die Mächtigkeit von L-Systemen zusätzlich zu den bereits genannten Möglichkeiten. Detailliertere Ausführungen können im Paper *Design and Implementation of the L+C Modeling Language* von Karwowski und Prusinkiewicz [33] nachgeschlagen werden.

3.4.3 Regelbasierte Objekterzeugung

Eine Art Mittelweg zwischen prozeduralen Verfahren und regelbasierten Verfahren wie L-Systemen wurde 1997 von Lintermann und Deussen [32] vorgestellt. Ziel des Verfahrens ist es, die Mächtigkeit von L-Systemen mit der vergleichsweise intuitiven Arbeitsweise von prozeduralen Verfahren zu verbinden. Die regelbasierte Objekterzeugung arbeitet hierzu mit einer Reihe von Bausteinen, die parametrisiert und aneinander gehangen werden, um die verschiedensten Pflanzenmodelle zu erzeugen. [32, S. 1ff.] Bisherige Softwarelösungen wie SpeedTree oder Unitys Tree Editor benutzen dieses Konzept zur Erzeugung von Bäumen. So ist zwar eine präzise Modellierung möglich, es muss jedoch eine Vielzahl von Parametern

kontrolliert werden. Da letzteres für die geplante Software vermieden werden soll und das Konzept in [Abschnitt 3.1](#) bereits erläutert wurde, folgen an dieser Stelle keine weiteren Ausführungen zur regelbasierten Objekterzeugung.

3.4.4 Wachstumssimulation mittels Punktwolken

Mit einem Verfahren von Runions et al. [\[25\]](#) wird das Pflanzenwachstum simuliert, indem der Wettbewerb der Knospen um Platz eine zentrale Rolle einnimmt. Da trotz des einfachen Konzepts gute Ergebnisse erzielt werden können, wird das Verfahren im Folgenden etwas ausführlicher beschrieben.

Im ersten Schritt wird auf Basis einer dreidimensionalen Hülle eine Punktwolke generiert ([Abbildung 17](#), a). Die Punkte repräsentieren Stellen, an denen noch Platz zum Wachsen ist, und werden entfernt, sobald sie von einem Ast erreicht werden. Auf Basis der Punktwolke wird nun in einem iterativen Prozess die Baumstruktur erzeugt ([Abbildung 17](#), b und c). Ausgehend von einem einzelnen Knoten unterhalb der Punktwolke werden neue Knoten hinzugefügt, wobei die Wachstumsrichtung immer durch die am nächsten liegenden Punkte bestimmt wird. Dies wird so lange fortgeführt, bis keine Punkte mehr übrig sind, keine Punkte mehr in Reichweite sind, oder eine festgelegte Anzahl an Iterationen erreicht wurde. An der resultierenden Knotenstruktur können noch Änderungen vorgenommen werden, um beispielsweise den runden Verlauf von Ästen besser zu modellieren und anschließend kann eine Überführung in eine geometrische Repräsentation zur Darstellung erfolgen ([Abbildung 17](#), d-f).

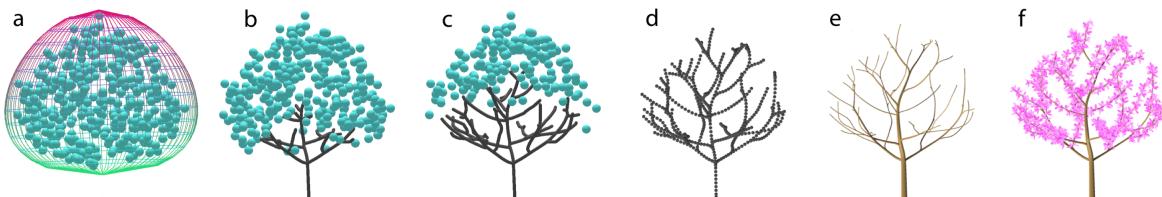


Abbildung 17: Grundlegende Schritte des Verfahrens

Die in [Abbildung 17](#) dargestellten Schritte b und c stellen die Essenz des Verfahrens dar und werden von Runions et al. als *Space Colonization Algorithmus* bezeichnet. Der Algorithmus wird im Folgenden zunächst formal erläutert und anschließend anhand eines Beispiels veranschaulicht.

Wie bereits erwähnt, wird die Wachstumsrichtung der Knoten von den am nächsten gelegenen Punkten aus der Punktwolke beeinflusst - diese werden auch als Anziehungspunkte bezeichnet. Eine festgelegte *Einflussdistanz* d_i bestimmt, ob ein Anziehungspunkt die Wachstumsrichtung eines Knotens beeinflusst oder nicht. Ein Knoten v kann auch von mehreren Anziehungspunkten beeinflusst werden - diese Menge der Anziehungspunkte in der Nähe wird hier als $S(v)$ bezeichnet. Ist $S(v)$ nicht leer, so wird ein neuer Knoten v' erstellt und an v angehängt. Der neue Knoten v' wird im Abstand D von v in Richtung \hat{n} positioniert. Zur Ermittlung der Richtung \hat{n} werden die Vektoren zu allen $s \in S(v)$ berechnet, addiert und das

3 Anforderungserhebung und -analyse

Ergebnis normalisiert. Es gilt also:

$$v' = v + D\hat{n} \quad (8)$$

wobei:

$$\hat{n} = \frac{\vec{n}}{\|\vec{n}\|} \quad \text{und} \quad \vec{n} = \sum_{s \in S(v)} \frac{s - v}{\|s - v\|} \quad (9)$$

Optional kann die Wachstumsrichtung noch durch einen Vektor \vec{g} beeinflusst werden, um beispielsweise Tropismen einzubeziehen:

$$\tilde{n} = \frac{\hat{n} + \vec{g}}{\|\hat{n} + \vec{g}\|} \quad (10)$$

Nachdem neue Knoten zum Baum hinzugefügt wurden, werden alle Anziehungspunkte entfernt, die sich innerhalb einer *Löschdistanz* d_c zu einem Knoten befinden.

Abbildung 18 veranschaulicht den Space Colonization Algorithmus anhand eines Beispiels. Hier sind bereits sechs Knoten vorhanden und vier Anziehungspunkte befinden sich in der Umgebung (a). Zuerst wird jeder Anziehungspunkt unter Berücksichtigung der Einflussdistanz d_i mit dem Knoten verbunden, der ihm am nächsten ist (b). Darauf aufbauend können alle Vektoren zu den Anziehungspunkten gebildet (c) und die Richtung \hat{n} bestimmt werden (d). Nun werden zwei neue Knoten angefügt, einer führt die Hauptachse weiter und einer beginnt einen seitlichen Ast (e). Im nächsten Schritt wird geprüft, ob Anziehungspunkte entfernt werden müssen. Die blauen Kreise in (f) repräsentieren das Umfeld der Anziehungspunkte innerhalb der Löschdistanz d_c , wobei die Kreise mit dickeren Linien bedeuten, dass Knoten im Umfeld vorhanden sind. Im Umkreis der beiden linken Anziehungspunkte befinden sich Knoten (bzw. deren Mitte), also werden diese Anziehungspunkte entfernt (g). Die nächste Iteration beginnt (h).

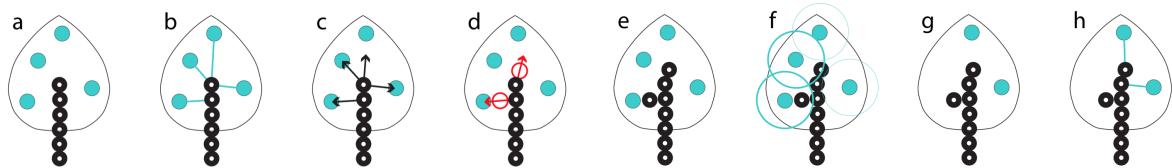


Abbildung 18: Der Space Colonization Algorithmus

Mit Hilfe des eben beschriebenen Algorithmus kann bei überschaubarer Anzahl von Parametern eine Vielzahl von Bäumen und Sträuchern erzeugt werden. Wenige Anziehungspunkte und größere Werte für die Löschdistanz führen zu einer niedrigeren Verzweigungsdichte (siehe **Abbildung 19**).



Abbildung 19:
Auswirkung der Anzahl von Anziehungspunkten und Größe der Löschdistanz

Beim Vergrößern der Einflussdistanz treten zwei Effekte auf. Zum einen werden die Äste immer gerader (weniger kurvig) und außerdem ergibt sich eine zunehmend exkurrente Form, d.h. ein klarer Hauptstamm wird erkennbar.² Folgende Abbildung veranschaulicht diese Effekte:

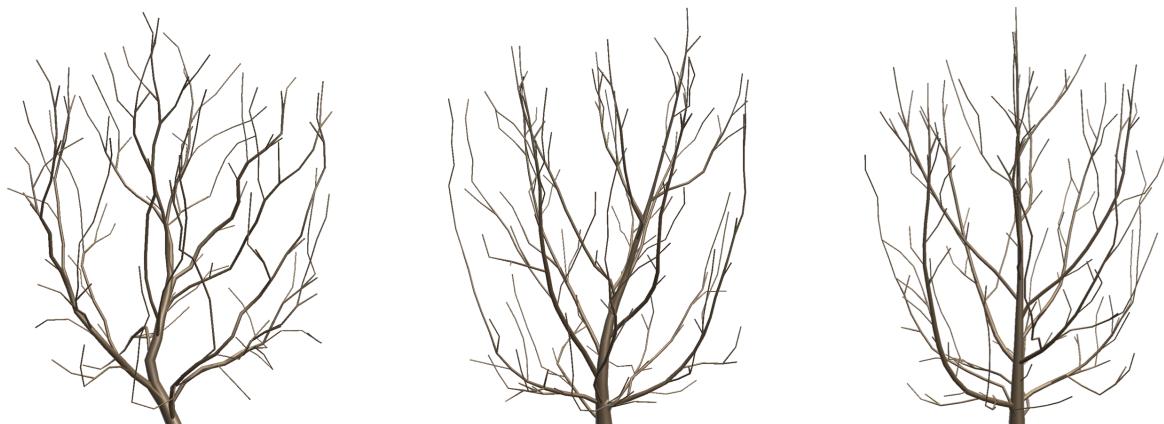


Abbildung 20: Auswirkung der Einflussdistanz (von links nach rechts zunehmend)

Wird die Dichte der Anziehungspunkte am Rand der Punktwolke erhöht, entstehen realistischere Bäume, da die Astdichte so nach außen hin zunimmt. [Abbildung 21](#) zeigt einen Baum bei dem die Anziehungspunkte ausschließlich am Rand platziert wurden. [\[25, S. 1ff.\]](#)

²Im Paper von Runions et al. [\[25, S. 66\]](#) wird die exkurrente Form als eine emergente Eigenschaft des Algorithmus angesehen, die zunehmend auftritt, wenn die Höhe der Punktwolke größer ist als die Breite und Tiefe. Die Ausführungen im Paper berücksichtigen jedoch nicht, dass dies nur dann der Fall ist, wenn die Einflussdistanz groß genug ist.

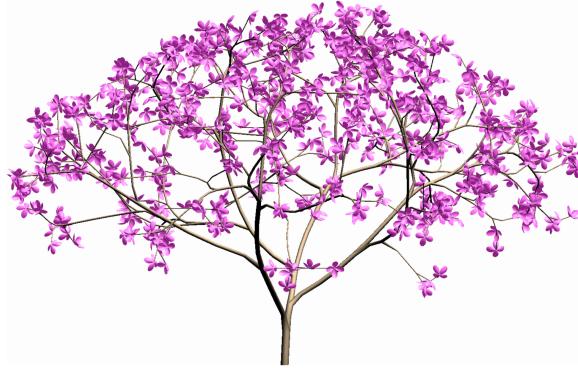


Abbildung 21: Platzierung der Anziehungspunkte am Rand der Punktwolke

Eigene Untersuchungen am Algorithmus haben ergeben, dass ein ähnlicher Effekt erzielt werden kann, indem der Löschdistanz zu Beginn ein großer Wert zugewiesen wird, der dann während des Wachstums verkleinert wird. Dieses Vorgehen birgt zwei Vorteile: Zum einen ist die Generierung der Punktwolke weniger komplex und zum anderen können Äste auch stärkere Verzweigungen aufweisen, wenn sie sich nicht am Rand der Punktwolke befinden. Entsprechende Ergebnisse sind in [Kapitel 6](#) abgebildet.

Herunterhängende Äste sind mit dem Space Colonization Algorithmus nur eingeschränkt modellierbar. Der Ansatz von Runions et al. dazu lautet, mittels Tropismenvektor das Wachstum nach oben zu verstärken, so dass im unteren Bereich der Punktwolke noch Anziehungspunkte übrig bleiben. So wachsen die Äste in späteren Iterationen wieder nach unten. [25, S. 69] Eigene Untersuchungen haben ergeben, dass diese Art der Modellierung eher unzuverlässig optisch ansprechende Ergebnisse liefert. Ein entscheidendes Problem hierbei ist, dass die oberen Äste durch die darunter liegenden keinen Platz mehr haben, um nach unten zu wachsen.

Bei den in diesem Abschnitt gezeigten Pflanzenstrukturen wurde bereits berücksichtigt, dass jüngere Zweige dünner als ältere sind. Beim verwendeten Verfahren zur Ermittlung der Zweigdurchmesser wird davon ausgegangen, dass alle Astenden gleich dick sind, also einen initialen Radius r_0 besitzen. Hat ein Knoten zwei Äste mit den Radien r_1 und r_2 , so berechnet sich sein Radius wie folgt: [34, nach [25, S. 65]]

$$r = \sqrt[n]{r_1^n + r_2^n} \quad (11)$$

n ist ein einstellbarerer Parameter der typischerweise Werte zwischen 2 und 3 annimmt. [35, nach [25, S. 65]] Für k Unteräste gilt:

$$r = \sqrt[n]{r_1^n + r_2^n + \dots + r_k^n} \quad (12)$$

3.4.5 Herunterhänge Äste

Zur Modellierung herunterhängender Äste wurde ein eigenes Verfahren entwickelt. Dieses arbeitet mit einer bereits fertig erzeugten Verzweigungsstruktur und biegt deren Äste in

3 Anforderungserhebung und -analyse

Richtung Boden. Die Biegung kann über zwei Parameter eingestellt werden. Der erste Parameter gibt an, ab welcher Ordnung (bzw. Verzweigungstiefe) die Äste nach unten gebogen werden sollen (ausgehend von einem Monochasium). Die Stärke der konkreten Biegungen wird über den zweiten Parameter festgelegt, im Folgenden als λ bezeichnet. Der Biegeprozess soll anhand eines Beispiels erklärt werden. In Abbildung 22 links ist eine “fertige” Verzweigungsstruktur abgebildet. Die schwarzen Punkte repräsentieren Knoten.

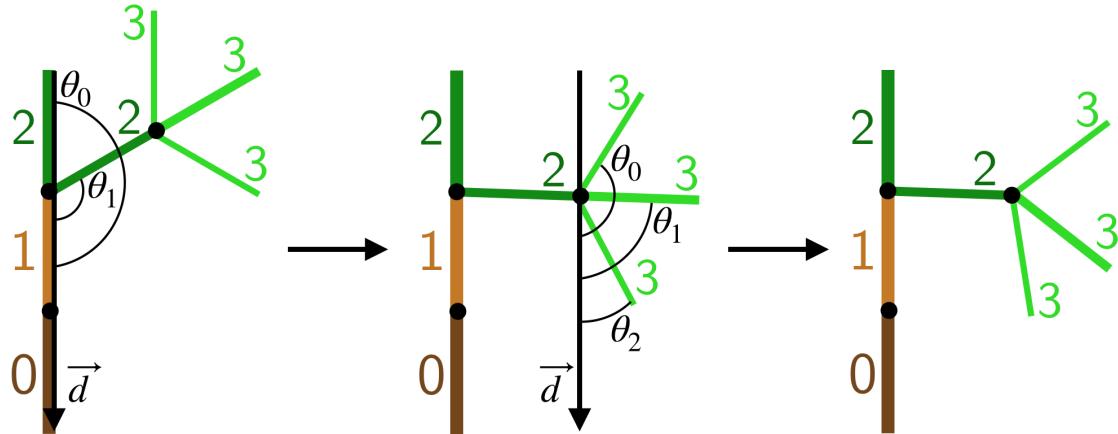


Abbildung 22: Methode zur Modellierung herunterhängender Äste

Die Internodien sollen ab einer Ordnung von 2, und mit einer Intensität von $\lambda = 0.8$ nach unten gebogen werden. Im ersten Schritt werden demzufolge alle Äste mit Ordnung 2 (inklusive ihrer untergeordneten Äste) um ihren übergeordneten Knoten um den Winkel ψ rotiert. ψ ist dabei ein Bruchteil des Winkels θ - dieser steht für den Winkel zwischen Ast und dem Vektor nach unten ($\vec{d} = (0, -1, 0)$) (siehe Abbildung 22). Es gilt:

$$\psi = \lambda \cdot \left(1 - \frac{\theta}{180}\right) \cdot \theta \quad (13)$$

Die zentrale Annahme der Gleichung lautet, dass der Einfluss der Gravitation zunimmt, je kleiner θ ist. In Abbildung 22 links beträgt $\theta_0 = 180^\circ$, daher erfolgt keine Rotation, denn:

$$\begin{aligned} \psi &= 0.8 \cdot \left(1 - \frac{180}{180}\right) \cdot 180 \\ &= 0.8 \cdot 0 \cdot 180 \\ &= 0 \end{aligned} \quad (14)$$

Für den Seitenast mit $\theta_1 = 120^\circ$ ergibt sich:

$$\begin{aligned} \psi &= 0.8 \cdot \left(1 - \frac{120}{180}\right) \cdot 120 \\ &= 0.8 \cdot \frac{1}{3} \cdot 120 \\ &= 32 \end{aligned} \quad (15)$$

Die mittlere Darstellung in Abbildung 22 zeigt das Ergebnis des ersten Schrittes und die für den nächsten Schritt benötigten θ . Rechts ist die vollständig modifizierte Pflanze abgebildet.

3 Anforderungserhebung und -analyse

Die erzielbaren Verzweigungsstrukturen sehen realistisch aus, Beispiele sind in [Kapitel 6](#) zu sehen.

3.5 Modellierung von Blättern

Für die Modellierung von Blättern gibt es verschiedene Möglichkeiten. Die realistischste Variante wäre, für jedes Blatt ein präzise geformtes Mesh zu verwenden und davon unzählige an der erzeugten Pflanze anzubringen. Für den geplanten Verwendungszweck ist dieses Verfahren jedoch nicht tragbar, da so, insbesondere bei etwas größeren Strukturen, extrem viele Vertices und Dreiecke für die Blätter benötigt werden.

In SpeedTree können für die Modellierung von kleinen Zweigen und Blättern sogenannte Cluster verwendet werden. Cluster sind bereits fertige Astenden mit Blättern, die in Form von Texturlappen³ an die Spitzen der generierten Äste gehangen werden. [36] Stellen in der Textur, an denen sich kein Zweig oder Blatt befindet, werden selbstverständlich transparent dargestellt. Für die Texturlappen werden im einfachsten Fall 2 bis 4 Dreiecke benötigt, dennoch sieht das Ergebnis realistisch aus.

Eine weitere Methode die in SpeedTree zur Blattmodellierung genutzt wird, ist die Verwendung von Cross-Foil-Meshes. Solch ein Mesh besteht aus zwei senkrecht aufeinander stehenden Quadraten und wird mit einer Menge von Blättern texturiert. [17] Aus etwas Entfernung sieht das Ergebnis immer noch realistisch aus obwohl nur wenige Vertices und Dreiecke benötigt werden.



Abbildung 23: SpeedTree Cluster

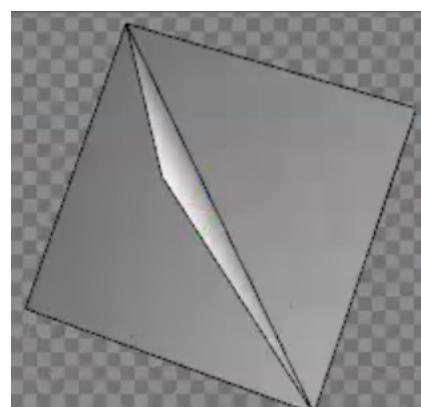


Abbildung 24:
SpeedTree Cross-Foil-Mesh ohne Textur

³Mit Texturlappen sind hier texturierte rechteckige Meshes gemeint, die je Seite aus zwei Dreiecken bestehen.

3.6 Kamerasteuerung

FAO10 fordert eine Betrachtung der dargestellten Pflanzenstruktur von allen Seiten, das heißt die Kamera muss in der Szene entsprechend bewegt und ausgerichtet werden. Zur Bestimmung der Kameraposition eignet sich die Verwendung von Kugelkoordinaten. Das Zentrum der Kugel bestimmt den Punkt, auf den die Kamera ihren Blick richtet, und kann beispielsweise als das Zentrum der Pflanze definiert werden. Der Nutzer kann dann bestimmen, aus welcher Richtung und Entfernung geschaut wird. Die Richtung kann mittels zweier Winkel θ, ϕ , und die Entfernung über den Radius r der Kugel bestimmt werden. **Abbildung 25** veranschaulicht die drei Parameter und zeigt außerdem wie die Position P der Kamera berechnet werden kann.

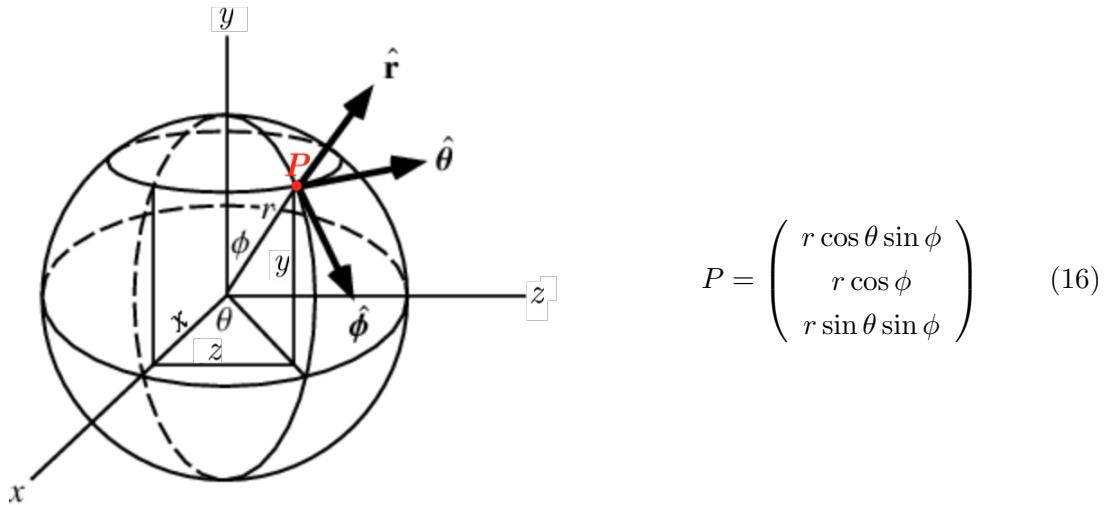


Abbildung 25: Berechnung von Kugelkoordinaten

3.7 Entwicklungsumgebungen

Bei der Entwicklung von 3D-Anwendungen kann auf zahlreiche vorhandene Bibliotheken und Tools zurückgegriffen werden. Da die algorithmische und logische Komponente dieser Arbeit für sich stehend schon relativ komplex ist, muss bei der Wahl der Entwicklungsumgebung darauf geachtet werden, dass die übrigen Anforderungen in vergleichsweise kurzer Zeit umgesetzt werden können. Die angestrebte Benutzeroberfläche - bestehend aus Elementen mit denen interagiert werden kann und der grafischen Darstellung des derzeitigen Ergebnis - sollte also nach Möglichkeit einen vergleichsweise geringen Anteil des Entwicklungsaufwandes ausmachen.

OpenGL (Open Graphics Library) ist die am weitesten verbreitetste Schnittstelle zwischen Programmierer und Grafikkarte, mit deren Hilfe zwei- und dreidimensionale Objekte gerendert werden können [38]. Die Programmierung findet bei OpenGL jedoch auf hardwarenaher Ebene statt, so dass relativ viel Code nötig ist, um sichtbare Ergebnisse zu erzielen. Aus diesem Grund wurde Unity als Grafik-Engine gewählt.

Eine Benutzeroberfläche kann mit Hilfe von Unity leicht erstellt werden und auch das Rendern von erzeugten Geometriedaten ist mit Hilfe der **Mesh**-Klasse unter Verwendung von wenigen Code-Zeilen möglich. Standard-Shader-Implementierungen sind ebenfalls vorhanden und können bei Bedarf modifiziert werden. Die umfassende Scripting-API von Unity bietet zahlreiche

3 Anforderungserhebung und -analyse

Funktionalitäten, um mit Objekten im dreidimensionalen Raum zu arbeiten und ist außerdem gut dokumentiert. Zudem unterstützt Unity verschiedene Schnittstellen zur Hardware (darunter auch OpenGL) und wählt zwischen diesen je nach Verfügbarkeit auf dem jeweiligen Betriebssystem [39]. Dadurch können einmalig in Unity entwickelte Anwendungen problemlos für beispielsweise Linux, macOS, Windows und auch iOS sowie Android zur Verfügung gestellt werden und Anforderung **NFAO2** ist prinzipiell bereits umgesetzt.

Ein Nachteil von Unity ist, dass die maximale Anzahl der Vertices und Dreiecke je **Mesh**-Objekt auf 65536 begrenzt ist. Da Anforderung **FAO15** ohnehin eine Limitation diesbezüglich verlangt, ist es jedoch kein Problem. Falls dennoch Objekte mit mehr Vertices und Dreiecken benötigt werden sollten, ist theoretisch immer noch eine Aufteilung in mehrere **Mesh**-Objekte möglich.

Anwendungen die mit Unity entwickelt werden, bestehen aus drei grundlegenden Bestandteilen. An oberster Stelle stehen *Szenen*. Diese können Objekte vom Typ *GameObject* enthalten - das sind beispielsweise ein Licht, ein 3D-Würfel oder auch einfach nur leere Objekte. [40] *GameObjects* wiederum können Objekte vom Typ *Component* zugeordnet werden. [41] Ein Beispiel dafür ist die Component *Transform*. Sie wird genutzt, um die Position, Rotation und Skalierung eines *GameObjects* zu speichern und zu verändern. [42] Das Verhalten von *GameObjects* kann programmiert werden, indem ein C#-Skript als Component hinzufügt wird. So ein Skript erbt dabei immer von der Klasse *MonoBehaviour* [43] und enthält typischerweise die Methoden **Start()** und **Update()**. Die **Start()**-Methode wird genau einmal zu Beginn der Lebenszeit des *GameObjects* aufgerufen. [44] Die **Update()**-Methode wird danach automatisch einmal pro Frame aufgerufen. [45]

4 Entwurf

Dieses Kapitel beschäftigt sich mit den zentralen Komponenten der zu entwickelnden Software und deren Zusammenspiel. Dazu wird zunächst eine geeignete Benutzeroberfläche und anschließend die Softwarearchitektur entworfen. Zuvor jedoch werden geeignete Modellierungsansätze ausgewählt, um den Anforderungen gerecht zu werden.

4.1 Auswahl der Modellierungsansätze

Im letzten Kapitel wurden einige Ansätze zur Pflanzenmodellierung vorgestellt. Es hat sich gezeigt, dass mit prozeduralen Ansätzen zwar präzise modelliert werden kann, für eine größere Vielfalt sind jedoch viele Parameter nötig. L-Systeme sind insbesondere in ihren weiterentwickelten Formen mächtig, jedoch ist die Herstellung auf Grund des sehr abstrakten Konzeptes eher wenig intuitiv. Die regelbasierte Objekterzeugung macht den Modellierprozess für Nutzer um einiges einfacher als die zuvor präsentierten Ansätze und erlaubt dennoch eine präzise Modellierung. Trotz allem muss eine Vielzahl von Parametern kontrolliert werden.

Der Space Colonization Algorithmus von Runions et al. verwendet vergleichsweise wenige Parameter, um realistisch aussehende Verzweigungsstrukturen zu erzeugen. Neben der geringen Anzahl anpassbarer Parameter besitzt der Algorithmus einige weitere nützliche Eigenschaften. Überlappungen von Ästen werden implizit vermieden, da dort, wo sich schon Äste befinden, auch keine Anziehungspunkte mehr existieren. Der Modellieraspekt, der bei den bisherigen Softwarelösungen eine zentrale Rolle einnimmt, entfällt mit der Nutzung des Space Colonization Algorithmus fast vollständig. Dies wird unter anderem durch die Verwendung einer Punktwolke ermöglicht. So muss lediglich definiert werden, wo im Raum die Pflanze wachsen soll. Die geplante Software muss dazu Interaktionsmöglichkeiten zur Modellierung der Punktwolke bereitstellen, im Folgenden soll sie ausgehend von einer Kugel modelliert werden. Indem die Kugel in die Höhe, Tiefe und Breite gezogen sowie oben und unten abgeschnitten werden kann, ist eine Vielzahl von Baumkronen-Formen erreichbar.

Ein weiterer Vorteil des Algorithmus ist es, dass die ohnehin wenigen Parameter sich direkt auf bestimmte Merkmale der erzeugten Pflanzenstruktur auswirken. So soll die Löschdistanz d_c im weiteren Verlauf zur Regulierung der Verzweigungsdichte dienen. Für realistischere Darstellungen sollte durch den Nutzer ein Start- und Endwert definiert werden können. Während des Wachstums wird dann zwischen den beiden Werten interpoliert. Der größtmögliche Wert, den der Nutzer einstellen kann, muss dabei sehr nahe an der Einflussdistanz d_i liegen, um eine geringe Astdichte erzielbar zu machen.

Die Einflussdistanz d_i bietet eine einfache Kontrolle darüber, ob die Äste eher gekrümmt oder gerade wachsen. Die entsprechende Anforderung ist zwar fakultativ, da sie jedoch einfach umzusetzen ist, bietet sich eine Anpassbarkeit der Einflussdistanz an. Dies trägt außerdem dazu bei, eine breitere Masse an Ergebnisse erzielen zu können. Der Nutzer soll hierbei innerhalb eines fest definierten Bereiches den Wert für die Löschdistanz festlegen können. Abhängig von der Einflussdistanz muss dann für eine geringe Astdichte auch das mögliche Intervall für die Löschdistanz aktualisiert werden. Aus Experimenten mit den Parameterkombinationen geht hervor, dass die Dichte der Punktwolke für unterschiedliche Einflussdistanzen (und somit auch Löschdistanzen) ebenfalls angepasst werden muss. Wird dies nicht getan, verändert sich die Astdichte.

Theoretisch könnten über die Einflussdistanz auch exkurrente Strukturen konfiguriert werden, dafür sind jedoch sehr große Werte nötig. Dies hat den Nachteil, dass mögliche Optimierungsmaßnahmen für das Finden des nächstgelegenen Knotens zu einem Anziehungspunkt drastisch eingeschränkt werden. (Detailliertere Ausführungen hierzu folgen im nächsten Kapitel.) Aus diesem Grund soll der Hauptstamm in der Baumkrone prozedural erzeugt werden und die Generierung der Verzweigungsstruktur erfolgt dann mittels des Space Colonization Algorithmus. Der häufig bei Bäumen auffindbare initiale Stamm bis zum Beginn der Baumkrone soll ebenfalls prozedural erzeugt werden.

Tropismen können bei der Nutzung des Space Colonization Algorithmus durch einen Vektor simuliert werden. So ist ein Streben des Wachstums in Richtung der Lichtquelle simulierbar, was zum realistischen Aussehen der erzeugten Pflanzen beiträgt. Wie sich noch herausstellen wird, ist es von Vorteil, einen zweiten Vektor für eine Gewichtung der Tropismen zu verwenden. So kann der Einfluss der Tropismen auf einfache Art und Weise temporär gedämpft oder intensiviert werden.

Das Alter der erzeugten Pflanze lässt sich beim Space Colonization Algorithmus über die Anzahl der Iterationen regulieren. Zur Modellierung herunterhängender Äste soll die beschriebene eigens entwickelte Methode verwendet werden.

Ein Problem der einfachen Version des Space Colonization Algorithmus ist, dass Äste manchmal “plötzlich” rückwärts wachsen. Dies soll im Folgenden vermieden werden, indem die Anziehungspunkte, die das Wachstum eines Knotens beeinflussen, zusätzlich über einen Wahrnehmungswinkel eingegrenzt werden. Das Konzept stammt aus einem Verfahren von Palubicki et al. [7], das den Space Colonization Algorithmus ebenfalls benutzt. In Abbildung 26 ist der Wahrnehmungswinkel mit θ bezeichnet.

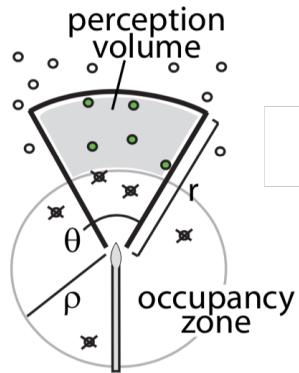


Abbildung 26: Wahrnehmungswinkel nach Palubicki et al.

Die Stammdicke soll mit dem selben Verfahren ermittelt werden, das auch Runions et al. [25, S. 26] verwenden.

Für die Blattmodellierung werden Cross-Foil-Meshes verwendet und motiviert durch ein Verfahren von Reeves und Blau [46], mit vielen farbigen Ellipsen, statt echt aussehenden Blättern texturiert. So bleibt die Anzahl der benötigten Vertices relativ niedrig, wobei die Ergebnisse trotzdem an echte Pflanzen erinnern.

Für etwas Flexibilität soll dennoch konfigurierbar sein, dass einfache Texturlappen statt Cross-Foil-Meshes bei der Blatterzeugung verwendet werden. Für beide Möglichkeiten ist es zudem sinnvoll, selbst erstellte Texturen in das Programm laden zu können.

4.2 Benutzeroberfläche

Die entworfene Benutzeroberfläche der geplanten Software besteht aus zwei Komponenten. Ein Bereich in der Mitte dient zur Darstellung des derzeitigen Ergebnis, und links und rechts davon befinden sich Elemente mit denen Parameter modifiziert werden können:

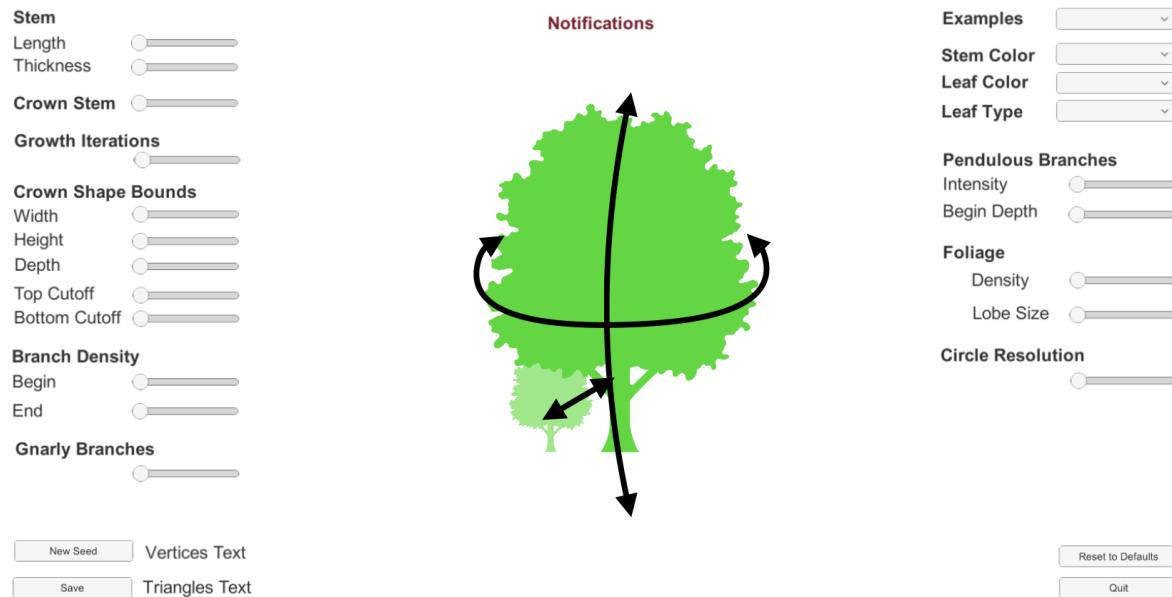


Abbildung 27: Benutzeroberfläche

Für Parameter mit diskreten Werten werden Dropdown-Elemente verwendet und für Parameter mit stetigen Werten Slider. Hinzu kommen Buttons für Funktionalitäten wie das Speichern des Ergebnis oder Generieren eines neuen Seeds. Zwei Textfelder zeigen außerdem die Anzahl der verwendeten Vertices und Dreiecke der angezeigten Pflanze. Mittels eines weiteren Textfeldes am oberen Rand des Fensters / Bildschirms können dem Nutzer Nachrichten angezeigt werden.

Mittels gehaltenem Linksklick und anschließender Mausbewegung kann die dargestellte Pflanze von allen Seiten betrachtet werden. Die Entfernung zur Pflanzenstruktur wird mittels Mausradscrollen eingestellt. Beides ist mit Hilfe der im letzten Abschnitt beschriebenen Kamerasteuerung umsetzbar und wird in Abbildung 27 durch die schwarzen Pfeile verdeutlicht. Die Position der Kamera kann außerdem samt Punkt, auf den sie ihren Blick richtet, mittels linker Umschalttaste nach unten, und mittels Leertaste nach oben verschoben werden. Dies erhöht die Flexibilität der Kamerasteuerung zusätzlich.

4.3 Softwarearchitektur

Anwendungen mit grafischer Benutzeroberfläche folgen typischerweise dem MVC-Entwurfsmuster (**Model View Controller**). So ist die Darstellungsschicht (View) über eine Vermittlungsschicht (Controller) von der dahinter liegenden Logik und Datenhaltung (Model) entkoppelt und Änderungen können an einzelnen Schichten vorgenommen werden, ohne die anderen zu überarbeiten.

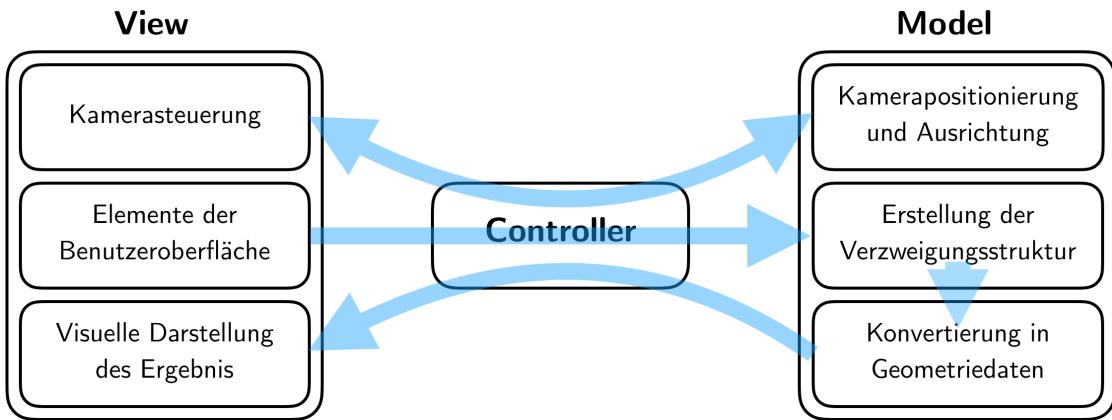


Abbildung 28: Grundlegende Komponenten der geplanten Software

Aus softwarearchitektonischer Perspektive besteht die Darstellungsschicht der geplanten Anwendung aus drei grundlegenden Komponenten. Das sind Elemente mit denen der Nutzer das Ergebnis modellieren kann, die Darstellung des Ergebnis selbst und die Kamerasteuerung, mit deren Hilfe das Ergebnis von allen Seiten betrachtet werden kann. Die Model-Schicht sollte demzufolge ähnlich aufgebaut sein. Eine Komponente ist für die korrekte Positionierung und Ausrichtung der Kamera zuständig, eine für die Generierung der Verzweigungsstruktur auf Basis der vom Nutzer definierten Parameter, und eine für die Konvertierung der erzeugten Verzweigungsstruktur in darstellbare Geometriedaten. Die Kommunikation zwischen den Schichten erfolgt über die Controller-Schicht.

5 Implementierung

Dieses Kapitel beschäftigt sich mit der Implementierung, also der konkreten Umsetzung der ausgewählten Konzepte, um den Anforderungen gerecht zu werden. Die im letzten Kapitel entworfene Softwarearchitektur wird dazu in konkrete Klassen umgewandelt. Die wichtigsten Klassen und deren Zusammenspiel werden im Folgenden beschrieben.

5.1 Datenstruktur und Geometriedaten

Zur Speicherung der erzeugten Verzweigungsstruktur wird eine geeignete Datenstruktur benötigt. Diese verfügt hier, neben ihrer Funktion als Speicherort, über Methoden, die die gespeicherte Verzweigungsstruktur in Geometriedaten überführen können. Die zentrale Datenstruktur dafür ist die **Tree**-Klasse. Bei ihrer Instantiierung bekommt sie ein Objekt der Klasse **GeometryProperties** übergeben. In diesem sind alle Parameter enthalten, die für die Erzeugung und Darstellung von Geometriedaten wichtig sind. Zur Laufzeit gibt es immer nur ein **Tree**-Objekt. Darin wird neben den **GeometryProperties** auch eine Referenz auf den Wurzelknoten der Verzweigungsstruktur gespeichert - dieser wird durch ein **Node**-Objekt repräsentiert.

Node-Objekte bekommen bei ihrer Erstellung ebenfalls das **GeometryProperties**-Objekt übergeben. Jeder Knoten besitzt zudem eine Referenz zu seinem übergeordneten Knoten und speichert die Referenzen seiner untergeordneten Knoten in einer Liste. Neue Knoten können zu einem vorhandenen Knoten über die **Add()**-Methode hinzugefügt werden. Übergeben wird hierfür lediglich die Position des neuen Knotens. Der Superknoten erstellt den Subknoten dann, übergibt ihm eine Referenz auf sein **GeometryProperties**-Objekt, eine Referenz auf sich selbst (so dass der Subknoten seinen Superknoten kennt) und die übergebene Position der **Add()**-Methode. Anschließend berechnet der Superknoten seinen Radius und seine Ausrichtung neu. Die Ausrichtung wird in Form eines Normalen-Vektors gespeichert und entspricht den blauen Pfeilen in Abbildung 29. Je dicker ein untergeordneter Knoten ist, desto mehr zeigt der Normalen-Vektor in die entsprechende Richtung. Die Gewichtung erfolgt über den Radius der Subknoten.

5 Implementierung

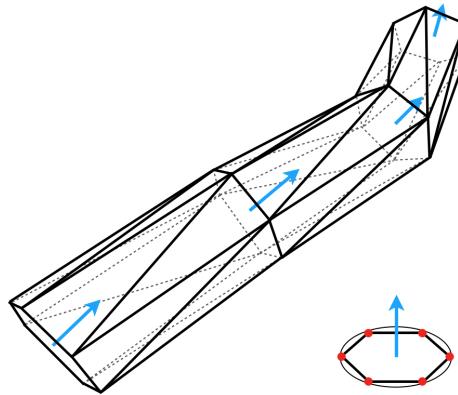


Abbildung 29: Generalisierter Zylinder mit eingezeichneten Normalen

Bei der Erstellung von Knoten erzeugen diese sich automatisch eine groß gewählte, feste Anzahl von Blättern und speichern die Referenzen in einer Liste. Die Blätter erhalten bei ihrer Instantiierung eine Position und das **GeometryProperties**-Objekt des Knotens. Die Position wird zufällig gewählt und liegt zwischen Knoten und seinem übergeordneten Knoten. Anschließend berechnen die Blätter sich selbst eine zufällige Ausrichtung und speichern diese in Form eines Quaternions. Die Anzahl der tatsächlich dargestellten Blätter ist im Parameter **DisplayedLeavesPerNode** in den **GrowthProperties** gespeichert und kann somit modifiziert werden, ohne dass ihre Ausrichtung sich verändert. (Dies wäre der Fall, wenn die Blätter für jede Änderung der Anzahl neu erstellt werden würden.)

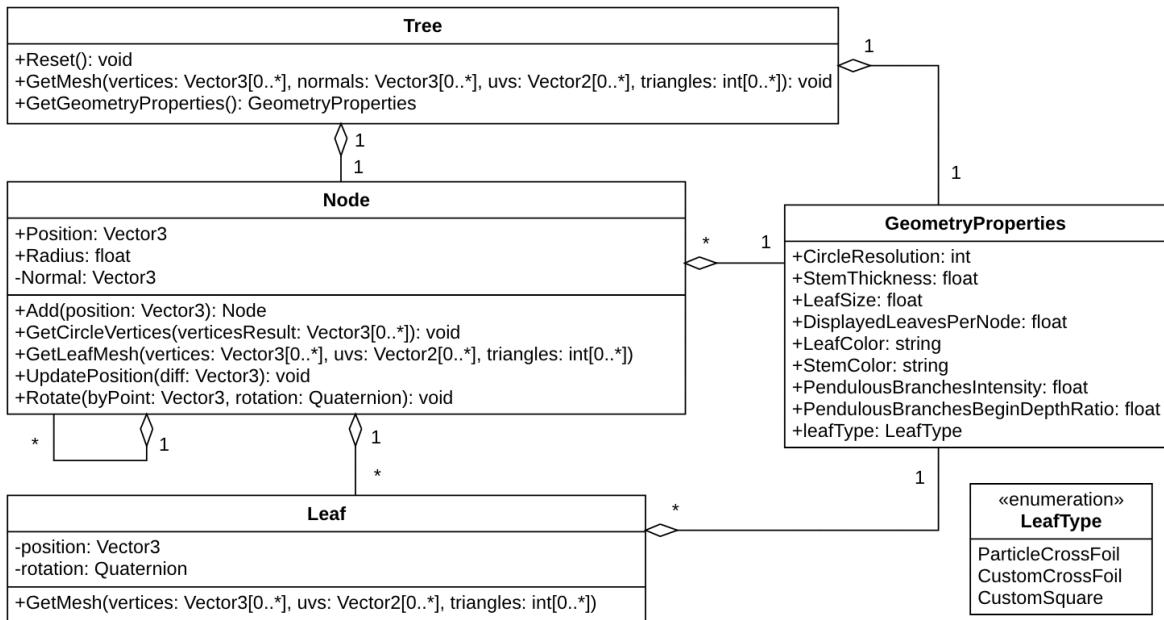


Abbildung 30: Datenstruktur zur Speicherung der Verzweigungsdaten und Erzeugung der Geometriedaten²

Wird auf dem **Tree**-Objekt die Methode `GetMesh()` aufgerufen, prüft es zunächst über den Wert des Attributs **PendulousBranchesIntensity** in den **GeometryProperties**, ob die Äste

²Die Idee der grundlegenden Struktur aus **Tree**, **Node** und **Leaf** stammt aus einem Video von “The Coding Train” [47]. Für den Zweck dieser Arbeit wurde diese allerdings noch stark modifiziert und ausgebaut.

5 Implementierung

nach unten gebogen werden müssen. Ist dies der Fall, wird der gesamte Baum geklont und die in [Unterabschnitt 3.4.5](#) beschriebene Methode auf den Klon angewandt. Die Geometriedaten werden anschließend für den Klon berechnet. Wenn keine Biegung stattfinden soll, also der Wert des Attributs **PendulousBranchesIntensity** 0 beträgt, werden die Geometriedaten auf Basis des Originalbaums berechnet. Die Verwendung des Klons ist im zuerst beschriebenen Fall zwingend nötig, da die Modifikation für realistische Ergebnisse immer nur ein einziges Mal auf einen fertigen Baum angewandt werden darf. Würde kein Klon verwendet werden, könnte der Space Colonization Algorithmus anschließend auch nicht ordnungsgemäß auf dem Baum weiterarbeiten.

Bei der Berechnung der Geometriedaten fordert das **Tree**-Objekt in einem rekursiven Prozess die Kreiskoordinaten der Knoten an, verbindet diese zu Dreiecken und berechnet zu den Kreiskoordinaten die UV-Koordinaten. Für realistischere Darstellungen, sollte während dieses Prozesses ein entscheidendes Detail beachtet werden. Hat ein Nachfolgerknoten an einer Verzweigung einen viel kleineren Radius als der Vorgängerknoten, ist es für optisch ansprechendere Ergebnisse sinnvoll, den Vorgängerknoten zu duplizieren. Das Duplikat wird dann ausschließlich in Richtung des Nachfolgerknotens mit kleinem Radius ausgerichtet und übernimmt auch seinen Radius. Verbunden werden dann das Duplikat und der Nachfolgerknoten. [Abbildung 31](#) zeigt den Unterschied, der insbesondere bei einzelnen Segmenten an dickeren Ästen sichtbar ist. Wie groß der Unterschied zwischen den Radien sein darf, ist in einem Parameter in den **GeometryProperties** fest definiert.

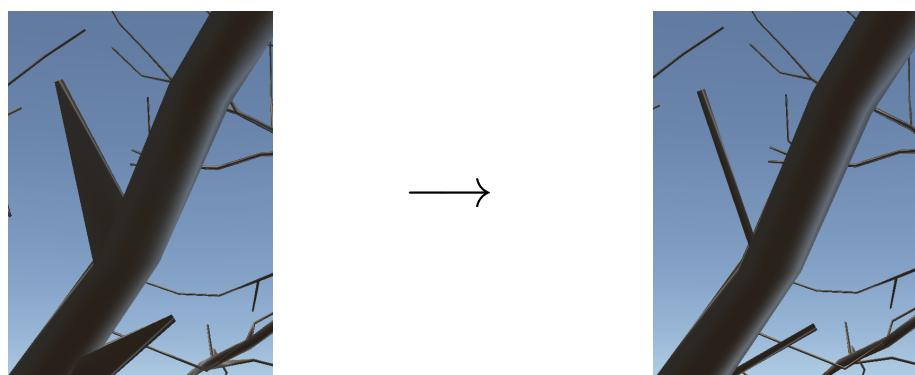


Abbildung 31: Verbesserte Darstellung unter Verwendung von modifizierten Duplikatknoten

Um die Geometriedaten der Blätter zu erhalten, wird auf jedem Knoten **GetLeafMesh()** aufgerufen. Ist der Radius des Knotens klein genug, gibt er für so viele Blätter die Geometriedaten zurück, wie es im Parameter **DisplayedLeavesPerNode** in den **GrowthProperties** definiert ist. Ist der Wert des Parameters keine ganze Zahl, wird der gebrochene Anteil - zusätzlich zum Ganzzahligen - als Wahrscheinlichkeit interpretiert, mit der ein Blatt dargestellt werden soll. Dies ermöglicht eine einfache Kontrolle über die Dichte der Blätter, ohne absolute Werte definieren zu müssen. Zur Berechnung und Speicherung der Geometriedaten verfügen die Blätter über eine **GetMesh()**-Methode. Die Größe der Blätter wird bei Abruf der Geometriedaten jedes mal erneut auf Basis der definierten **LeafSize** im **GeometryProperties**-Objekt ermittelt. Die **LeafSize** repräsentiert dabei den Mittelwert μ einer Normalverteilung mit fest definierter Standardabweichung $\sigma = 0.2\mu$, so dass die Blätter nicht alle gleich groß sind. Für

5 Implementierung

die Erzeugung von Zufallszahlen der Normalverteilung wurde die Implementierung von [48] genutzt. Abhängig vom Parameter `LeafType` gibt das Blatt ein Cross-Foil-Mesh oder einen quadratischen Texturlappen für seine Geometriedaten zurück.

Alle Methoden, die Geometriedaten erzeugen, fügen diese direkt zu zentralen `List`-Objekten³ hinzu, so dass die Ergebnisse nicht noch zusammengeführt werden müssen. Nachdem alle Vertices, UV-Koordinaten und Dreiecke ermittelt wurden, berechnet das `Tree`-Objekt die Normalen nach [49] und gibt alle vier Komponenten in Form von Listen an den Aufrufer zurück. Die Rückgabe erfolgt dabei über Call-by-Reference-Parameter.

5.2 Erzeugung der Verzweigungsstruktur

Wie in Kapitel 4 festgelegt, wird der Space Colonization Algorithmus für die Generierung der Verzweigungsstruktur verwendet und mit prozeduraler Stammerzeugung ergänzt. Dieser Abschnitt stellt die konkrete Implementierung vor.

5.2.1 Grundlegende Klassen

SpaceColonization

Der Kern des Algorithmus wurde in die Klasse `SpaceColonization` verpackt und enthält neben dem Algorithmus einige zusätzliche Methoden. Bei der Instantiierung eines `SpaceColonization`-Objekts wird ein Objekt der Klasse `GrowthProperties` und eine Implementierung des Interface `GrowerListener` übergeben.

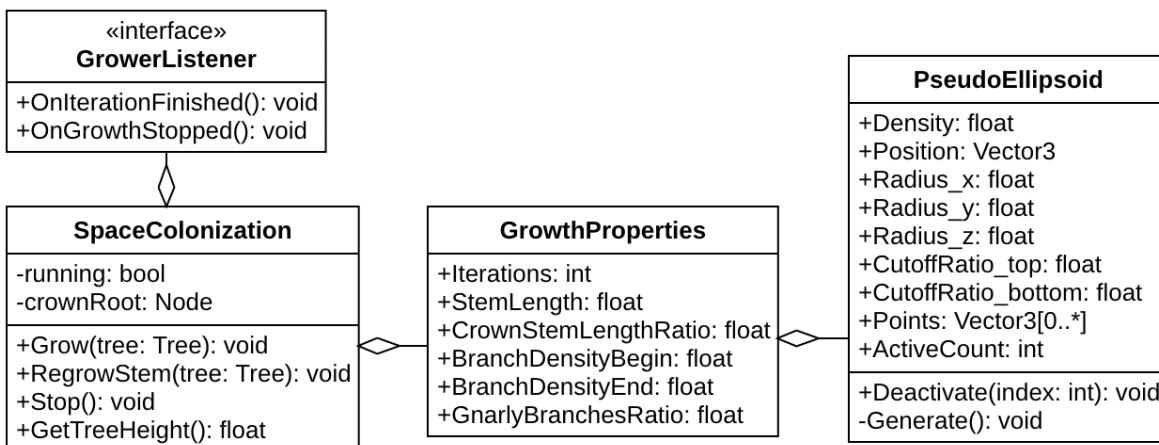


Abbildung 32: Essenz der relevanten Klassen für den Wachstumsvorgang

Die Methode `Grow(Tree tree)` der Klasse `SpaceColonization` lässt zuerst einen initialen Stamm (gegebenenfalls verlängert in die Baumkrone) und dann die Baumkrone auf dem übergebenen `Tree`-Objekt wachsen. Entscheidend ist hierbei, dass ein Aufruf der `Grow()`-Methode asynchron ist, d.h. es wird ein neuer Thread erzeugt, der dann die Berechnungen ausführt. So kann der Nutzer auch während der Berechnungen mit der Benutzeroberfläche interagieren.

³In C# sind Objekte der Klasse `List` intern als Array implementiert. [50]

5 Implementierung

Das Attribut `crownRoot` der `SpaceColonization`-Klasse speichert eine Referenz auf den Wurzelknoten der Baumkrone. So kann mittels `RegrowStem()` auch nur der Stamm neu generiert werden, was eine Veränderung der Stammlänge ermöglicht, ohne die gesamte Baumkrone neu zu berechnen. Nachdem der neue Stamm erzeugt wurde, werden die Unterknoten der alten `crownRoot` auf die neue `crownRoot` übertragen. In einem rekursiven Prozess erfolgt dann mittels der `Rotate()`-Methode der `Node`-Klasse eine Verschiebung aller Knoten der Baumkrone um die Positions differenz zwischen alter und neuer `crownRoot`.

Die Methode `Stop()` führt zu einem schnellstmöglichen Abbruch der derzeitigen Berechnungen und blockiert bis zum Ende des Threads. Dazu wird zwischen den Berechnungsschritten dauerhaft der Wert des Attributs `running` geprüft. Wenn es - durch die `Stop()`-Methode verursacht - den Wert `false` annimmt, werden die Berechnungen abgebrochen.

Die Methode `GetTreeHeight()` ist für die Kamerasteuerung wichtig und gibt die y-Koordinate des am höchsten platzierten `Node`-Objekts zurück. Dies ist der höchste Punkt, anhand dessen die Kamera ihre Blickrichtung ausrichten darf.

Der Space Colonization Algorithmus arbeitet in Iterationen. Dadurch bietet es sich an, den Fortschritt der Berechnungen nach jeder Iteration anzuzeigen. Dazu wird die Implementierung des Interface `GrowerListener` genutzt. Nach jeder Iteration wird auf dem `GrowerListener` die Methode `OnIterationFinished()` aufgerufen. Das verursacht dann jeweils eine Neuberechnung und Darstellung der Geometriedaten auf Basis der derzeit erzeugten Verzweigungsstruktur. Über die `OnGrowthStopped()`-Methode kann außerdem dem Listener mitgeteilt werden, dass das Wachstum frühzeitig beendet wurde. Wozu das wichtig ist, wird im weiteren Verlauf noch erklärt.

GrowthProperties

Das Objekt der Klasse `GrowthProperties` dient als zentraler Speicherort sämtlicher Parameter, die für die Erzeugung der Verzweigungsstruktur relevant sind. In [Abbildung 32](#) sind nur die Attribute abgebildet, die durch den Nutzer direkt modifiziert werden können. Die Namen der Attribute `Iterations` und `StemLength` sind selbsterklärend, das Attribut `CrownStemLengthRatio` gibt an, wie sehr der Hauptstamm in die Baumkrone verlängert werden soll. (Zur Erinnerung: Dies hat den Zweck, exkurrente Strukturen zu ermöglichen.) Ein Wert von 0 bedeutet dabei, dass keine Verlängerung stattfinden soll, und bei einem Wert von 1 wird der Hauptstamm prozedural bis zum oberen Ende der Punktwolke verlängert.

Wie bereits im letzten Kapitel erläutert, lässt sich die Astdichte über die Löschdistanz regulieren und sollte für realistische Darstellungen während des Wachstums verschiedene Werte annehmen können. Dazu nehmen die Parameter `BranchDensityBegin` und `BranchDensityEnd` Werte von 0 bis 1 an und modifizieren intern entsprechend die Anfangs- und End-Löschdistanz. Die Löschdistanz kann dabei nur Werte in einem festen, sinnvoll definierten Intervall annehmen. Auf Basis eines Ausschnittes der Sigmoid-Funktion wird dann während des Wachstums zwischen den Anfangs- und Endwerten der Löschdistanz interpoliert. [Abbildung 33](#) zeigt die Sigmoid-Funktion.

5 Implementierung

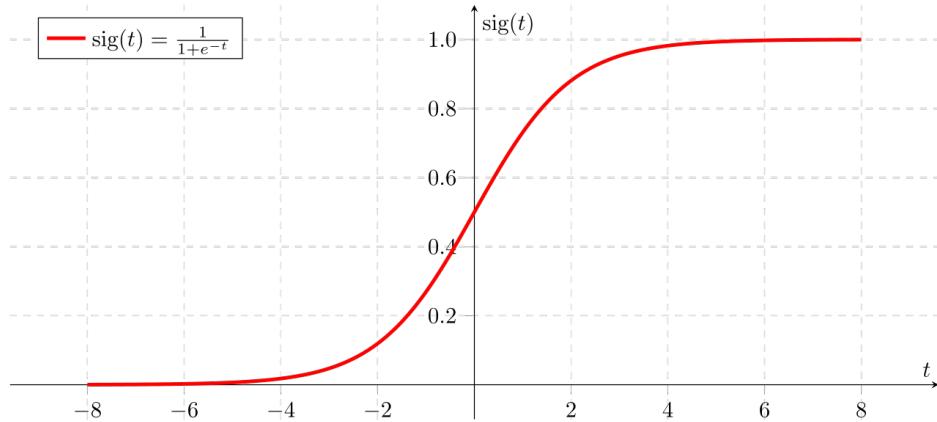


Abbildung 33: Sigmoid-Funktion

Ein Beispiel soll die Interpolation veranschaulichen. Der gewählte Ausschnitt der Sigmoid-Funktion beginnt bei $sig_{min} = -8$ und endet bei $sig_{max} = 8$. Die Anfangs- und Endwerte der Löschdistanz werden definiert mit $d_{c_{begin}} = 2$ und der $d_{c_{end}} = 1.2$. (Zur Erinnerung: Eine größere Löschdistanz bedeutet eine geringere Astdichte, im Beispiel nimmt die Astdichte also während der Zeit zu.) Es sollen $n = 30$ Iterationen durchgeführt werden. Die Löschdistanz ist somit eine Funktion die von der Iteration i des Algorithmus abhängt:

$$d_c(i) = d_{c_{begin}} - sig(sig_{min} + \frac{sig_{max} - sig_{min}}{n - 1} \cdot i) \cdot (d_{c_{begin}} - d_{c_{end}}) \quad (17)$$

In Gleichung 17 wird immer ein Teil der Differenz zwischen $d_{c_{begin}}$ und $d_{c_{end}}$ von der Löschdistanz am Anfang ($d_{c_{begin}}$) abgezogen. Dieser Teil der Differenz hängt von der Iteration ab und wird somit während des Wachstums größer. Denn die Sigmoid-Funktion nimmt im Intervall $[-8, 8]$ Werte von fast 0 bis fast 1 an und die 30 Iterationen werden in dieses Intervall abgebildet. Für $i = 0$ beträgt die Löschdistanz dann:

$$\begin{aligned} d_c(0) &= 2 - sig(-8 + \frac{8 - -8}{30 - 1} \cdot 0) \cdot (2 - 1.2) \\ &\approx 2 - sig(-8) \cdot 0.8 \\ &\approx 2 - 0 \cdot 0.8 \\ &\approx 2 \end{aligned} \quad (18)$$

Für $i = 14$:

$$\begin{aligned} d_c(14) &= 2 - sig(-8 + \frac{8 - -8}{30 - 1} \cdot 14) \cdot (2 - 1.2) \\ &= 2 - sig(-0.276) \cdot 0.8 \\ &\approx 2 - 0.431 \cdot 0.8 \\ &\approx 1.655 \end{aligned} \quad (19)$$

1.655 entspricht ungefähr der Mitte zwischen 1.2 und 2. Und bei $i = 29$ ergibt sich für die Löschdistanz:

$$\begin{aligned}
 d_c(29) &= 2 - \text{sig}(-8 + \frac{8 - -8}{30 - 1} \cdot 29) \cdot (2 - 1.2) \\
 &= 2 - \text{sig}(8) \cdot 0.8 \\
 &\approx 2 - 1 \cdot 0.8 \\
 &\approx 1.2
 \end{aligned} \tag{20}$$

Die Sigmoid-Funktion ermöglicht so einen fließenden Übergang zwischen dem Anfangs- und Endwert der Löschdistanz.

Der Parameter **GnarlyBranchesRatio** nimmt ebenfalls Werte von 0 bis 1 an und erlaubt eine Konfiguration für den Grad der Astkrümmung. Abhängig davon werden die Einflussdistanz, die Dichte der Punktwolke und das mögliche Intervall für die Löschdistanz gesetzt.

PseudoEllipsoid

Die Klasse **PseudoEllipsoid** stellt Mittel zur Erzeugung der benötigten Punktwolke bereit und speichert diese in einem Array von Objekten der Klasse **Vector3**. Aus Performance-Gründen werden die Anziehungspunkte während des Wachstums nicht aus dem Array gelöscht, sondern es wird ein zweites Array mit **boolean**-Werten genutzt, um zu speichern, welche Anziehungspunkte nicht mehr verwendet werden können. Damit die Punktwolke verformt werden kann, ist es sinnvoll, bei der Erstellung eine Punktdichte zu definieren - bei einer festen Anzahl hätte jede Verformung eine Veränderung der Astdichte zur Folge. Das Attribut **Position** dient zur Positionierung der Baumkrone und über die **Radius**- und **CutoffRatio**-Attribute wird die Form des **PseudoEllipsoids** definiert. Dabei nehmen die **Radius**-Attribute absolute, und die **CutoffRatio**-Attribute relative Werte an. Letztere sagen aus, welcher Anteil der Gesamthöhe der Kugel abgeschnitten werden soll. Die Erzeugung einer elliptisch geformten Punktwolke läuft wie folgt ab:

Zuerst wird das Volumen V_{base} einer Kugelschicht auf Basis der **CutoffRatio**-Parameter errechnet, wobei der Radius r der Kugel 1 beträgt.

Über die definierten Radien wird dann eine Transformationsmatrix M erstellt, die Punkte im Raum entsprechend skaliert. Anschließend wird das Volumen V_{target} des angestrebten Ellipsoids aus V_{base} und der Determinante von M berechnet (Formel nach [52]).

$$V_{target} = \det(M) \cdot V_{base} \tag{21}$$

Aus diesem Volumen V_{target} und einer sinnvollen fest definierten Punktdichte ρ kann dann berechnet werden, wie viele Punkte generiert werden müssen.

$$n = \frac{V_{target}}{\rho} \tag{22}$$

Die errechnete Anzahl an Punkten wird schließlich innerhalb des Kugelschnittes mit Radius 1 erzeugt, über M skaliert und an die definierte Position P verschoben.

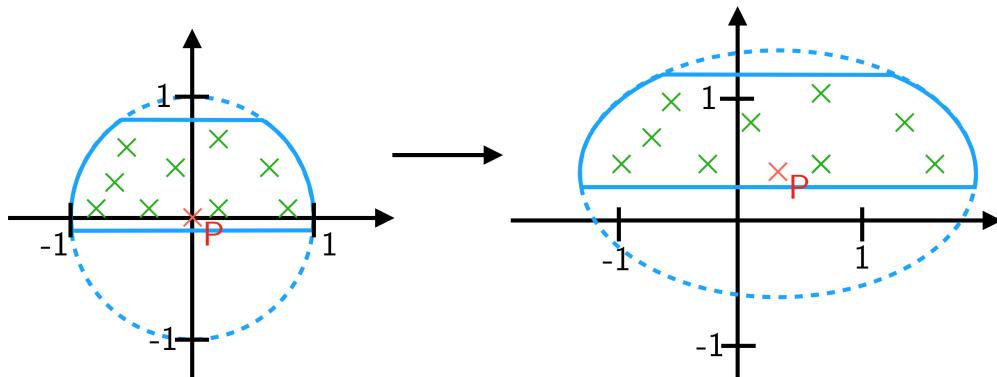


Abbildung 34: Konstruktion der Punktwolke im zweidimensionalen Raum

5.2.2 Optimierung der Distanzberechnungen

In jeder Iteration des Space Colonization Algorithmus muss zu jedem Anziehungspunkt der nächstgelegene Knoten gefunden werden. Schon Runions et al. [25] S. 68] stellten fest, dass dies viel Rechenzeit benötigt. Die wachsende Anzahl der Knoten während des Wachstumsprozesses führt insbesondere bei sehr großen Bäumen zu einer enormen Anzahl von Distanzberechnungen. Für die Verwendung des Algorithmus im interaktiven Kontext wurden deshalb Optimierungsschritte vorgenommen, deren Darstellung nun folgt. Die Effektivität der Optimierungsschritte wird am Ende des Abschnitts anhand eines Baumes verdeutlicht, der auf Basis von etwas über 25 000 Anziehungspunkten und 40 Iterationen entsteht (Abbildung 51 im Anhang).

Anziehungspunkte eingrenzen (Kürzel N)

Insbesondere bei großen Punktwolken und noch wenig Knoten wird im einfachsten Fall für alle Anziehungspunkte geschaut, ob ein Knoten in der Nähe ist. Die Anzahl der Anziehungspunkte, die hier betrachtet werden, lässt sich verringern, indem gespeichert wird, welcher Raum überhaupt von Knoten besiedelt ist. Dazu werden die Minimal- und Maximalwerte der x-, y- und z-Koordinaten bezogen auf alle Knoten gespeichert. Jedes Mal wenn ein Knoten erzeugt wird, erfolgt eine Aktualisierung der sechs Werte. Wenn dann für einen Anziehungspunkt geschaut werden soll, welcher Knoten ihm am nächsten ist, wird zunächst geprüft, ob er sich überhaupt in dem besiedelten Raum befindet. (Die Einflussdistanz wird in die Prüfung selbstverständlich einbezogen.)

Anziehungspunkte entfernen (Kürzel D)

Am Ende jeder Iteration des Space Colonization Algorithmus müssen alle Anziehungspunkte entfernt werden, die sich innerhalb der Löschdistanz d_c eines Knotens befinden. Der schnellste Weg dies zu tun ist es, die Löschung zum Beginn der nächsten Iteration durchzuführen - also während den Knoten die Anziehungspunkte in ihrer Umgebung zugeordnet werden. Denn dazu wird sowieso über alle Anziehungspunkte iteriert, um den jeweils nächstgelegenen Knoten zu finden. (Idee von Schneider [53])

Finden des nächstgelegenen Knotens

Die einfachste Implementierung für das Finden des nächstgelegenen Knotens zu einem Anziehungspunkt betrachtet in jeder Iteration, für jeden Anziehungspunkt, jeden Knoten, um den mit der kleinsten Distanz zum Anziehungspunkt zu finden. Die Distanzen werden im dreidimensionalen Raum über den euklidischen Abstand berechnet:

$$d(p_1, p_2) = \sqrt{(p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2} \quad (23)$$

Da der tatsächliche Distanzwert unwichtig ist, sondern nur welcher Abstand am geringsten ist, muss die Wurzel nicht berechnet werden, was den Rechenaufwand verringert: (Idee von Heywood [54])

$$d(p_1, p_2)^2 = (p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2 \quad (24)$$

Allerdings verändert sich so nur die benötigte Rechenzeit für einzelne Berechnungen, nicht deren Anzahl.

Ein weiterer Vorschlag von Heywood [54] lautet, die x-, y-, und z-Koordinaten der Knoten in drei sortierte Listen einzufügen, um anschließend mit binärer Suche, die in Frage kommenden Knoten einzugrenzen. Die benötigte Rechenzeit kann mit dieser Methode nochmals reduziert werden. Wie genau die Suche nach dem nächsten Knoten mittels der binären Suche abläuft, wird an dieser Stelle nicht näher erläutert, da ein effektiverer Weg zur Optimierung gefunden wurde.

Schneider [53] schlägt zur Eingrenzung der in Frage kommenden Knoten die Verwendung eines Voxelsgrids⁴ vor. Die Größe der Voxel wird durch die Einflussdistanz d_i des Space Colonization Algorithmus bestimmt. Neue Knoten werden in das Voxelgrid eingesortiert. Jedem Voxel ist also eine Liste aus Knoten zugeordnet, die sich in ihm befinden. Das Finden des nächstgelegenen Knoten zu einem Anziehungspunkt läuft dann wie folgt ab: Anhand der Koordinaten des Anziehungspunktes wird zuerst geschaut, in welchem Voxel er sich befindet. Als zweites werden zu diesem Voxel alle umliegenden Voxel ermittelt. Nun wird die Distanz zwischen Anziehungspunkt und allen Knoten berechnet, die sich im gleichen und den umliegenden Voxeln befinden. Die Anzahl der nötigen Distanzberechnungen wird auf diesem Weg stark reduziert. Der Algorithmus kann weiter optimiert werden, indem für jeden Anziehungspunkt gecached wird, in welchem Voxel er sich befindet und auch für jedes Voxel gespeichert wird, welches die umliegenden Voxel sind.

⁴Ein Voxel ist die dreidimensionale Variante eines Pixels und repräsentiert einen würfelförmigen Bereich im Raum. [55]

5 Implementierung

Abbildung 35 veranschaulicht die Suche nach dem nächstgelegenen Knoten im zweidimensionalen Raum. Die Kreuze repräsentieren bereits existierende Knoten. Der rote Kreis steht für einen Anziehungspunkt, zu dem der nächstgelegene Knoten gefunden werden soll. Alle blau markierten Knoten kommen in Frage, da sie sich in den umliegenden “Voxeln” befinden. Im Beispiel befindet sich der nächstgelegene Knoten im gleichen “Voxel” wie der Anziehungspunkt.

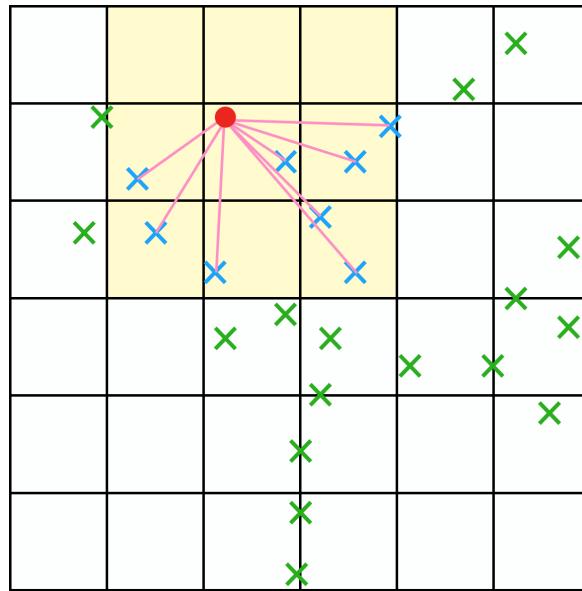


Abbildung 35: Beispiel für die Suche nach in Frage kommenden nächstgelegenen Knoten

Ergebnisse

Tabelle 1 zeigt die Effektivität der vorgenommenen Optimierungsschritte anhand der benötigten Zeit in Sekunden. Die einfachste Version des Algorithmus wird dabei mit “S” abgekürzt. Die binäre Suche und der Voxelgrid-Algorithmus arbeiten ebenfalls mit quadratischen Abständen.

Algorithmus	S	D	N	D & N
Euklidischer Abstand	49,3	45,1	27,7	22,2
Quadratischer Abstand	39,2	37,2	22,3	17,7
Binäre Suche	20,5	20,4	11,4	8,5
Voxelgrid	10,8	11	8,7	3,4

Tabelle 1: Benötigte Zeit zur Findung des nächstgelegenen Knoten

5.3 Kern

Den Kern der Implementierung bildet die **Core**-Klasse. Als Component des GameObjects “Core” erbt sie von der Klasse **MonoBehaviour** und verfügt über die Methoden **Start()** und **Update()**. In der **Start()**-Methode erstellt das instantiierte **Core**-Objekt ein **SpaceColonization**- und ein **Tree**-Objekt inklusive ihrer Konfiguration (**GrowthProperties** mit **PseudoEllipsoid**, und **GeometryProperties**). Die Referenzen auf die Konfigurationsobjekte speichert der **Core** außerdem, um später Anpassungen vornehmen zu können. Der **GrowerListener** für das

SpaceColonization-Objekt ist das **Core**-Objekt selbst. Nach der Initialisierung der genannten Objekte wird auf dem **SpaceColonization**-Objekt **Grow()** aufgerufen, wobei das erstellte **Tree**-Objekt übergeben wird. Der erste Aufruf der **Update()**-Methode des **Core**-Objekts initialisiert auf Basis der erstellten Konfigurationsklassen die Elemente der Benutzeroberfläche. Beispielsweise wird so zu Beginn der Wert des Sliders “Growth Iterations” auf 30 gesetzt, wenn dies im **GrowthProperties**-Objekt so definiert wurde.

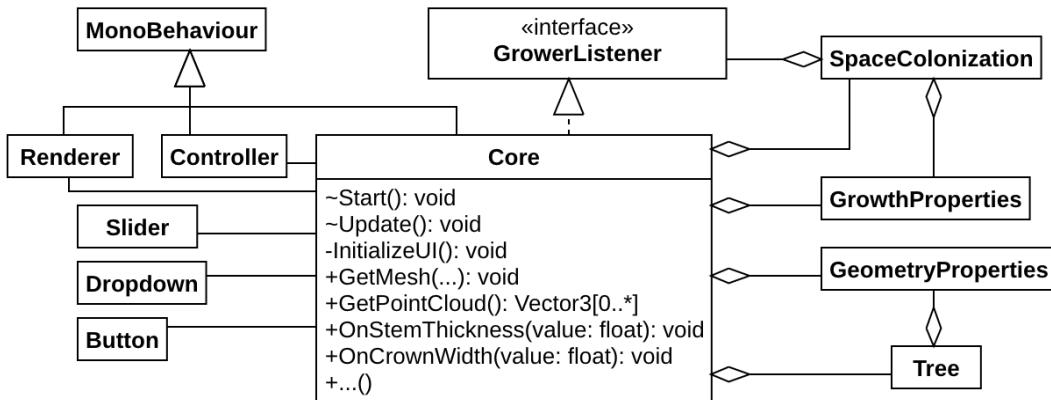


Abbildung 36: **Core**-Klasse im Zusammenhang⁵

Nachdem alle Elemente der Benutzeroberfläche initialisiert wurden, ist es die Aufgabe des **Core**s, sämtliche Ereignisse aus der Darstellungsschicht über den **Controller** entgegenzunehmen. Dementsprechend verändert er Werte im Model und verursacht eine Neuberechnung und Darstellung der Verzweigungsstruktur. Eine detailliertere Beschreibung dieser Abläufe folgt in Abschnitt 5.5.

5.4 Benutzeroberfläche und Controller

Die Benutzeroberfläche wurde über Unitys Benutzeroberfläche erstellt und besteht, wie bereits im letzten Kapitel gezeigt, aus Buttons, Slidern und Dropdowns. Textfelder dienen als Beschreibung der Funktionalitäten. Jedes dieser Elemente wird in Unity durch ein GameObject repräsentiert. Abbildung 37 gibt eine umfassendere Übersicht zu den erstellten GameObjects. C#-Skripte sind in blau dargestellt. Das GameObject “Plane” hat den Zweck, einen Boden zu repräsentieren. So schwebt die erstellte Pflanzenstruktur nicht in der Luft. Das “Directional Light” macht Licht, damit das Ergebnis gut sichtbar ist. Das GameObject “EventSystem” wird von Unity automatisch zur Szene hinzugefügt, sobald UI-Elemente existieren. Es kann beispielsweise genutzt werden, um herauszufinden, ob der Mauszeiger sich über einem UI-Element befindet.

⁵Tatsächlich besteht der Controller aus mehreren Klassen, diese werden im nächsten Abschnitt noch genauer erläutert. Gleichermaßen gilt für den Renderer.

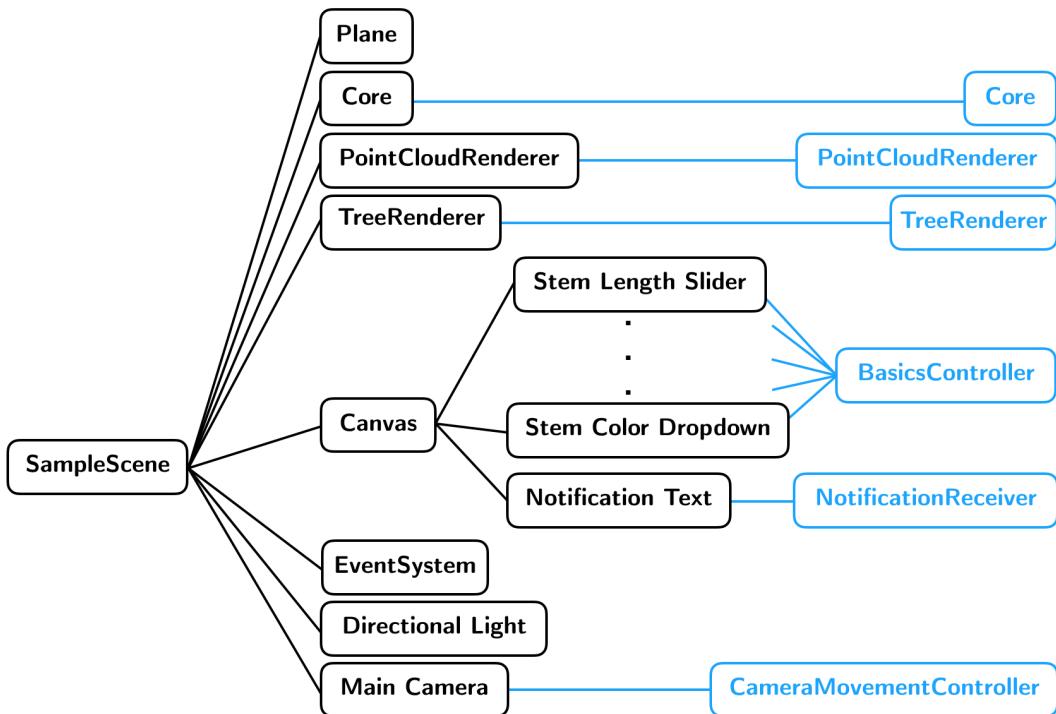


Abbildung 37: Szenengraph der entwickelten Software

Als Controller für die allermeisten UI-Elemente dient eine Klasse mit dem Namen **BasicsController**, die jedem UI-Element als Component zugewiesen wurde. Sie besteht aus Funktionen wie beispielsweise `OnValueChanged_Iterations()` oder `OnValueChanged_StemThickness()`. Die Funktionen lesen den Wert des jeweiligen Elements aus der Benutzeroberfläche und geben ihn über Unitys `GameObject.Find()`-Funktion an den **Core** weiter. Beim Klick von Buttons wird ebenfalls eine entsprechende Funktion im **Core** aufgerufen.

Der Controller für die Kamerasteuerung wurde in einer Klasse mit dem Namen **CameraMovementController** implementiert und dem GameObject "Main Camera" zugewiesen. Da das Modell der Kamerasteuerung nur wenige Zeilen Code lang ist, wurde es ebenfalls in den Controller gebaut. Als Component eines GameObjects verfügt die Klasse **CameraMovementController** über die Methoden `Start()` und `Update()`. In der `Start()`-Methode werden Standardwerte für die Ausrichtung gesetzt. Jeder Aufruf der `Update()`-Methode wartet dann auf Eingaben des Nutzers, die die Kamerasteuerung beeinflussen und verändert entsprechend die Werte von vier Parametern. Drei davon wurden bereits in [Kapitel 3](#) erläutert (über zwei Winkel θ und ϕ kann die Richtung, aus der geschaut wird und über einen Distanzwert r die Entfernung zur dargestellten Pflanzenstruktur eingestellt werden). Über einen vierten Parameter kann die Kamera samt Punkt, auf den sie schaut, nach unten und oben verschoben werden. Auf Basis der vier Parameter und eines Punktes L , auf den die Kamera ihren Blick richten soll, wird ihre Position und Ausrichtung kontinuierlich neu berechnet. Während der Modifikation der Punktwolke, wird sie für den Nutzer sichtbar dargestellt und die Kamera so platziert, dass möglichst die gesamte Punktwolke zu sehen ist. Dazu hängen L und r von der Stammlänge, und der Höhe, Breite und Tiefe der Punktwolke ab. Nach erfolgter Modifikation

5 Implementierung

kann der Nutzer die Kamera wieder frei bewegen.

Das GameObject “Notification Text” pollt über sein zugeordnetes Skript permanent etwaige Nachrichten beim **Core** und zeigt diese an. So wird beispielsweise, nach dem Klick des “Save”-Buttons, der Text “Done :)” dargestellt.

Zur Darstellung der Punktwolke wurde die Klasse **PointCloudRenderer** implementiert. Zum Rendern von Geometriedaten macht sie Gebrauch von Unitys **Mesh**-Klasse und ist dem GameObject “PointCloudRenderer” als Component zugeordnet. Analog dazu funktioniert das Rendern der derzeitigen Verzweigungsstruktur mittels der Klasse **TreeRenderer**.

Der **TreeRenderer** pollt beim **Core** die derzeit eingestellte Textur (genauer gesagt ihren Dateinamen) und wenn es Veränderungen im **Tree**-Objekt gab, werden die neuen Geometriedaten angefordert und dargestellt.

Der **PointCloudRenderer** arbeitet ähnlich, erhält jedoch vom **Core** nur eine Menge von Punkten. Diese wandelt er dann selbstständig in Dreiecke um - für jeden Punkt wird dabei ein Dreieck erzeugt. Die Dreiecke werden nur dargestellt, wenn gerade die Punktwolke modifiziert wird. Ob dies der Fall ist, findet der **PointCloudRenderer** ebenfalls über Polling bei **Core** heraus.

Zur Blattmodellierung werden die Cross-Foil-Meshes standardmäßig mit Partikeln texturiert. Die Texturen für Blätter und Stamm wurden mit OpenCV⁶ für eine Auswahl von Farben generiert. Abbildung 38 zeigt eine solche Textur exemplarisch.

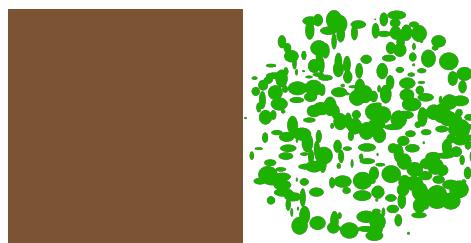


Abbildung 38: Textur bestehend aus Stammfarbe und Blattpartikeln

Es können jedoch auch Texturen geladen und die Verwendung von Texturlappen konfiguriert werden. Ermöglicht wird dies durch die Optionen “Custom Cross Foil” und “Custom Square” im “Leaf Type”-Dropdown. Die fertige Textur (inklusive Stammtextrur) muss dazu im gleichen Verzeichnis liegen, wie die Anwendung ausgeführt wird und den Dateinamen “custom_texture.png” besitzen. Der Nutzer wird darüber informiert, sobald eine der “Custom”-Optionen ausgewählt wurde.

⁶OpenCV ist eine quelloffene Computer Vision und Machine Learning Bibliothek [56] mit der unter anderem farbige Markierungen in Bildern gemacht werden können.

5.5 Abläufe

In diesem Abschnitt wird für die zentralen Features der entwickelten Software erklärt, welche Abläufe der Nutzer bei einer Interaktion auslöst.

5.5.1 Änderung von GrowthProperties

Das Sequenzdiagramm in Abbildung 39 zeigt die wichtigsten Schritte der ablaufenden Vorgänge, wenn der Nutzer die Höhe der Baumkrone mit einem Slider verändert (Schritt 1).

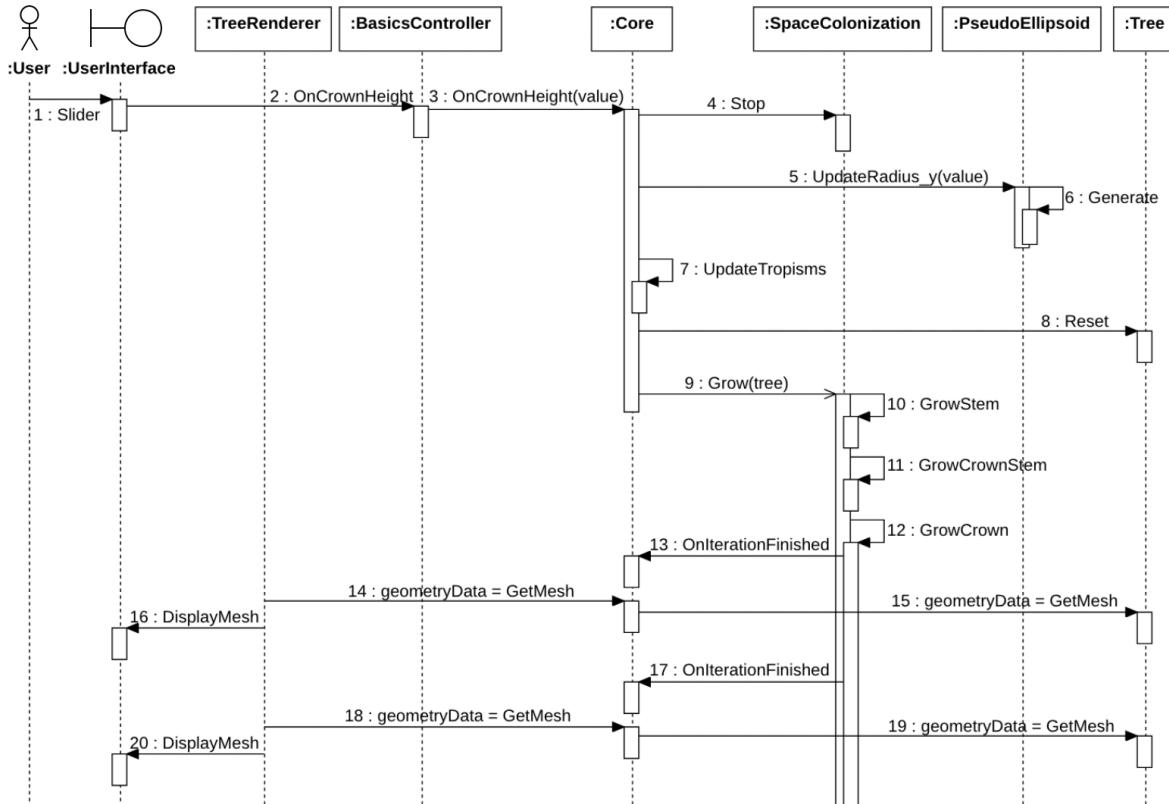


Abbildung 39: Ablauf beim Verändern des "CrownHeight"-Sliders

Dem Slider für die Höhe der Baumkrone ist ein Objekt der Klasse **BasicsController** zugeordnet (als Component), welches das Ereignis als erstes mitgeteilt bekommt (Schritt 2). Das **BasicsController**-Objekt liest den neuen Wert des Sliders und teilt ihn in Schritt 3 dem **Core** mit. Der **Core** stoppt daraufhin die derzeitigen Berechnungen des **SpaceColonization**-Objektes - falls überhaupt welche durchgeführt werden (Schritt 4). Danach wird der Radius des **PseudoEllipsoids** in y-Richtung verändert (Schritt 5), was eine Neuberechnung der Punktwolke nach sich zieht (Schritt 6: Methode **Generate()**).

Anschließend ruft der **Core** die Methode **UpdateTropisms()** auf (Schritt 7). Hier werden gegebenenfalls die Tropismen in positive y-Richtung über einen Gewichtungsvektor gedämpft. Dieser Schritt ist nötig, da sonst ein unerwünschter Effekt auftreten kann. Ist die Höhe der Punktwolke ungefähr genau so groß oder kleiner als ihre Breite und Tiefe, so sorgen die Tropismen in positive y-Richtung dafür, dass die Pflanzenstruktur schnell den oberen Rand der Punktwolke erreicht und alle danach wachsenden Äste eher nach unten wachsen. Analog

5 Implementierung

dazu muss die Dämpfung auch wieder aufgehoben werden, wenn ein Schwellenwert für das Verhältnis zwischen Höhe und Breite und Tiefe überschritten wurde.

Nun kann die Pflanzenstruktur auf Basis der neuen Punktwolke erzeugt werden. Dazu wird zunächst vom **Core** mittels der **Reset()**-Methode die derzeitige Struktur im **Tree** gelöscht (Schritt 8). Anschließend wird der leere **Tree** über die **Grow()**-Methode dem **SpaceColonization**-Objekt übergeben, welches zunächst den Stamm und anschließend die Baumkrone generiert (Schritte 9-12). Der Aufruf der **Grow()**-Methode ist hierbei asynchron, damit der Nutzer während der Berechnungen weiterhin mit dem Programm interagieren kann. Nach jeder Iteration des Space Colonization Algorithmus informiert das **SpaceColonization**-Objekt den **Core** darüber, dass neue darstellbare Daten vorliegen (Schritt 13). Der **Core** speichert diese Information in einer **boolean**-Variable mit dem Namen **recalculateMesh**. Denn der **TreeRenderer** polt mittels seiner **Update()**-Methode beim **Core** permanent die aktuell dargestellenden Geometriedaten (Schritte 14 bzw. 18). Falls diese neu berechnet werden müssen, also der Wert von **recalculateMesh true** ist, ermittelt der **Core** über die **GetMesh()**-Methode beim **Tree** die neuen Geometriedaten (Schritte 15 bzw. 19) und übergibt sie anschließend dem **TreeRenderer**.

Die Vorgänge sind hier für die Erläuterung vereinfacht dargestellt. Denn vor allem bei größeren Bäumen kann es vorkommen, dass die maximale Anzahl von 65536 Vertices je Renderer-GameObject überschritten wird. In diesem Fall erfolgt nach der Berechnung der Geometriedaten (Schritt 15 bzw. 19) eine Aufteilung in ausreichend viele Meshes. Anschließend werden vom **Core** so viele Renderer-GameObjects mit **TreeRenderer**-Component erzeugt, dass eine einwandfreie Darstellung möglich ist. Die **TreeRenderer** erfragen nach ihrer Erstellung beim **Core** eine ID, anhand dessen dieser später ermitteln kann, welcher **TreeRenderer** welchen Teil des Meshes erhält.

Mit Ausnahme des Sliders für die initiale Stammlänge gilt der eben beschriebene Ablauf für alle veränderbaren **GrowthProperties**. Bei einer Veränderung der Länge des initialen Stammes, wird nur dieser neu berechnet und die Baumkrone entsprechend verschoben. Dies geschieht, indem das Wurzel-**Node**-Objekt der Baumkrone eine Positions differenz in Form eines Vektors mitgeteilt bekommt und seine eigene Position auf Basis dessen anpasst. Danach benachrichtigt es alle seine Unterknoten, die den gleichen Prozess wiederholen.

5.5.2 Änderung von **GeometryProperties**

Im letzten Unterabschnitt wurde der ablaufende Vorgang beim Ändern eines Wertes in den **GrowthProperties** erläutert. Bei der Änderung von Werten im **GeometryProperties**-Objekt findet prinzipiell das gleiche statt, allerdings wird die Verzweigungsstruktur nicht neu berechnet (Schritte 9-12 in Abbildung 39).

Bei Änderungen von beispielsweise der Blattdichte, Blattgröße oder Kreisauflösung müssen nur die Geometriedaten neu berechnet werden, denn die drei Parameter fließen ausschließlich in diese Berechnungen ein.

Wenn die Stammdicke verändert wird, muss zuvor eine Neuberechnung der Radien aller Knoten im Baum durchgeführt werden. Dies passiert in zwei Schritten, denn die Berechnung

der Radien muss von oben nach unten erfolgen - also von den jüngsten Knoten zu den ältesten. (Zur Erinnerung: Die ausgewählte Formel zur Berechnung des Radius eines Knotens benötigt bereits den Radius aller Unterknoten.) In einem rekursiven Prozess werden also zunächst alle Spitzknoten darüber informiert, dass die Radien neuberechnet werden sollen. Diese verursachen dann wieder in einem rekursiven Prozess die tatsächliche Neuberechnung der Radien bei allen ihren übergeordneten Knoten.

5.5.3 Neuer Seed

Die Erzeugung der Punktwolke erfolgt auf Basis eines Zufallsgenerators, der mit einem Seed initialisiert wird. Der verwendete Seed wird in der Klasse **PseudoEllipsoid** gespeichert. Bei der Änderung eines Parameters wie der Astdichte wird dann für die Neuberechnung der Verzweigungsstruktur wieder der gleiche Seed und damit die gleiche Punktwolke verwendet. Dadurch entsteht der Eindruck, dass es sich um die gleiche Pflanzenstruktur handelt - nur in modifizierter Form. Ist nun eine neue Pflanzenstruktur auf Basis der definierten Parameter gewünscht, wird lediglich der Seed zur Generierung der Punktwolke geändert. Der Ablauf ist dann bis auf Schritt 5 der gleiche wie in [Abbildung 39](#).

5.5.4 Export

Ein weiteres wichtiges Feature der Software ist es, die erstellte Pflanzenstruktur und ihre Textur exportieren zu können. Die Schritte 14 bis 16, sowie 18 bis 20 in [Abbildung 39](#) verdeutlichen, dass jedes Mal, wenn die Geometriedaten neu berechnet werden, der **Core** involviert ist. Damit später ein Export der Geometriedaten einfach möglich ist, speichert er diese nach jeder Berechnung. Drückt der Nutzer den Save-Button, so werden die Geometriedaten mit einem Algorithmus von Hlöðversson [\[27\]](#) in das OBJ-Format überführt und zusammen mit der Textur im PNG-Format gespeichert. Der Dateiname der OBJ-Datei enthält dabei die Anzahl der Vertices und Dreiecke. So ist beim Export mehrerer Modelle mit großer Wahrscheinlichkeit gewährleistet, dass keine bereits gespeicherten Modelle überschrieben werden. Gleichermaßen wird bei der Textur mit eindeutigen Namen für die Farben erreicht.

6 Evaluierung

In diesem Kapitel wird evaluiert, inwieweit der Funktionsumfang der implementierten Software den definierten funktionalen und nicht-funktionalen Anforderungen genügt. Anschließend wird auf einige Limitationen der verwendeten Verfahren eingegangen.

6.1 Erfüllung der Anforderungen

6.1.1 Funktionale Anforderungen

Obligatorisch:

FAO01 Die erzeugten Pflanzenstrukturen besitzen charakteristische äußerliche Merkmale echter Pflanzen und wirken dadurch realistisch, die nachfolgenden Abbildungen zeigen dies.

FAO02 Dekurrente Strukturen können erzeugt werden. Nachfolgende Abbildungen zeigen Beispiele dafür. In [Abbildung 41](#) wurde eine Punktwolke gewählt, deren Höhe größer als ihre Breite und Tiefe ist.



Abbildung 40: Breiter dekurrenter Baum



Abbildung 41: Hoher dekurrenter Baum

FAO03 Exkurrente Strukturen können erzeugt werden, Abbildung 42 zeigt dies. Die Punktwolke wurde dazu stark in die Höhe gezogen und ihre untere Hälfte abgeschnitten, wodurch eine konische Form entsteht. Über den anpassbaren Parameter **CrownStemLengthRatio** (in den **GrowthProperties**) wurde zudem der initiale Stamm vollständig in die Baumkrone verlängert. Damit schon zu Beginn des Wachstums viele Äste entstehen, wurde außerdem eine höhere Verzweigungsdichte für den Anfang gewählt.



Abbildung 42: Exkurrenter Baum

6 Evaluierung

FAO04 Es kann eingestellt werden, wie sehr die Äste der erzeugten Verzweigungsstrukturen herunterhängen. In Abbildung 43 sind Beispiele für unterschiedliche Intensitäten dargestellt - links beträgt Lambda 0.4, in der Mitte und rechts 0.8.

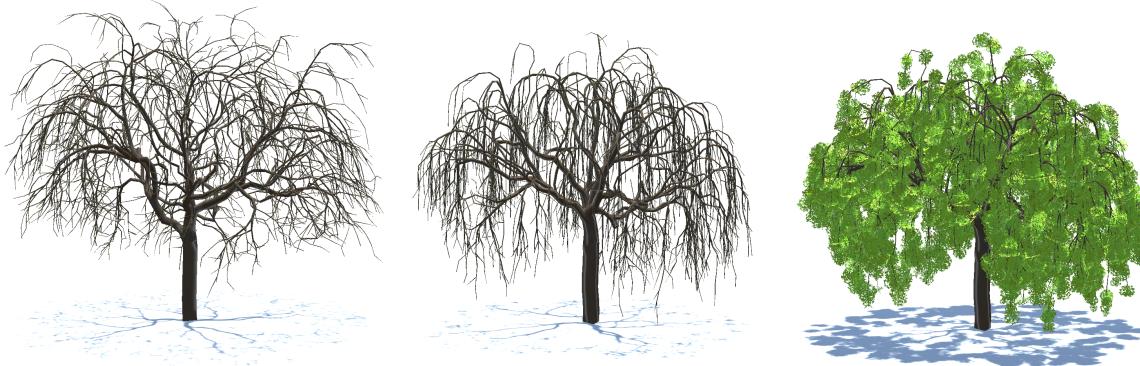


Abbildung 43: Bäume mit herunterhängenden Ästen ab einer Zweigordnung von 25

FAO05 Die Farbe der Borke ist einstellbar. Zur Demonstration ist die Borke in Abbildung 44 gräulich gefärbt.

FAO06 Die Anzahl und Farbe der Blätter ist einstellbar, beides ist in Abbildung 44 sichtbar.

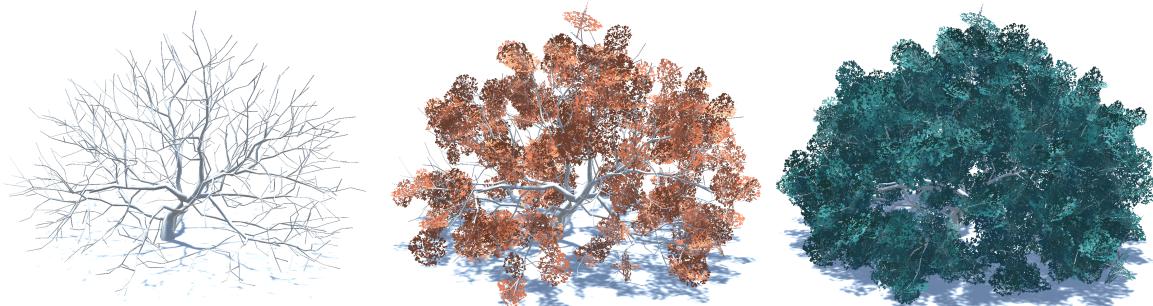


Abbildung 44: Busch

FAO07 Das Alter der Pflanzenstrukturen ist einstellbar. Die Baumkronen in Abbildung 40 und 45 unterscheiden sich lediglich durch diesen Parameter.



Abbildung 45: Junger dekurrenter Baum

FAO08 Bei Programmstart wird eine zufällig erzeugte dreidimensionale Baumstruktur mit Blättern dargestellt. Dies geschieht durch die Initialisierung, die der **Core** zu Programmstart durchführt.

FAO09 Die Generierung eines neuen Seeds ist möglich, dabei behalten alle eingestellten Parameter ihre Werte. Die dargestellten Bäume in [Abbildung 46](#) unterscheiden sich nur in ihrem Seed.



Abbildung 46: Dekurrente Bäume mit unterschiedlichem Seed

FAO10 Die Benutzeroberfläche ermöglicht es, die derzeit erstellte Struktur von allen Seiten zu betrachten. Dies wird umgesetzt, indem für die Kamerasteuerung Kugelkoordinaten verwendet werden.

FAO11 Erzeugte Pflanzenstrukturen können im OBJ-Format inklusive ihrer Textur exportiert werden.

FAO12 Die Stammdicke ist einstellbar, die Darstellungen in [Abbildung 42](#) verdeutlichen dies.

FAO13 Die Stammlänge ist einstellbar. Der Busch in [Abbildung 44](#) besitzt beispielsweise eine Stammlänge von 0, der breite dekurrente Baum in [Abbildung 40](#) eine Stammlänge von 2.

FAO14 Der Rechenfortschritt der Software wird angezeigt. Entscheidend hierfür ist, dass nach jeder Iteration des Space Colonization Algorithmus der **Core** benachrichtigt wird, so dass der **TreeRenderer** dann eine Darstellung der derzeit berechneten Struktur verursacht.

FAO15 Die Anzahl der Vertices und Dreiecke wird durch zwei Faktoren niedrig gehalten. Für die Zweigmodellierung wird standardmäßig eine Kreisauflösung von 3 verwendet, so dass der Querschnitt der Äste dreieckig ist. Dies fällt insbesondere bei dünneren Strukturen wenig auf. Die Nutzung von Cross-Foil-Meshes reduziert die Anzahl der benötigten Vertices und Dreiecke gegenüber der Modellierung einzelner Blätter.

Werden viele kleine Blätter verwendet (wie in [Abbildung 43](#) links), ist eine hohe Anzahl an Vertices und Dreiecken unvermeidbar. Gleiches gilt derzeit für große Bäume mit hoher Verzweigungsdichte und Kreisauflösung. [Abbildung 52](#) zeigt ein Beispiel.

Fakultativ:

FAF1 Es kann nicht gewählt werden, ob die erzeugte Pflanzenstruktur Wurzelansätze hat oder nicht. Eine Möglichkeit zur Umsetzung wäre hier eine weitere Wachstumsphase mittels eigener Punktwolke und Space Colonization Algorithmus.

FAF2 Der Grad der Astkrümmung ist einstellbar. In [Abbildung 47](#) links ist diese minimal und rechts maximal.



Abbildung 47: Bäume mit minimal und maximal gekrümmten Ästen

FAF3 Die Textur der Borke sowie der Blätter ist über eine ladbare PNG-Datei einstellbar. Die Ausrichtung der Blätter erfolgt hierbei in jedem Fall zufällig. Es wird also bei der Verwendung von einfachen Texturlappen keine Orientierung am Licht simuliert, was in zukünftige Versionen noch integriert werden sollte.

FAF4 Blüten und Früchte können nicht hinzugefügt werden.

FAF5 Für den Export ist es nur indirekt möglich, zwischen verschiedenen Auflösungen zu wählen. So kann, wie in [Abbildung 41](#) rechts dargestellt, eine geringe Blattdichte mit dafür großen Texturlappen konfiguriert werden. Dadurch wird im Beispiel die Anzahl der benötigten Vertices und Dreiecke von 13488/9622 auf 5792/5774 reduziert, während

das Ergebnis aus etwas Entfernung noch einen guten Gesamteindruck macht. Außerdem kann die Auflösung der Kreise für die generalisierten Zylinder eingestellt werden.

FAF6 Bei der Entwicklung der Software wurde versucht, die sieben Grundsätze der Dialoggestaltung einzubeziehen. Da keine Testpersonen zur Benutzerfreundlichkeit des Programms befragt wurden und eine Einbeziehung in den Entwicklungsprozess ebenfalls nicht erfolgte, ist eine objektive Bewertung nur eingeschränkt möglich.

Aufgabenangemessenheit Es können effektiv verschiedene Bäume und Sträucher modelliert werden, jedoch ist es an manchen Stellen nicht gelungen, die zugrundeliegende Technologie zu verbergen. Dies wird im nächsten Abschnitt noch genauer erläutert.

Selbstbeschreibungsfähigkeit Der Aufbau der Benutzeroberfläche ist einfach gehalten und die Elemente sind mit Textfeldern beschrieben. Dadurch, dass die Parameter nur wenig technische Natur besitzen, ist prinzipiell kaum Grundwissen für die Bedienung erforderlich. Auf Limitationen diesbezüglich wird im nächsten Abschnitt noch eingegangen.

Erwartungskonformität Für die Gestaltung der Benutzeroberfläche wurden übliche Elemente wie Buttons, Slider und Dropdowns verwendet. Eine Rotation der Szene mittels gehaltenem Mausklick und Mausbewegung liegt nahe. Die anpassbaren Parameter entsprechen dem Nutzungskontext der Pflanzenmodellierung.

Lernförderlichkeit Die Anzahl der Elemente mit denen interagiert werden kann ist relativ gering. So kann beim Erlernen des Umgangs mit der Software nach dem Prinzip “Learning by doing” vorgegangen werden. Die Darstellung der Punktwolke während der Verformung sollte verständlich machen, was die Parameter bedeuten. Trotz allem ist es bei der Verwendung des Programms von Vorteil, Wissen über die Arbeitsweise des zugrundeliegenden Algorithmus zu haben.

Steuerbarkeit Die anpassbaren Parameter können bei einfachen Strukturen in beliebiger Reihenfolge und Geschwindigkeit verändert werden. Da die Änderung von **GeometryProperties** immer eine sofortige Neuberechnung der Geometriedaten nach sich zieht, kommt es hier schon bei etwas größeren Bäumen (ab ca. 8000 Knoten) zu spürbaren Verzögerungen. Bei Bäumen ab ca. 20 000 Knoten ist eine Änderung von **GrowthProperties** ebenfalls nicht mehr flüssig, da immer darauf gewartet wird, dass der Thread, in dem die Verzweigungsstruktur wächst, beendet wird. Nachrichten an den Benutzer werden für sechs Sekunden angezeigt und verschwinden danach automatisch. Optimalerweise sollte hier der Nutzer bestimmen können, wann die Nachricht ausgeblendet wird.

Fehlertoleranz Die Software ist gegenüber fehlerhaften Eingaben tolerant, da diese sofort wieder korrigiert werden können. Eine Ausnahme bildet hierbei der “New Seed”-Button, denn ein Widerrufen der Seedänderung ist nicht möglich.

Individualisierbarkeit Eine Individualisierbarkeit des Dialoges der Software ist nicht möglich.

6.1.2 Nicht-funktionale Anforderungen

Obligatorisch:

NFAO1 Dadurch, dass die Berechnungen zur Erzeugung der Verzweigungsstruktur in einem separaten Thread stattfinden, kann jederzeit mit der Benutzeroberfläche interagiert werden. Jede Veränderung der Parameter fließt dabei sofort in die Berechnungen ein, so dass der Nutzer sofort eine Reaktion der Software auf seine Interaktion sieht. Ein Beenden der Software während der Berechnungen ist ebenfalls möglich, der laufende Thread wird dazu, wie in [Kapitel 5](#) beschrieben, gestoppt.

NFAO2 Die Software kann über Unity für die Betriebssysteme Linux, macOS und Windows kompiliert werden und läuft auf allen Systemen fehlerfrei.

Fakultativ:

NFAO1 Der Space Colonization Algorithmus wurde so sehr optimiert, dass eine interaktive Verwendung einwandfrei möglich ist.

NFAF2 Die Software läuft überwiegend stabil, es existieren jedoch weder Tests noch qualitative oder quantitative Daten zur Stabilität. Bei der Erzeugung der abgebildeten Ergebnisse traten keine Exceptions auf und es kam auch nicht zu Programmabstürzen.

6.2 Limitationen

Über die Modellierung der Punktwolke des Space Colonization Algorithmus ist es einfach, die Baumform zu kontrollieren. Dies bringt jedoch auch Nachteile mit sich. Wird die Punktwolke gerade so komplett ausgefüllt, nimmt die Baumkrone exakt die Form der Punktwolke an, was vor allem bei größeren Bäumen unrealistisch aussehen kann. [Abbildung 48](#) zeigt ein Beispiel. Denn in der Realität kommt es häufig vor, dass Bereiche der Baumkrone am Rand “ungenutzt” bleiben. Um einen ähnlichen Effekt zu erzielen, könnte die Punktwolke etwas verrauscht werden, so dass ihr Umriss nicht mehr perfekt gerade ist.

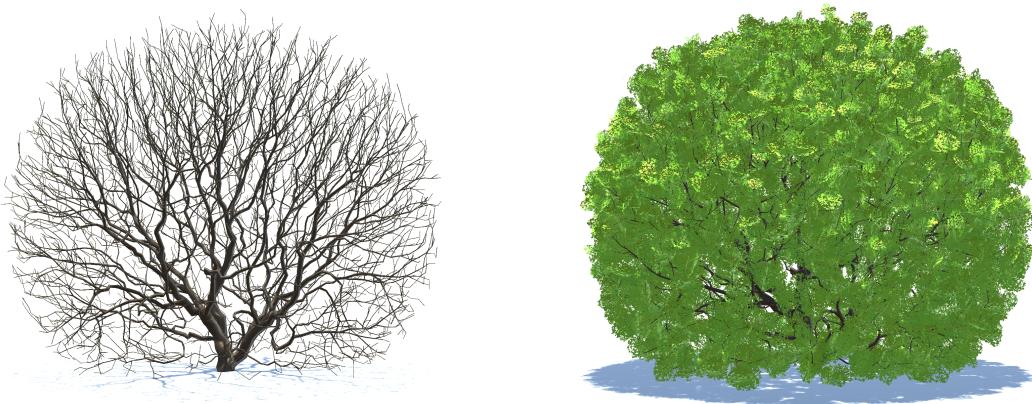


Abbildung 48: Baum bei exakt ausgefüllter Punktwolke

In [Abbildung 48](#) rechts wird außerdem sichtbar, dass die Blätter mit dem verwendeten Verfahren unabhängig von den Lichtverhältnissen platziert werden. In der Realität hätten die

6 Evaluierung

unteren Äste weniger Blätter, da es dort schattig ist.

Wenn die Pflanzenstruktur den Rand der Punktwolke bereits erreicht hat, bevor die definierte Anzahl an Iterationen durchlaufen wurde, kommt es ebenfalls zu unrealistischen Ergebnissen. Ist die Anzahl der “übrigen” Iterationen bei Erreichen des Randes klein, entsteht eine Struktur wie in Abbildung 49 links. Die Äste wachsen dann am Rand der Punktwolke um die bereits bestehende Struktur herum, statt dorthin, wo theoretisch noch Platz wäre.

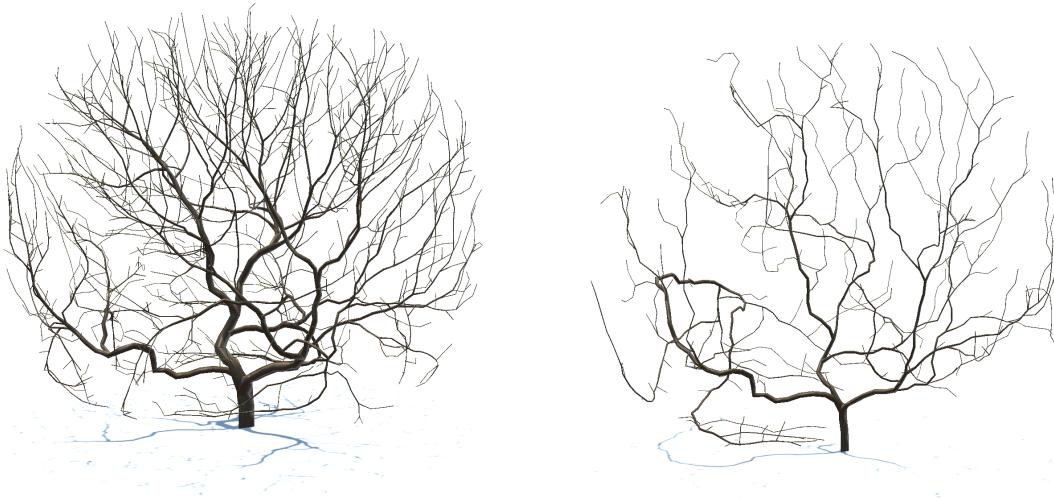


Abbildung 49: Bäume beim frühzeitigen Erreichen des Punktwolkenrandes

Wird der Rand schon sehr früh erreicht, kommt es zu einer Struktur wie in Abbildung 49 rechts, welche sich lediglich durch die Anzahl der Iterationen von der Struktur links unterscheidet. Im Beispiel wurde eine geringe Astdichte zu Beginn und eine hohe Astdichte zum Ende definiert. Dadurch, dass die Punktwolke jedoch schon nach einem Bruchteil der Iterationen ausgefüllt wurde, kommt es nicht zur Ausprägung von stärker verzweigten Ästen, denn die Astdichte nimmt erst in den späteren Iterationen zu. Erkennt das Programm einen frühzeitigen Stopp des Wachstums, wird der Nutzer über ein Textfeld darüber informiert, dass die Punktwolke größer oder ein geringeres Alter gewählt werden sollte. Die Erkennung ist jedoch nicht immer einfach, da das Wachstum manchmal trotzdem minimal weitergeht. In diesem Fall ist für den Nutzer, ohne Kenntnisse über die zugrundeliegenden Algorithmen, nicht ersichtlich, wieso das Ergebnis keine hohe Astdichte aufweist. In seltenen Fällen passiert es außerdem, dass die Pflanze besonders schnell zum oberen oder einen der seitlichen Ränder der Punktwolke wächst und erst danach den übrigen Raum besiedelt. Dies sieht, wenn es passiert, unrealistisch aus.

Die eben erläuterten Probleme treten nicht auf, wenn die Punktwolke größer als die resultierende Verzweigungsstruktur gewählt wird. Auf diesem Weg werden auch die realistischsten Ergebnisse erzielt. Bis auf in Abbildung 41 und 42 hat keiner der gezeigten Bäume die Punktwolke ausgefüllt. Der Busch in Abbildung 44 ist zwar durch eine niedrigere Punktwolke entstanden, jedoch nur deshalb, weil in diesem Fall die Tropismen in Richtung Licht automatisch gedämpft werden.

Durch ungünstige Verteilungen der Anziehungspunkte kommt es in seltenen Fällen dazu, dass

6 Evaluierung

plötzlich das Wachstum aufhört. Denn durch das Löschen der Anziehungspunkte (am Ende jeder Iteration des Space Colonization Algorithmus) kann es passieren, dass für den nächsten Schritt keine Anziehungspunkte mehr in der Nähe eines Knotens sind - obwohl davon eigentlich noch viele vorhanden sind. Dies ist vor allem zu Beginn des Wachstums ein Problem, da dann noch nicht viele Knoten existieren. Hört das Wachstum auf, bevor die definierte Anzahl von Iterationen durchlaufen wurde, wird dem Nutzer mitgeteilt, dass er einen neuen Seed benutzen sollte - dies behebt das Problem in den meisten Fällen.

Bei dekurrenten Bäumen lässt sich über die Anzahl der Iterationen das "Alter" der Baumkrone konfigurieren. Die initiale Stammlänge muss extra angepasst werden. Bei exkurrenten Bäumen muss zusätzlich die Form der Punktewolke entsprechend aktualisiert werden, da die Länge des verlängerten Hauptstammes von ihrer Höhe abhängt. Aus diesem Grund wurde die Bezeichnung "Growth Iterations" statt "Age" für den entsprechenden Slider gewählt. Eine direkte Konfiguration des Alters ist also nur bei dekurrenten Strukturen ohne Hauptstamm und groß gewählter Punktewolke möglich.

Die Methode zur Modellierung herunterhängender Äste findet in einem einfachen Nachbearbeitungsschritt der Verzweigungsstruktur statt. Dadurch muss bei Veränderungen der Parameter die Verzweigungsstruktur nicht neu berechnet werden. Ein Nachteil der Methode ist, dass der verfügbare Platz nicht in die Berechnungen einfließt, wodurch Überlappungen von Ästen theoretisch möglich sind. Außerdem arbeitet der Algorithmus derzeit mit der Zweigordnung nach monochasialer Verzweigung. Dadurch kommt es vor allem bei exkurrenten Bäumen zu unrealistischen Ergebnissen, da aus Sicht des Algorithmus keine Hauptachsen existieren.

Abbildung 50 zeigt das.



Abbildung 50: Herunterhängende Äste bei einem exkurrenten Baum

Aus dem gleichen Grund hat eine Veränderung der Stammlänge derzeit auch Auswirkungen auf die Zweigbiegungen. Denn dadurch werden Knoten zum Baum hinzugefügt oder entfernt. Somit verändert sich die Zweigordnung aller untergeordneten Äste in der Baumkrone und demzufolge werden diese anders gebogen (siehe Abbildung 50 rechts).

7 Zusammenfassung und Ausblick

Mit Hilfe der entwickelten Software kann eine Vielzahl verschiedener Bäume und Sträucher erzeugt und anschließend exportiert werden. Durch die Verwendung des Space Colonization Algorithmus wird eine automatische Generierung von Verzweigungsstrukturen möglich. So können diejenigen Parameter eingespart werden, die überhaupt erst zu einem realistischen Aussehen führen - im Beispiel von SpeedTree in [Abschnitt 3.1](#) sind das die drei aneinander gehangenen Verzweigungskomponenten. Jede dieser Verzweigungskomponenten besitzt in SpeedTree einen Satz von Parametern. Bei der entwickelten Software hingegen dient ein einziger Parametersatz dazu, die charakteristischen Merkmale für die gesamte Verzweigungsstruktur zu definieren. Die Parameter sind so gewählt, dass sie eine deklarative Natur besitzen, das heißt, es muss lediglich definiert werden, welche äußerlichen Erscheinungsmerkmale die Verzweigungsstruktur haben soll. Nicht wie diese entsteht. Inwieweit die Benutzerfreundlichkeit davon tatsächlich profitiert, muss noch anhand von Testnutzern evaluiert werden.

Von den insgesamt 20 Parametern dienen 12 dazu, die Verzweigungsmerkmale der Baumkrone zu definieren. Fünf der 12 Parameter werden zur Modellierung der Punktfolge verwendet. Ursprünglich war geplant, auf diesem Weg verschiedene Formen von Bäumen zu ermöglichen. Aus den im letzten Kapitel erläuterten Limitationen geht jedoch hervor, dass die alleinige Modellierung der Punktfolge zur Definition der Baumform nicht gut geeignet ist. Es sollte also für zukünftige Versionen nach einer besseren Lösung gesucht werden.

Die Optimierung der Distanzberechnungen macht das Verfahren auch zur interaktiven Modellierung größerer Bäume nutzbar. Die in [Abschnitt 6.2](#) beschriebenen Schwachstellen bezüglich der Performance (Anforderung [FAF6](#): Steuerbarkeit), können mittels konsequenterer asynchroner Kommunikation zwischen Darstellungsschicht und Model behoben werden.

Zusätzlich zu den nicht erfüllten Anforderungen, sollten in Weiterentwicklungen der Software folgende Features einfließen:

- Interpolation zwischen Knoten für glattere Rundungen. Dabei muss darauf geachtet werden, die Anzahl der Vertices und Dreiecke niedrig zu halten
- Animation von Seitenwind inklusive geeignetem Format für den Export (verbreitetes Feature unter den bisherigen Tools)
- Bereitstellung von realistischen Stamm- und Blatttexturen

7 Zusammenfassung und Ausblick

- Aufhübschung der Benutzeroberfläche

Um die Anzahl der Parameter weiter zu reduzieren und die auftretenden Probleme mit der Punktwolke zu beheben, ist bei der Weiterentwicklung die Verwendung eines Verfahrens von Palubicki et al. [7] denkbar. Dieses simuliert den Wachstumsprozess von Pflanzen präziser, indem nicht nur der verfügbare Platz, sondern auch die Lichtverhältnisse für das Austreiben und Wachsen der Knospen eine Rolle spielen. Dabei werden ebenfalls nur wenige Parameter benutzt, um sehr realistische Darstellung von verschiedenen Bäumen und Sträuchern zu ermöglichen. Die derzeit eingesetzte prozedurale Stammerzeugung könnte so ersetzt werden und exkurrente Bäume sind über Apikalkontrolle modellierbar. Die Konfigurierbarkeit des Alters der Pflanze wäre mit nur einem Parameter gegeben und die Platzierung der Blätter könnte ebenfalls anhand der Lichtverhältnisse geschehen. Der Space Colonization Algorithmus ist ein Teil des Verfahrens. Welche Knospen austreiben und wachsen, wird jedoch auf anderem Weg entschieden. Die Eignung des Verfahrens von Palubicki et al. müsste für eine Integration noch genauer untersucht werden, die grundlegende Softwarearchitektur ist jedoch weiterhin verwendbar.



A Anhang



Abbildung 51: Testbaum für Optimierungsschritte

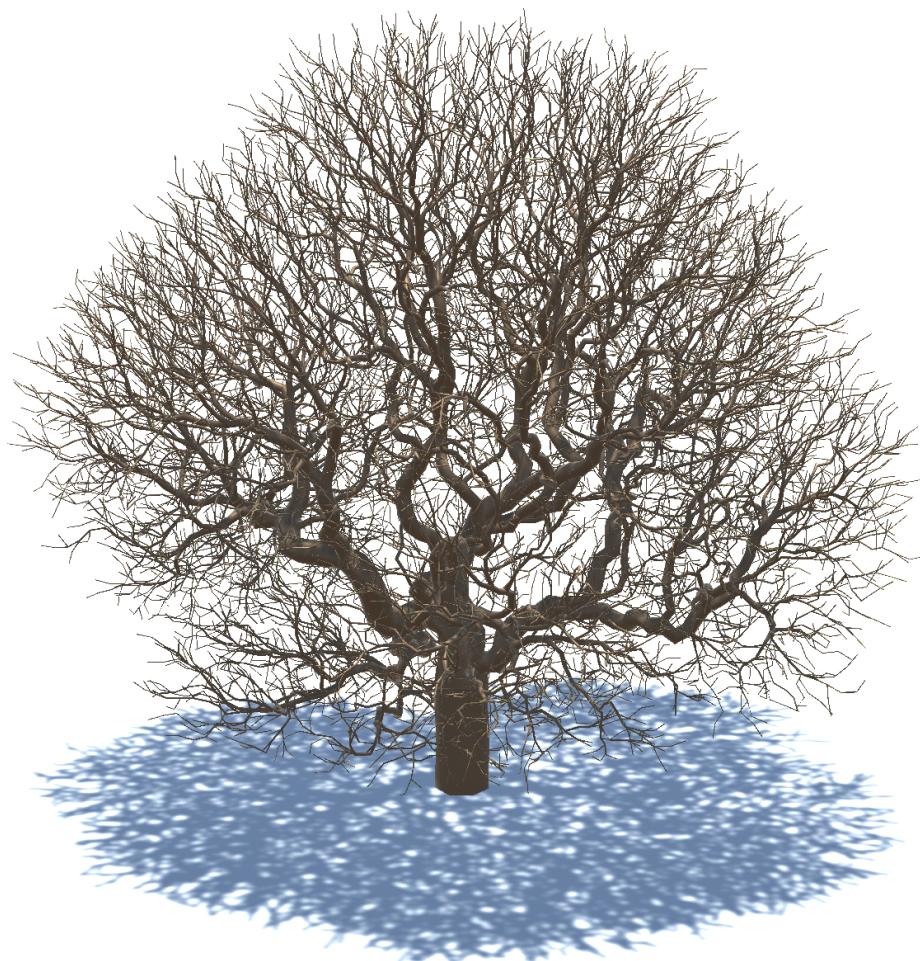


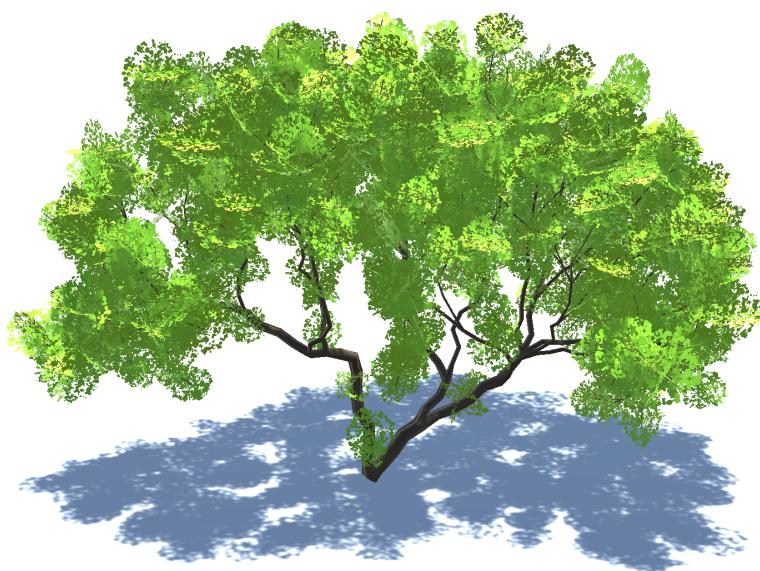
Abbildung 52: Baum mit hoher Astkrümmung, in 50 Iterationen durch prinzipiell offene Punktwolke entstanden



Abbildung 53: Buschvariationen

A Anhang

Baum	Vertices	Dreiecke	Benötigte Rechenzeit in Sekunden
Abbildung 40 (Links)	10292	13458	0,8
Abbildung 40 (Rechts)	25620	21122	0,8
Abbildung 41 (Links)	3328	4542	0,13
Abbildung 41 (Mitte)	13488	9622	0,13
Abbildung 41 (Rechts)	5792	5774	0,13
Abbildung 42 (Links)	7832	10380	0,2
Abbildung 42 (Rechts)	18480	15492	0,2
Abbildung 43 (Links)	18984	25428	1,3
Abbildung 43 (Mitte)	18984	25428	1,3
Abbildung 43 (Rechts)	78376	55124	1,3
Abbildung 44 (Links)	5840	7776	0,5
Abbildung 44 (Mitte)	9440	9576	0,5
Abbildung 44 (Rechts)	18080	13896	0,5
Abbildung 45 (Links)	1808	2406	0,3
Abbildung 45 (Rechts)	4368	3686	0,3
Abbildung 46 (Links)	14188	18570	1,3
Abbildung 46 (Mitte)	15344	20046	1,6
Abbildung 46 (Rechts)	11700	15324	1,8
Abbildung 47 (Links)	16000	21354	1,9
Abbildung 47 (Rechts)	12188	16428	1,6
Abbildung 51	24101	31200	3,4
Abbildung 52	190504	294392	19,5



Abbildungsverzeichnis

1	Pflanzenaufbau, eigene Darstellung	3
2	Verzweigungsarten, Nachbildung in Anlehnung an [4, S. 3]	4
3	Proleptisches und sylleptisches Wachstum, modifizierte Darstellung aus [7, S. 3]	5
4	Generalisierte Zylinder, linke Abbildung aus [14, S. 306], rechte Abbildung Nachbildung in Anlehnung an [14, S. 306]	6
5	Einheitskreis, modifizierte Darstellung aus [15]	7
6	Grundsätze der Dialoggestaltung nach DIN EN ISO 9241-110, aus [16, S. 61]	8
7	In SpeedTree erstellter Baum mit zugehörigem Graphen, aus [17]	10
8	Eiche, aus [23]	12
9	Pappel, aus [24]	12
10	Baum nach Oppenheimer, aus [28, S. 60]	17
11	Ableitung eines einfachen L-Systems, aus [30, S. 4]	18
12	Funktionsweise einer Turtle, aus [30, S. 7]	19
13	Konstruktion der Koch-Kurve, eigene Darstellung	19
14	Mit L-System erzeugte Pflanzenstruktur, eigene Darstellung unter Verwendung von [31]	20
15	Ausrichtung und Befehle der Turtle im dreidimensionalen Raum, aus [30, S. 19]	20
16	Dreidimensionale Pflanzenstruktur, aus [30, S. 26]	21
17	Grundlegende Schritte des Verfahrens, modifizierte Darstellung aus [25, S. 64]	23
18	Der Space Colonization Algorithmus, modifizierte Darstellung aus [25, S. 65]	24
19	Auswirkung der Anzahl von Anziehungspunkten und Größe der Löschdistanz, modifizierte Darstellung aus [25, S. 66]	25
20	Auswirkung der Einflussdistanz, eigene Darstellung	25
21	Platzierung der Anziehungspunkte am Rand der Punktwolke, aus [25, S. 67]	26
22	Methode zur Modellierung herunterhängender Äste, eigene Darstellung	27
23	SpeedTree Cluster, aus [36]	28
24	SpeedTree Cross-Foil-Mesh ohne Textur, aus [17]	28
25	Berechnung von Kugelkoordinaten, modifizierte Darstellung aus [37]	29
26	Wahrnehmungswinkel nach Palubicki et al., modifizierte Darstellung aus [7, S. 3]	33
27	Benutzeroberfläche, eigene Darstellung	34
28	Grundlegende Komponenten der geplanten Software, eigene Darstellung	35

Abbildungsverzeichnis

29	Generalisierter Zylinder mit eingezeichneten Normalen, eigene Darstellung	37
30	Datenstruktur zur Speicherung der Verzweigungsdaten und Erzeugung der Geometriedaten, eigene Darstellung	37
31	Verbesserte Darstellung unter Verwendung von modifizierten Duplikatknoten, eigene Darstellung	38
32	Essenz der relevanten Klassen für den Wachstumsvorgang, eigene Darstellung	39
33	Sigmoid-Funktion, aus [51]	41
34	Konstruktion der Punktwolke im zweidimensionalen Raum, eigene Darstellung	43
35	Beispiel für die Suche nach in Frage kommenden nächstgelegenen Knoten, eigene Darstellung	45
36	Core-Klasse im Zusammenhang, eigene Darstellung	46
37	Szenengraph der entwickelten Software, eigene Darstellung	47
38	Textur bestehend aus Stammfarbe und Blattpartikeln, eigene Darstellung	48
39	Ablauf beim Verändern des “CrownHeight”-Sliders, eigene Darstellung	49
40	Breiter dekurrenter Baum, eigene Darstellung	52
41	Hoher dekurrenter Baum, eigene Darstellung	53
42	Exkurrenter Baum, eigene Darstellung	53
43	Bäume mit herunterhängenden Ästen, eigene Darstellung	54
44	Busch, eigene Darstellung	54
45	Junger dekurrenter Baum, eigene Darstellung	55
46	Dekurrente Bäume mit unterschiedlichem Seed, eigene Darstellung	55
47	Bäume mit minimal und maximal gekrümmten Ästen, eigene Darstellung	56
48	Baum bei exakt ausgefüllter Punktwolke, eigene Darstellung	58
49	Bäume beim frühzeitigen Erreichen des Punktwolkenrandes, eigene Darstellung	59
50	Herunterhängende Äste bei einem exkurrenten Baum, eigene Darstellung	60
51	Testbaum für Optimierungsschritte, eigene Darstellung	63
52	Baum mit hoher Astkrümmung, eigene Darstellung	64
53	Buschvariationen, eigene Darstellung	64

Quellenverzeichnis

- [1] Oliver Deussen, *Computergenerierte Pflanzen* (2003), Berlin Heidelberg New York: Springer-Verlag, ISBN: 3-540-43606-5
- [2] Interactive Data Visualization, Inc., *SpeedTree*, URL: <https://store.speedtree.com/>, letzter Zugriff: 02.07.2019
- [3] EVOLVED-Software, *TreeIt*, URL: <http://www.evolved-software.com/treeit/>, letzter Zugriff: 02.07.2019
- [4] Veit M. Dörken, *Morphologie und Anatomie der Sprossachse*, URL: https://cms.uni-konstanz.de/fileadmin/biologie/ag-doerken/pdf/Morphologie/1_Sprossachse.pdf, letzter Zugriff: 28.06.2019
- [5] Pflanzenforschung.de, Lexikon A-Z, *Verzweigungstypen*, URL: <https://www.pflanzenforschung.de/de/themen/lexikon/verzweigungstypen-268/>, letzter Zugriff 30.06.2019
- [6] Morris G. Cline und Constance A. Harrington, *Apical dominance and apical control in multiple flushing of temperate woody species* (2007), Canadian Journal of Forest Research, Volume 37, März 2007, S. 74-83
- [7] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch und Przemysław Prusinkiewicz, *Self-organizing tree models for image synthesis* (2009), ACM Transactions on Graphics, Volume 28 Issue 3, August 2009, Artikel Nr. 58, S. 1-10
- [8] Spektrum, Lexikon der Biologie, *Monochasium*, URL: <https://www.spektrum.de/lexikon/biologie/monochasium/43719>, letzter Zugriff: 30.06.2019
- [9] Peter del Tredici, Arnold Arboretum, *Tree Architecture Definitions* (2013), URL: <https://www.arboretum.harvard.edu/wp-content/uploads/Tree-Architecture.pdf>, letzter Zugriff: 29.06.2019
- [10] Daniel Barthélémy und Yves Caraglio, *Plant Architecture: A Dynamic, Multilevel and Comprehensive Approach to Plant Form, Structure and Ontogeny* (2007), Annals of Botany 99, S. 375-407
- [11] Spektrum, Lexikon der Biologie, *Plagiotropismus*, URL: <https://www.spektrum.de/lexikon/biologie/plagiotropismus/52086>, letzter Zugriff: 30.06.2019

Quellenverzeichnis

- [12] Spektrum, Lexikon der Biologie, *Tropismen*, URL: <https://www.spektrum.de/lexikon/biologie/tropismus/67893>, letzter Zugriff: 30.06.2019
- [13] Gerald Jacob Agin, *Representation and Description of Curved Objects* (1972), Stanford Artificial Intelligence Report, Memo AIM-173, Oktober 1972
- [14] Jules Bloomenthal, *Modeling the Mighty Maple* (1985), ACM SIGGRAPH Computer Graphics, Volume 19 Issue 3, Juli 1985, S. 305-311
- [15] Serlo Mathematik, *Trigonometrie am Einheitskreis*, URL: <https://de.serlo.org/mathe/geometrie/sinus-kosinus-tangens/sinus-kosinus-tangens-einheitskreis/trigonometrie-einheitskreis>, letzter Zugriff: 07.07.2019
- [16] Jochen Prümper, Jörn Hurtienne, Petra Abele, *Usability Management bei SAP-Projekten* (2007), Wiesbaden: Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH, ISBN: 978-3-8348-0244-6
- [17] Michal Seacrest (SpeedTree), *SpeedTree Tutorial: Modeler Basics* (2012), URL: https://www.youtube.com/watch?v=sbmQhdQ_2VI, letzter Zugriff: 20.08.2019
- [18] Interactive Data Visualization, Inc., *SpeedTree 7 Features*, URL: <https://store.speedtree.com/games/>, letzter Zugriff: 22.08.2019
- [19] Unity Technologies, *Unity*, URL: <https://unity.com/de>, letzter Zugriff: 22.08.2019
- [20] Study Break Tutorials, *Easy Unity tree tutorial* (2018), URL: <https://www.youtube.com/watch?v=MgfixiLs70zK>, letzter Zugriff: 22.08.2019
- [21] Resurrection 21, *Treeit Guide for Unreal engine 4 - Amazing Tree software* (2017), URL: <https://www.youtube.com/watch?v=nXW3hk1Ptzs>, letzter Zugriff: 22.08.2019
- [22] Blender Foundation, *About*, <https://www.blender.org/about/>, letzter Zugriff: 22.08.2019
- [23] URL: <https://upload.wikimedia.org/wikipedia/commons/8/89/Eiche.JPG>, letzter Zugriff: 21.07.2019
- [24] URL: https://upload.wikimedia.org/wikipedia/commons/6/69/Pappel_Wonfurt,_2.jpg, letzter Zugriff: 21.07.2019
- [25] Adam Runions, Brendan Lane und Przemysław Prusinkiewicz, *Modeling Trees with a Space Colonization Algorithm* (2007), Proceedings of the 2007 Eurographics Workshop on Natural Phenomena, S. 63-70
- [26] Dibya Chakravorty, *8 Most Common 3D File Formats in 2019*, URL: <https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/>, letzter Zugriff: 20.07.2019

Quellenverzeichnis

- [27] Hrafnkell Freyr Hlöðversson, *ObjExporter*, URL: <http://wiki.unity3d.com/index.php/ObjExporter>, letzter Zugriff: 20.07.2019
- [28] Peter E. Oppenheimer, *Real Time Design and Animation of Fractal Plants and Trees* (1986), ACM SIGGRAPH Computer Graphics, Volume 20 Issue 4, August 1986, S. 55-64
- [29] Jason Weber, Joseph Penn, *Creation and Rendering of Realistic Trees* (1995), SIGGRAPH '95 Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, S. 119-128, New York: ACM, ISBN: 0-89791-701-4
- [30] Przemysław Prusinkiewicz, Aristid Lindenmayer, *The Algorithmic Beauty of Plants* (2004), New York: Springer-Verlag, elektronische Version
- [31] Kevin Roast, *L-Systems*, URL: <http://www.kevs3d.co.uk/dev/lsystems/>, letzter Zugriff: 27.07.2019
- [32] Bernd Lintermann, Oliver Deussen, *Erzeugung komplexer botanischer Objekte in der Computergrafik* (1997), URL: <http://graphics.uni-konstanz.de/publikationen/Deussen1997Erzeugungkomplexerbotanischer/Deussen1997Erzeugungkomplexerbotanischer.pdf>, letzter Zugriff: 26.07.2019
- [33] Radoslaw Karwowski, Przemysław Prusinkiewicz, *Design and Implementation of the L+C Modeling Language* (2003), Electronic Notes in Theoretical Computer Science 86(2), S. 19ff.
- [34] K. Shinozaki, K. Yoda, K. Hozumi, T. Kira, *A quantitative analysis of plant form - the pipe model theory. I. Basic analyses* (1964), Japanese Journal of Ecology 13, 3 (1964), S. 97-104
- [35] N. MacDonald, *Trees and networks in biological models* (1983), New York: J. Wiley & Sons
- [36] Sonia Piasecki (SpeedTree), *SpeedTree 8.3: How to Make Clusters* (2019), URL: <https://www.youtube.com/watch?v=0IR6zpisaMM>, letzter Zugriff: 20.08.2019
- [37] Eric Weisstein, *Spherical Coordinates*, MathWorld - A Wolfram Web Resource, URL: <http://mathworld.wolfram.com/SphericalCoordinates.html>, letzter Zugriff: 28.07.2019
- [38] Khronos Group, *OpenGL Overview*, URL: <https://www.opengl.org/about/>, letzter Zugriff: 27.07.2019
- [39] Unity Technologies, *Graphics API support*, URL: <https://docs.unity3d.com/Manual/GraphicsAPIs.html>, letzter Zugriff: 27.07.2019
- [40] Unity Technologies, *GameObject*, URL: <https://docs.unity3d.com/ScriptReference/GameObject.html>, letzter Zugriff: 14.08.2019

Quellenverzeichnis

- [41] Unity Technologies, *Component*, URL: <https://docs.unity3d.com/ScriptReference/Component.html>, letzter Zugriff: 14.08.2019
- [42] Unity Technologies, *Transform*, URL: <https://docs.unity3d.com/ScriptReference/Transform.html>, letzter Zugriff: 14.08.2019
- [43] Unity Technologies, *MonoBehaviour*, URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>, letzter Zugriff: 14.08.2019
- [44] Unity Technologies, *MonoBehaviour.Start()*, URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html>, letzter Zugriff: 14.08.2019
- [45] Unity Technologies, *MonoBehaviour.Update()*, URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html>, letzter Zugriff: 14.08.2019
- [46] William T. Reeves, Ricki Blau, *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems* (1985), ACM SIGGRAPH Computer Graphics, Volume 19 Issue 3, Juli 1985, S. 313-322
- [47] The Coding Train, *Coding Challenge #17: Fractal Trees - Space Colonization* (2016), <https://www.youtube.com/watch?v=kKT0v3qhIQY>, letzter Zugriff 22.08.2019
- [48] yoyoyoyosef, Antwort auf *Random Gaussian Variables* (2008), URL: <https://stackoverflow.com/a/218600>, letzter Zugriff: 27.08.2019
- [49] datenwolf, Antwort auf *Calculating Vertex Normals of a mesh [duplicate]* (2013), URL: <https://stackoverflow.com/a/16341131>, letzter Zugriff: 28.08.2019
- [50] Microsoft, *List<T> Class*, <https://docs.microsoft.com/de-de/dotnet/api/system.collections.generic.list-1?view=netframework-4.8>, letzter Zugriff: 27.08.2019
- [51] URL: <https://upload.wikimedia.org/wikipedia/commons/thumb/5/53/Sigmoid-function-2.svg/1280px-Sigmoid-function-2.svg.png>, letzter Zugriff: 20.08.2019
- [52] 3Blue1Brown, *The determinant / Essence of linear algebra, chapter 6* (2016), <https://www.youtube.com/watch?v=Ip3X9L0h2dk>, letzter Zugriff: 27.08.2019
- [53] Persönliches Gespräch mit Rudi Schneider (Informatik-Student an der Technischen Universität Berlin), 26.08.2019
- [54] Rik Heywood, Antwort auf *Closest point to a given point*, URL: <https://stackoverflow.com/questions/1901139/closest-point-to-a-given-point>, letzter Zugriff: 28.07.2019
- [55] Voxel, URL: <https://de.wikipedia.org/wiki/Voxel>, letzter Zugriff: 28.08.2019
- [56] OpenCV, *About*, URL: <https://opencv.org/about/>, letzter Zugriff: 22.08.2019

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Datum, Ort

Unterschrift