HAW HAMBURG

BACHELOR THESIS
Nils Martens

Prozedurale Generierung von Thema-variablen Gebäudegruppen aus Grundrissen mit einer Formgrammatik

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Nils Martens

Prozedurale Generierung von Thema-variablen Gebäudegruppen aus Grundrissen mit einer Formgrammatik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Thomas Lehmann

Eingereicht am: 04.12.2023

Nils Martens

Thema der Arbeit

Prozedurale Generierung von Thema-variablen Gebäudegruppen aus Grundrissen mit einer Formgrammatik

Stichworte

Computer Grafik, 3D, Prozedurale Contentgenerierung, Form Grammatik, Gebäude, Stadt, Grundriss

Kurzzusammenfassung

In dieser Arbeit wird die Möglichkeit der prozeduralen Generierung von Gebäude Gruppen verschiedener Themen auf Basis von Grundrissen untersucht. Dafür wird ein Software Projekt, welches Shape Grammars zur Erzeugung von 3D-Modellen nutzt, konzeptioniert und implementiert. Die Ergebnisse, Qualität, Probleme und das Potenzial dieser Generierung von Stadtmodellen werden aufgezeigt, analysiert und diskutiert.

Nils Martens

Title of Thesis

Procedural Generation of Theme-variable Building Groups from Floor Plans with a Shape Grammar

Keywords

Computer Graphics, 3D, Procedural Content Generation, Shape Grammar, Building, City, Floor Plan

Abstract

In this thesis the possibility of procedural generation of building groups of different themes based on floor plans is investigated. For this purpose a software project that uses shape grammars to generate 3D models is conceptualized and implemented. The results, quality, problems and potential of this generation of city models are shown, analyzed and discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Prozedurale Content Generierung (PCG) ist ein sehr spannendes Feld der Informatik und speziell der Computergrafik. Besonders in Branchen wie der Filmindustrie und der Videospielentwicklung, aber auch der Bildung und Forschung hat sie großes Potenzial. Die Nachfrage nach immersiven und vielfältigen digitalen Inhalten und Umgebungen wächst ständig weiter und PCG kann hier eine gewichtige Rolle spielen. Dieser innovative Ansatz zur Erstellung von Inhalten nutzt Algorithmen, um komplexe und realistische visuelle Elemente zu erzeugen, und bietet eine skalierbare Lösung für die Herausforderungen bei der Gestaltung ausgedehnter virtueller Welten [7].

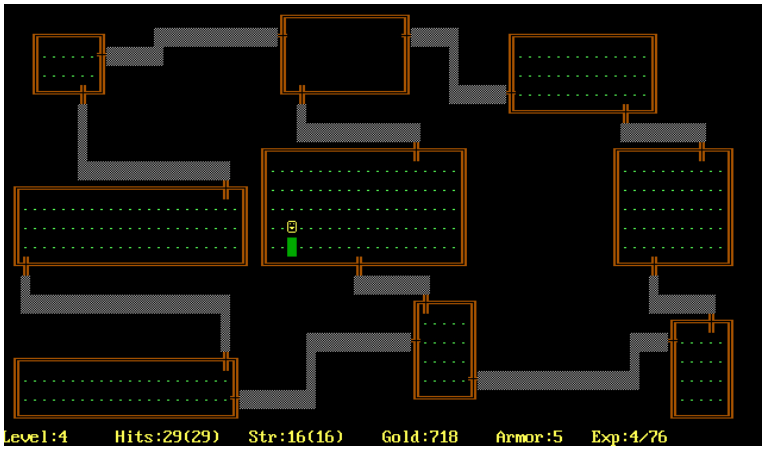


Abbildung 1.1: Künstlerische Nachempfindung eines Screenshots des Spiels *Rogue*[1]

Da die Grenzen der digitalen Kreativität immer weiter ausgedehnt werden, ist die Integration prozeduraler Techniken in die Computergrafik für die Gestaltung der Zukunft interaktiver Erlebnisse von entscheidender Bedeutung. Von der Generierung komplexer Landschaften bis hin zur Gestaltung anpassungsfähiger Spielebenen bieten prozedurale

[1] Quelle: https://commons.wikimedia.org/wiki/File:Rogue_Screen_Shot_CAR.PNG, Urheber: Artoftransformation, Lizenz: CC BY-SA 3.0 DEED, Abrufdatum: 01.12.2023

Techniken ein vielseitiges Werkzeug für die Inhaltserstellung, das es Entwicklern ermöglicht, die Grenzen des visuellen Storytellings und der Benutzerinteraktion zu erweitern. Viele Beispiele für den Einsatz von PCG finden sich in Videospielen. So wurde bereits in dem Spiel *Rogue* (Michael Toy, Glenn Wichman, Ken Arnold und Jon Lane, 1980) prozedural Inhalt erstellt, indem Algorithmen zur Erzeugung von Spielumgebungen wie Dungeons und Höhlen wie in Abbildung 1.1 genutzt wurden. *Left4Dead* (Valve und Turtle Rock Studios, 2008) nutzt prozedurale Techniken, um Begegnungen mit Feinden dynamisch zu erstellen und berechnet dabei sogar den Stress Level des Spielers, während *The Elder Scrolls IV: Oblivion* (Bethesda GameStudios, 2006) und *Elden Ring* (FromSoftware, 2022) die Software SpeedTree nutzen, um prozedural Vegetation zu erzeugen. Große Vorteile der Prozeduralen Content Generierung sind, dass in kurzer Zeit mit wenig Aufwand Inhalt mit großer Variation erzeugt werden kann [9].

In dieser Arbeit wird die Prozedurale Content Generierung im Bereich der Computergrafik und genauer dem Bereich der prozeduralen Erzeugung von Gruppen von Gebäuden untersucht, wobei formale Grammatiken und besonders die sogenannten Shape Grammars in diesem Bereich eine große Rolle spielen [18]. Dabei werden die theoretischen Grundlagen und der aktuelle Stand der Technik näher beleuchtet und es wird ein Prototyp einer Anwendung konzeptioniert und implementiert, welcher die Möglichkeiten untersuchen soll, aus einem bereits vorhandenen Grundriss Gruppen von Gebäuden verschiedener Themen zu generieren.

2 Grundlagen

In diesem Kapitel werden die Grundlagen, auf denen diese Arbeit aufbaut, erklärt.

2.1 Prozedurale Content Generierung

Unter prozeduraler Generierung von Inhalten - im Folgenden PCG genannt - versteht man die Erstellung von Inhalten mithilfe von Algorithmen in einem Computerprogramm oder -system, unter anderem im Zusammenhang mit Videospielen. Das Hauptmerkmal von PCG ist die algorithmische Generierung von Inhalten mit begrenzten oder indirekten Benutzereingaben. Dieser Inhalt kann verschiedene Elemente wie Levels, Karten, Charaktere, Gegenstände, Geschichten, Texturen und mehr umfassen. Im Bereich der Videospiele wird PCG eingesetzt, um Spielinhalte dynamisch zu erstellen und so vielfältige und einzigartige Erlebnisse für die Spieler zu schaffen. Anstatt sich ausschließlich auf vorgefertigte Inhalte zu verlassen, ermöglicht PCG die Erstellung von Spielelementen während des Spiels, was den Wiederspielwert und die Abwechslung in Spielen erhöht. PCG-Methoden können von einfachen Algorithmen bis hin zu komplexen Systemen reichen und werden häufig zur Erstellung ganzer Spielwelten, Levels oder bestimmter Spielelemente eingesetzt. Ziel ist es, den Prozess der Inhaltserstellung zu automatisieren, sodass die Entwickler effizient große Mengen an abwechslungsreichen Inhalten produzieren können, ohne dass eine manuelle, zeitaufwändige Gestaltung erforderlich ist. PCG ist nicht auf Videospiele beschränkt, sondern kann auch in anderen Bereichen eingesetzt werden, zum Beispiel bei der Erstellung von Kunstwerken oder der Musikkomposition. Im Kontext von Spielen ist PCG jedoch ein wertvolles Werkzeug zur Verbesserung des Spielerlebnisses durch die Einführung von Unvorhersehbarkeit und Vielfalt [12].

Für PCG existieren mehrere Ansätze, zu denen unter anderem such-basierte und konstruierende Methoden, Machine Learning und die Benutzung von Grammatiken gehören. Grammatiken werden äußerst erfolgreich für beispielsweise die Generierung von Vegetation in Videospielen eingesetzt. Andere Bereiche, die vom Einsatz von Grammatiken für

PCG profitieren, sind die Filmindustrie, Architektur und die Generierung von Stadtmodellen [3].

Im Folgenden wird sich auf die Methode der Nutzung von Grammatiken konzentriert.

2.2 Formale Grammatik

Eine Grammatik ist eine Gruppe von Regeln, welche eine Sprache beschreibt, die wiederum eine natürliche oder eine Programmiersprache sein kann. Sie wird verwendet, um die Syntax der Sprache mithilfe eines mathematischen, theoretischen Ansatzes zu beschreiben und kommt unter anderem in der Linguistik, Logik und Informatik vor [5].

Eine formale *Grammatik* G ist ein 4-Tupel $G = (N, T, P, s)$ mit folgenden Eigenschaften [5]:

- N: Menge der Nichtterminale (grammatikalische Variablen)
- T: Menge der Terminale (Alphabetzeichen)
- N, T sind nichtleere, endliche und disjunkte Mengen
- P: endliche Menge von Regeln

  $P = \{\alpha \to \beta \mid \alpha \in (N \cup T)^* \backslash T^* \text{ und } \beta \in (N \cup T)^*\}$
- s: *Startsymbol*, wobei $s \in N$

Laut der *Chomsky-Hierarchie* existieren vier Klassen formaler Grammatiken, wobei diese sich in der Gestalt der Regel unterscheiden:

- Typ 0 (unbeschränkt): keine Einschränkung
- Typ 1 (kontextsensitiv): in den Regeln darf genau ein Nichtterminalsymbol auf eine Zeichenfolge abgebildet werden, wobei auf der linken Seite weitere Symbole stehen dürfen
- Typ 2 (kontextfrei): wie Typ 1 und zusätzlich darf hier auf der linken Seite der Regeln nur genau ein Nichtterminalsymbol stehen

- Typ 3 (regulär): wie Typ 2 und zusätzlich darf die rechte Seite der Regeln aus höchstens einem Nichtterminalsymbol bestehen, dem höchstens ein Terminalsymbol folgt (linksregulär) beziehungsweise vorangeht (rechtsregulär)

Eine Grammatik besteht also aus Symbolen, welche entweder terminal oder nichtterminal sind. Die Menge $N \cup T$ wird dabei auch als $Vokabular\ V$ bezeichnet. Nichtterminale werden mithilfe der Regeln auf andere Symbole oder Symbolketten abgebildet, wobei diese Symbole wiederum Terminale oder Nichtterminale sein können. Ausgehend vom Startsymbol können so Wörter erzeugt werden und Wörter, die nur aus Terminalsymbolen bestehen, ergeben die formale Sprache der Grammatik, da Terminalsymbole nicht weiter abgeleitet werden können [5].

Ein Beispiel für eine kontextfreie Grammatik $G = (N, T, P, s)$ könnte folgendermaßen aussehen [5]:

$$N = \{Number, Digit\}$$
$$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$P = \{Number \rightarrow Digit \mid DigitNumber; Digit \rightarrow 0|1|2|3|4|5|6|7|8|9\}$$
$$s = Number$$

Wörter dieser Grammatik sind ganzzahlige Nummern und will man mit dieser Grammatik beispielsweise die Nummer 42 ableiten, wäre die folgende eine mögliche Ableitung:

$$Number \Rightarrow DigitNumber \Rightarrow 4Digit \Rightarrow 42$$

2.3 Computergrafik

Der Begriff der Computergrafik bezieht sich im Bereich der Informatik auf die Erstellung, Darstellung und Bearbeitung von Bildern, 3D-Modellen und Animationen mithilfe von Computern. Sie umfasst eine breite Palette von Techniken und Technologien zur Erzeugung und Darstellung visueller Inhalte, von einfachen 2D-Grafiken bis hin zu komplexen 3D-Animationen. Computergrafik ist ein Thema, welches sich in diversen Gebieten wie Informatik, Mathematik, Physik und Kunst finden lässt. Sie ist die Grundlage für viele Anwendungen und Technologien, die sich auf verschiedene Bereiche und Aspekte unseres

täglichen Lebens auswirken. Sie verbindet Kreativität mit technischem Fachwissen, um visuell überzeugende und interaktive digitale Erlebnisse zu schaffen [10].

Einige Beispiele für Disziplinen in der Computergrafik sind:

- 2D-Grafik: Umfasst die Erstellung und Bearbeitung zweidimensionaler Bilder. Dazu gehören Aufgaben wie das Zeichnen, die Bildbearbeitung und das Rendern von Grafiken für Benutzeroberflächen.
- 3D-Grafiken: Handelt von der Erstellung und Bearbeitung dreidimensionaler Objekte und Szenen. 3D-Grafiken werden häufig in Anwendungen wie Videospielen, Simulationen, virtueller Realität und computergestütztem Design eingesetzt.
- Computeranimation: Konzentriert sich auf die Erzeugung von dynamischen, bewegten Bildern. Animationstechniken werden in Filmen, Videospielen und Simulationen eingesetzt, um lebensechte und ansprechende visuelle Erlebnisse zu schaffen.
- Rendering: Der Prozess der Erzeugung realistischer Bilder aus 3D-Modellen. Dazu gehören Techniken wie Schattierung, Beleuchtung und Texture Mapping, um die Interaktion von Licht und Oberflächen zu simulieren.

So werden Techniken der 3D-Grafik und Computeranimation häufig in Bereichen wie der Grafik für Videospiele und Computer Generated Imagery (CGI) für Filme eingesetzt.

2.3.1 Dreiecksnetze

In der Computergrafik werden zur Modellierung und Darstellung von Modellen häufig Mengen von Polygonen verwendet, welche die Oberflächen der Modelle abbilden. Polygone werden verwendet, da diese einfache geometrische Objekte darstellen, die leicht zu berechnen sind. Der einfachste Vertreter dieser Polygone stellt dabei das Dreieck dar, weshalb oft Dreiecksnetze (englisch: Triangle Mesh) verwendet werden, siehe Abbildung 2.1. Das Ziel ist es, Modelle möglichst einfach darzustellen, was oftmals mit tatsächlich sehr wenigen Polygonen gelingt. Ein Polygon eines Polygonnetzes besteht aus Punkten (Vertices), Kanten und Flächen (Facetten). Im Beispiel eines Dreiecksnetzes besteht ein Polygon aus drei Vertices, welche verbunden und zu den Kanten werden, welche die Facetten des Polygons einschließen. Die einzelnen Polygone eines Netzes sind an ihren Vertices verbunden und ergeben so eine Struktur, welche die Oberfläche des darzustellenden Modells bildet, siehe Abbildung 2.2.

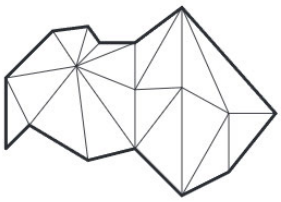


Abbildung 2.1: Dreiecksnetz [10]

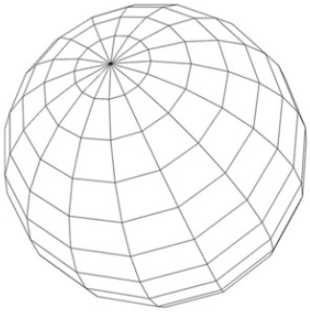


Abbildung 2.2: Polygonnetz, welches eine Kugel darstellt [10]

2.3.2 Shape Grammar

Shape Grammars, übersetzt Form-Grammatiken, sind ein formales System zur Generierung und Beschreibung von Formen und Strukturen. Sie folgen einem regelbasierten Ansatz, wobei eine Menge an Regeln verwendet wird, Beziehungen und Transformationen zwischen geometrischen Formen zu definieren, was die Generierung komplexer Modelle ermöglicht. Shape Grammars wurden von George Stiny und James Gips eingeführt [16] und finden Anwendung in der Architektur und der Computergrafik unter anderem für Videospiele und Filme, wo sie zur Generierung von 2D- und 3D-Modellen verwendet werden. Beispiele hierfür sind realistische Straßenpläne, Fassaden oder Innenräume von Gebäuden [8, 11, 19].

Die Definition einer Shape Grammar ähnelt der Definition formaler Grammatiken, welche in Kapitel 2.2 beschrieben wurden. Eine Shape Grammar $G = (S, L, R, I)$ besteht somit aus:

- S: endliche Menge von Formen
- L: endliche Menge von Symbolen

- R: endliche Menge von Formregeln mit $R = \{\alpha \rightarrow \beta \mid \alpha \in (S, L)^+ \text{ und } \beta \in (S, L)^*\}$
- I: Initial-Form in $(S, L)^+$

Die Initial-Form und Formen in den Regeln werden labelled Shapes, also beschriftete Formen genannt und setzen sich aus einer Form aus S und einem Symbol aus L zusammen, wobei Formen mit leerem Symbol Terminale, also abschließende Formen, die nicht weiter abgeleitet werden können, sind. Der Begriff Form wird von Stiny wie folgt definiert:

> "A shape is a limited arrangement of straight lines defined in a cartesian coordinate system with real axes and an associated euclidean metric." [15]

Eine Form sei also eine begrenzte Anzahl gerader Linien in einem kartesischen Koordinatensystem mit reellen Achsen und einer euklidischen Metrik. Mit den Formregeln können nun beschriftete Formen in andere beschriftete Formen abgeleitet werden, wobei jede Ableitung eine neue Form ergibt. Begonnen wird die Generierung einer Form immer mit der Initial-Form und der Startregel, also die Regel, welche die Initial-Form ableiten kann. Dann werden die Formregeln rekursiv auf die neu entstandene Form angewendet, bis eine Terminierungs-Regel angewandt wird, also eine Regel welche die aktuelle Form in eine terminale Form ableitet. Regeln, die weder startend noch terminierend sind, werden Transformationsregeln genannt und eine Shape Grammar benötigt mindestens eine Start- und eine Terminierungs-Regel, damit der Prozess gestartet und beendet werden kann. Da eine Shape Grammar nicht deterministisch ist, kann es vorkommen, dass auf eine Form mehrere Regeln anwendbar sind. Damit wird je nach Implementierung unterschiedlich umgegangen, sodass beispielsweise zufällig eine Regel ausgewählt und angewandt wird [14]. Die Sprache einer Shape Grammar bilden damit alle Formen s, die mit den Formregeln von der initialen Form I abgeleitet wurden und die kein zugehöriges Symbol besitzen. Alle diese Formen bestehen wiederum aus Formen aus S [15].

Abbildung 2.3 zeigt ein Beispiel für eine einfache Shape Grammar mit zwei Regeln (a) und einer Initialform (b), wobei hier das Quadrat eine Form und der Punkt am Rand des Quadrats ein Symbol darstellen. Es ist zu erkennen, dass Regel 1 eine Form erzeugt, die weiter abgeleitet werden kann, während Regel 2 eine Form erzeugt, die nicht weiter abgeleitet werden kann. Abbildung 2.4 zeigt im oberen Teil eine der möglichen Ableitungswege dieser Shape Grammar und somit eine mögliche Generierung einer Form. Im unteren Teil zeigt die Abbildung mehrere Formen aus der Sprache, welche die Shape Grammar definiert, das heißt Formen, die mit den Regeln der Shape Grammar aus der Initial-Form generiert werden können [15].

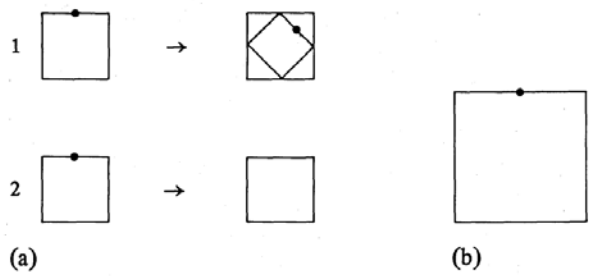


Abbildung 2.3: Beispiel einer einfachen Shape Grammar [15]

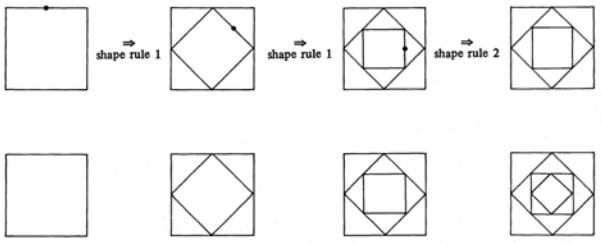


Abbildung 2.4: Eine mögliche Generierung einer Form (oben), mehrere Formen [15]

2.4 Stand der Technik

In diesem Abschnitt wird ein Blick auf den aktuellen Stand der Technik geworfen.

2.4.1 Procedural Modeling of Buildings

Shape Grammars werden aktiv für das Modellieren von Gebäuden eingesetzt. *CGA Shape* ist eine Implementierung einer Shape Grammar, die von Pascal Müller, Peter Wonka et al. entwickelt wurde und Modelle von Gebäuden mit hohem Detailgrad und visueller Qualität erzeugen kann, wie beispielhaft in Abbildung 2.5 zu sehen. Dabei arbeitet *CGA Shape* effizient und mit geringen Kosten, sodass es gut in Videospielen und Filmen eingesetzt werden kann [11].

Um mit *CGA Shape* Gebäude zu generieren, erzeugt diese iterativ immer mehr Details des Modells. So wird zunächst ein grober Körper des Gebäudes erzeugt, an dem nach und nach Fassaden und weitere Details wie Fenster und Türen erzeugt werden, was den Vorteil hat, dass prozedural eine große Vielfalt an Variationen generiert werden kann. Regeln, die in *CGA Shape* verwendet werden, sind [11]:

Abbildung 2.5: Beispiele von mit *CGA Shape* erzeugten Gebäuden [11]

- Scope Regeln: diese beinhalten Translation, Rotation und Skalierung von Formen
- Split: teilt eine Form entlang einer zu wählenden Achse in festgelegte Abschnitte gewählter Größen
- Repeat: teilt eine Form so oft wie möglich entlang einer Achse
- Component Split: teilt dreidimensionale Objekte in ihre Komponenten auf, wie beispielsweise die Seiten eines Würfels

Diese Regeln arbeiten auf Formen und deren Scope, welches sich aus einem Punkt P, den drei Achsen X, Y und Z, sowie einem Größenvektor S zusammensetzt und damit eine Box definiert, die die Position und Ausdehnung der Form im Raum darstellt, wie in Abbildung 2.6 dargestellt [11].

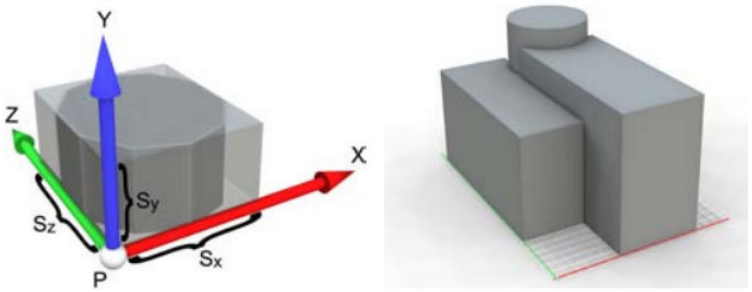


Abbildung 2.6: Links: Scope einer Form, Rechts: grober Körper eines Gebäudes, bestehend aus drei Formen [11]

Wie in Abschnitt 2.3.2 erläutert, bestehen Formen auch in *CGA Shape* aus einer Form und einem dazugehörigen Symbol, wobei es Terminal- und Non-Terminal-Symbole gibt. Abbildung 2.7 zeigt eine beispielhafte Anwendung der benannten Regeln, wobei zu erkennen ist, dass jede Form ein zugehöriges Symbol hat. So wurden die Stockwerke mit $floor$ und die Fenster mit A und B benannt [11].

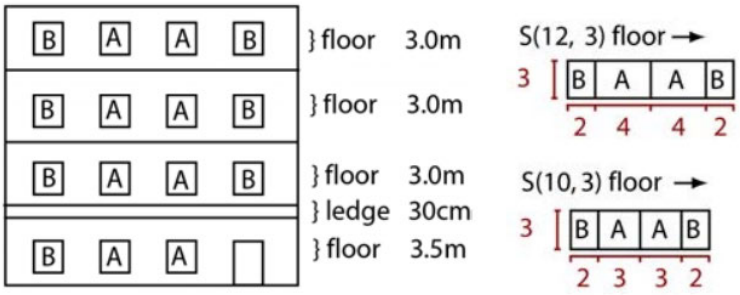


Abbildung 2.7: Links: Design einer Fassade mit Formen und deren Symbolen, Rechts: Split Regel, auf die oberen Stockwerke angewandt [11]

Die Implementierung von *CGA Shape* ist außerdem in das *CityEngine* Framework von Parish und Müller [19] integriert worden, wodurch Modelle von ganzen Städten teilweise mithilfe von GIS-Dateien generiert werden können, wie zum Beispiel ein Modell des alten Pompeii in Abbildung 2.8.

Abbildung 2.8: Modell von Pompeii [11]

2.4.2 Rome Reborn

Rome Reborn 2.0 ist ein weiteres Beispiel für die prozedurale Generierung von Städten. Dieses Projekt nutzt *CityEngine* mit *CGA Shape* und zielt darauf ab, die antike

Stadt Rom so nachzubilden, wie sie zu ihrer Blütezeit ausgesehen haben könnte. Das Projekt ist eine laufende Zusammenarbeit zwischen verschiedenen Institutionen und Wissenschaftlern aus den Bereichen Archäologie, Geschichte und Informatik. Ziel ist es, eine umfassende und genaue Darstellung des antiken Roms zu schaffen, die es den Benutzern ermöglicht, die Stadt in ihren verschiedenen historischen Phasen zu erkunden. Dazu gehören die Architektur, Straßen, Gebäude und andere Elemente der Stadt, wie sie in den verschiedenen Zeitabschnitten existierten. Bei dem Projekt geht es nicht nur um die Erstellung eines visuell beeindruckenden 3D-Modells, sondern auch um die Einbeziehung wissenschaftlicher Forschung und archäologischer Daten, um historische Genauigkeit zu gewährleisten. Es dient als Lehrmittel und bietet Studenten, Forschern und der Öffentlichkeit eine einzigartige Gelegenheit, die Geschichte und die städtische Entwicklung des antiken Roms virtuell zu erleben und zu erlernen [6]. Abbildung 2.9 zeigt einen Ausschnitt des generierten *Rome Reborn 2.0* Modells.

Abbildung 2.9: *Rome Reborn 2.0* Modell mit einem Modell des Circus Maximus in der Mitte [6]

3 Konzept

Die Motivation und Ziele dieser Arbeit wurden in Kapitel 1 vorgestellt. Um die Punkte aus dem Kapitel zu untersuchen, wird für diese Arbeit eine Software entwickelt, welche den Ansprüchen zur Genüge entspricht, sodass ein aussagekräftiges Ergebnis erreicht werden kann. In diesem Kapitel sollen dafür die Anforderungen an die Software erläutert und eine Herangehensweise zum Erreichen dieser erarbeitet werden.

3.1 Anforderungen

Allgemein wird gefordert, dass jede korrekte Eingabe zu einem korrekten Ergebnis führt. Das bedeutet, dass eine Eingabe, welche Informationen zur Lage von Gebäuden enthält zu einem 3D-Modell einer Gebäudegruppe als Ausgabe führen soll. Dies wird unten weiter spezifiziert.

3.1.1 Programm-Eingabe

Die Eingabe der Anwendung ist eine Datei, welche mindestens die Informationen zur Lage der Gebäude enthalten muss. Dabei soll jede beliebige Datei, welche das festgelegte Format besitzt, zu einem korrekten Ergebnis führen. Für den Rahmen dieser Arbeit, wurden zwei mögliche Formate festgelegt, welche beide auf JSON-Formaten basieren.

Das erste ist ein einfaches JSON-Format, welches ein Objekt beinhalten muss, welches eine Variable id mit dem Wert buildings sowie eine Variable coordinates beinhaltet. Die Variable coordinates hat als Wert ein Array, welches wiederum ein oder mehrere Arrays enthält. Diese Arrays enthalten Werte-Tupel, die Punkte eines Polygons darstellen. Jedes Array innerhalb des Arrays coordinates stellt mit seinem Polygon das Grundstück eines Gebäudes dar.

Das zweite Eingabe-Format ist ein GeoJSON-Format, welches die Variable features mit einem Array von Objekten als Wert beinhaltet. Jedes Objekt dieses Arrays stellt dabei ein Gebäude dar und enthält wiederum ein Objekt geometry. Das Objekt geometry enthält ein Array coordinates, welches wie bei dem ersten Format die Punkte eines Polygons darstellt. Dieses Polygon stellt das Grundstück des Gebäudes dar. Dabei müssen die Punkte der Polygone die Koordinaten der Gebäude im Stile des WGS 84 / Pseudo-Mercator Koordinatenbezugsystems abbilden.

3.1.2 Programm-Ausgabe und Gebäude-Themen

Die Ausgabe der Anwendung ist ein 3D-Modell einer Gruppe von Gebäuden wie einem Dorf oder einer Stadt, wobei das 3D-Modell als solches visuell erkennbar sein soll. Zwei nacheinander erzeugte Gebäudegruppen sollen sich bei gleicher Eingabe unterscheiden.

Thema	Gebäudehöhe	Dachtypen	(Fassaden-)Details
`Cyberpunk`	Hoch bis sehr hoch (Wolkenkratzer), 50 bis 150 Meter	Flachdächer; Raum für Dachzugang	Große, fast voll verglaste Fensterfassaden; Schwarz und Neonfarben
`Mittelalter Fantasy`	Flach, 6 bis 10 Meter; hohe Türme, 20 bis 30 Meter	Satteldächer mit wenigen, kleinen Gauben	Fenster und Fachwerk; weiße oder bunte Hauswände, graue Steinwände, braune Holzdetails
`Antikes Griechenland`	sehr Flach, 3 bis 6 Meter; höhere Tempel, 6 bis 10 Meter	Flach-, Sattel- und Walmdächer; Tempel mit Satteldächern	Fenster ohne Glas, Tempel mit Stufen und Säulen; Sandsteinwände, Dach aus Terrakotta-Ziegeln

Tabelle 3.1: Gebäudecharakteristika nach Thema

Es wurde entschieden, im Rahmen dieser Arbeit drei unterschiedliche Themen oder Stile, in denen die Gebäudegruppen generiert werden können, zu realisieren. Diese Themen sind *Cyberpunk*, *Mittelalter Fantasy* und *Antikes Griechenland*. Jedes generierte 3D-Modell eines bestimmten Themas soll sich von einem generierten 3D-Modell eines der zwei anderen

Themen visuell deutlich unterscheiden. Jedes Modell soll weiterhin eindeutig dem gewählten Thema zuzuordnen sein. Dafür sind die festgelegten Charakteristika für jeden Stil einzuhalten. Die Charakteristika wurden für jedes Thema so festgelegt, wie in Tabelle 3.1 zu sehen.

Cyberpunk

Modelle des Themas Cyberpunk sollen Gebäude im Zentrum einer Stadt aus dem dystopischen Science-Fiction-Genre Cyberpunk darstellen, wobei die Inspiration hierfür aus den Bereichen Kunst, Literatur und Medien kommt. So sind die Gebäude inspiriert von Werken wie den Spielen *Cyberpunk 2077* (CD Projekt RED, 2020) und *Deus Ex: Mankind Divided* (Eidos Montreal, 2016), Filmen wie *Blade Runner* (1982) und *Total Recall* (1990), Romanen wie *Neuromancer* (William Gibson, 1984) sowie diversen Kunstwerken. Abbildung 3.1 zeigt zwei dieser Inspirationen.

Die Gebäude sollen den Eindruck von hohen Wolkenkratzern machen, dessen Fassaden zu großen Teilen aus Fenstern bestehen und die sowohl in dunklen als auch Neon-Farben gehalten sind. Wie bei Hochhäusern üblich, haben diese Gebäude ausschließlich Flachdächer, auf denen in manchen Fällen ein kleinerer Raum aufgesetzt ist, welcher ein Zugang zum Dach darstellt. Auf Bodenhöhe können diese Gebäude eine größere Grundfläche als höhere Stockwerke haben und der Eingang ist entweder eine Tür oder ein Raum mit Tür, welcher aus dem Gebäude hervortritt. Die Form des Gebäudes kann aus einer Form bestehen, welche von unten bis oben den gleichen Grundriss und die gleiche Höhe hat, oder aus einer geteilten Form, wobei der Grundriss in der Mitte und manchmal schräg geteilt wird. Diese zwei Teile mancher Gebäude können verschieden große Grundflächen und unterschiedliche Höhen haben.

Mittelalter Fantasy

Die Modelle des Mittelalter Fantasy Themas sind, wie der Name schon sagt, von der Architektur des Mittelalters und der Architektur in Fantasy Settings inspiriert, welche in Abbildung 3.2 beispielhaft gezeigt werden. Dabei wurde hauptsächlich die Architektur von Fachwerkgebäuden aus dem mittelalterlichen Deutschland aber auch jene von mittelalterlichen Wehr- und Wohntürmen herangezogen und so vermischt, dass es eine Siedlung eines mittelalterlichen Fantasy Settings darstellen könnte. Denn die für dieses

(a) Concept Art für die Stadt *Night City* aus *Cyberpunk 2077*

(b) Mit dem KI-Modell *Dall-E 3* generiertes Bild einer Cyberpunk Stadt, eingegebener Text: "cyberpunk city"

Abbildung 3.1: Inspirationen für das Thema Cyberpunk[1]

Thema generierten Gebäude sollen Wohnhäuser und vereinzelte Türme innerhalb eines Dorfes oder einer Stadt des beschriebenen Settings darstellen.

Dafür werden einerseits Fachwerkhäuser generiert, welche teilweise ein Erdgeschoss mit kleinerer Grundfläche haben, dessen Fassaden kein Fachwerk aufweisen, und andererseits Gebäude ohne Fachwerk, wobei die Grundrisse aller dieser Gebäude rechteckig gehalten werden, da komplexe Grundrisse nicht dem Setting entsprechen. Die Fassaden ohne Fachwerk stellen dabei Steinwände dar, sodass Gebäude ohne Fachwerk komplett aus Stein

[1](a) Quelle: Cyberpunk 2077, CProjekt RED, 2020
(b) Quelle: Image Creator von Microsoft Designer, URL: https://www.bing.com/create, Abrufdatum: 01.12.2023

bestehen. Diese Gebäude ohne Fachwerk besitzen weiterhin keine Fenster und stellen Lagerhäuser, Waffenkammern oder ähnliches dar. Die Dächer dieser Gebäude sind Satteldächer und weisen kleine Gauben mit Fenstern auf und die Fachwerkwände sind traditionell im weißen Farbton des Kalkputzes gehalten oder mit einer Farbe bemalt [17].

(a) Fachwerkhäuser in Quedlinburg im Harz (Quelle: eigene Bildaufnahme)

(b) Screenshot von Fachwerkhäusern im Spiel *The Elder Scrolls Online*

Abbildung 3.2: Inspirationen für das Thema Mittelalter Fantasy[2]

Neben diesen Gebäuden gibt es die Türme, welche ein vielfaches höher als die anderen Häuser sind. Es gibt Wehrtürme, deren Spitzen mit einer begehbaren Plattform und Zinnen ausgestattet sind, sowie Wohntürme, an deren Spitzen ein in der Grundfläche vergrößerter Wohnraum mit Fachwerkwänden und Walmdach sitzt. Die Wände aller Türme sind aus Stein und mit wenigen, sehr kleinen Fenstern ausgestattet, welche Schießscharten darstellen. Im unteren Bereich der Türme befinden sich keine Fenster, um sie bei Angriffen sicherer zu machen. Abbildung 3.3 zeigt Inspirationen für die Türme.

[2](b) Quelle: The Elder Scrolls Online, ZeniMax Online Studios, 2014, https://www.elderscrollsonline.com/de/media/category/screenshots/, Abrufdatum: 01.12.2023

(a) Das Rödertor in Rothenburg ob der Tauber

(b) Hauptturm der Burg in Radicofani in Italien

Abbildung 3.3: Inspirationen für die Türme des Themas Mittelalter Fantasy[3]

Antikes Griechenland

Für das Thema Antikes Griechenland sollen zwei Arten Gebäude generiert werden. Wohnhäuser mit rechteckigem Grundriss, welche maximal zweistöckig sind, Fenster ohne Glas aufweisen und dessen Dächer entweder Flach-, Sattel- oder Walmdächer sein können. Die

[3](a) Quelle: https://de.m.wikipedia.org/wiki/Datei:Rödertor_Rothenburg_ob_der_Tauber_20200906_001.jpg, Urheber: Tilman2007, Lizenz: CC BY-SA 4.0 DEED, Abrufdatum: 01.12.2023
(b) Quelle: https://commons.wikimedia.org/wiki/File:RadicofaniCastelloTower.JPG, Urheber: LigaDue, Lizenz: CC BY-SA 3.0 DEED, Abrufdatum: 01.12.2023

zweite Art Gebäude sind Tempel, welche außen herum Stufen aufweisen, die ins Innere führen, sowie Säulen, welche ein Satteldach tragen. Die Farben der Gebäude werden durch die Baumaterialien bestimmt, welche aus Sandstein für die Wände und Terrakotta für die Dachziegel bestehen [2]. Abbildung 3.4 zeigt als eine Inspiration für die Gebäude dieses Themas einen Screenshot aus dem Spiel *Assassin's Creed Odyssey* (Ubisoft Quebec, 2018), in dem ein hoher Grad an historischer Authentizität erreicht wurde [13].

Abbildung 3.4: Screenshot der Stadt Athen im Spiel *Assassin's Creed Odyssey*[4]

3.1.3 Nicht-funktionale Anforderungen

Neben den oben genannten funktionalen Anforderungen an die Software dieser Arbeit gibt es auch nicht-funktionale Anforderungen, welche hier kurz erläutert werden.

- Leistung und Effizienz: Die Generierung der 3D-Modelle soll schnell und ohne große Ressourcenauslastung stattfinden. Es ist denkbar, dass die Generierung für bestimmte Anwendungen zur Laufzeit geschieht, sodass sie möglichst unbemerkt ablaufen soll.
- Benutzbarkeit: Die Anwendung der Software soll einfach zu verstehen und erlernen sein. Die Benutzeroberfläche soll dazu selbsterklärend sein.
- Korrektheit: Die Ergebnisse der Anwendung sollen fehlerfrei sein. Es sollen also korrekte 3D-Modelle erzeugt werden.

[4]Quelle: Assassin's Creed Odyssey, Ubisoft, 2018

- Änderbarkeit: Das Program soll änderbar und erweiterbar sein. Das heißt, es soll möglich sein ohne allzu großen Aufwand neue Funktionen hinzuzufügen oder bereits bestehende anzupassen.
- Skalierbarkeit: Es soll je nach zur Verfügung stehender Hardware möglich sein, 3D-Modelle mit einer größeren Anzahl an Gebäuden oder mit größeren oder detaillierteren Gebäuden beziehungsweise das Gegenteil zu erzeugen.

3.2 Entwicklungsansatz

Für diese Arbeit wurde der Ansatz der prozeduralen Generierung mit Grammatiken gewählt. Genauer gesagt wird hier der Ansatz der Shape Grammars genutzt, welcher in Abschnitt 2.3.2 erläutert wurde. Das Ziel der Entwicklung ist, Gebäude generieren zu können, die in einer Gruppe erzeugter Gebäude alle im Abschnitt 3.1.2 genannten Punkte einhalten. Dazu ist jeweils eine geeignete Grammatik für die Themen Cyberpunk, Mittelalter Fantasy und US-amerikanische Kleinstadt zu entwickeln.

Weiterhin ist es notwendig, eine Eingabe-Datei, welche die Anforderungen aus dem Abschnitt 3.1.1 einhält, korrekt zu verarbeiten. Es gilt, das Ergebnis dieser Verarbeitung auf eine geeignete Weise mit den oben erwähnten Grammatiken zu verbinden, sodass die gewünschten 3D-Modelle erzeugt werden können.

3.2.1 Eingabe Verarbeitung

Damit mithilfe der Grammatiken ein Gebäude generiert werden kann, wird ein sogenanntes Lot benötigt. Dieses Lot stellt den Grundriss oder das Grundstück eines Gebäudes dar und ist die Initial-Form einer Shape Grammar. Eine Eingabe-Datei enthält, wie in Abschnitt 3.1.1 beschrieben, Informationen zu einem oder mehreren Lots. Somit ist der Ansatz zur Verarbeitung einer Eingabe-Datei, diese zu analysieren, die Informationen aller enthaltenen Lots zu extrahieren und in einem geeigneten Format auszugeben.

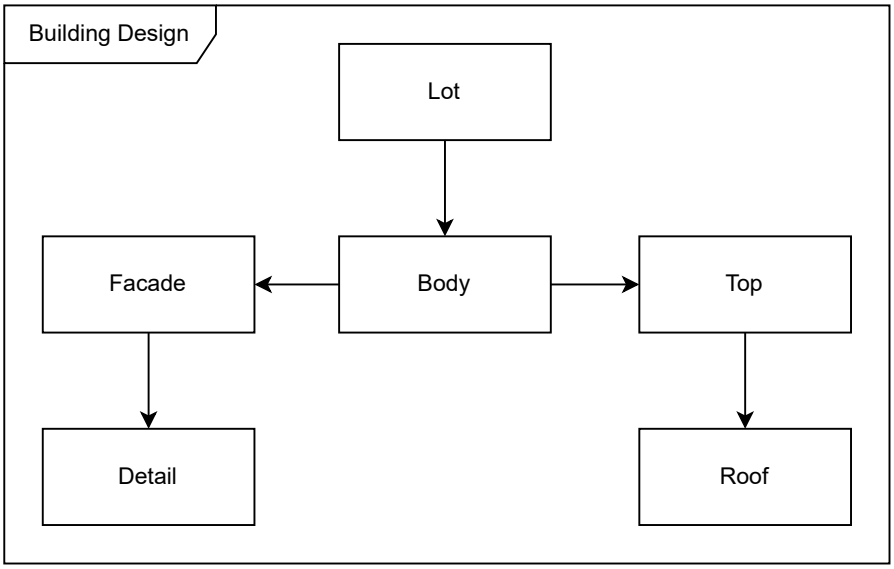


Abbildung 3.5: Allgemeiner Ablauf des Gebäudedesigns

3.2.2 Gebäude Entwurf

Der Entwurf der Gebäude hängt stark von den einzelnen Themen ab, folgt jedoch immer grob einem ähnlichen Ablauf, welcher in Abbildung 3.5 dargestellt ist. Wie in Abschnitt 3.2.1 beschrieben, basiert jedes Gebäude auf einem Lot. Ein Lot ist ein Polygon, welches zunächst nach oben erweitert und so zu einem Prisma wird, welches den Grundkörper eines Gebäudes darstellt. Dieses Prisma wird Body genannt. Der Body wird in Facades also die Seiten des Gebäudes und Top also das obere Teil aufgeteilt. Die Facades erhalten Details wie Fenster und Türen und Top wird zu einem Roof also einem Dach.

Jeder dieser Schritte wird von den Regeln der Grammatiken umgesetzt und dabei von ihnen beeinflusst. Das heißt, in den Grammatiken werden Größen für die Gebäudehöhe, Details, Dächer und andere festgelegt. Dabei können Größen sich auch in bestimmten Intervallen bewegen. Weiterhin kann es bei Grammatiken an bestimmten Stellen Zwischenschritte geben, um bestimmte Teile des Gebäudes anzupassen oder verschiedene Varianten zu erhalten. Zu diesem Zweck sind bestimmte Schritte zusätzlich mit Wahrscheinlichkeiten versehen. Die genannten Punkte dienen dazu, die in Abschnitt 3.1.2 festgelegten Charakteristika einzuhalten und somit die Themen eindeutig erkennbar und voneinander unterscheidbar zu machen. Sie dienen zusätzlich dazu, innerhalb der einzelnen Themen die geforderte Variation zwischen einzelnen Gebäuden und jedem generierten 3D-Modell zu erhalten.

4 Implementierung

In diesem Kapitel wird die Umsetzung des in Kapitel 3 vorgestellten Konzeptes beschrieben und erklärt. Die Implementierung dieser Bachelor Arbeit basiert auf dem Projekt *cgashape*, welches von Herrn Prof. Dr. P. Jenke geleitet wird und eine Implementierung einer Shape Grammar ist. Das Softwareprojekt dieser Arbeit wird im Folgenden *CityBuilder* genannt, nicht zu verwechseln mit der gleichnamigen Klasse.

Für die Umsetzung des Konzeptes wurden mehrere Teilimplementierungen durchgeführt, von denen manche in diesem Kapitel näher erläutert werden. Die durchgeführten Implementierungen und Anpassungen vorhandenen Codes beinhalten:

- Drei Grammatiken für die Generierung von Gebäuden unterschiedlicher Themen.
- Die Klasse *GeoParser* zur Generierung von Axiom-Texten aus JSON- und GEOJSON-Dateien.
- Neue Shape Operationen *Fachwerk* für die Darstellung von Fachwerk Details, *Rectify* um ein Polygon in ein Viereck zu transformieren und *Square* um ein Polygon in ein Quadrat zu transformieren.
- Erweiterungen der Shape Operationen *Split* und *RepeatedSplit* für Polygone die nicht gerade oder im Ursprung ihres Koordinatensystems liegen.
- Erweiterung der Shape Operation *Detail*, um ein Fenster ohne Glas darstellen zu können.
- Erweiterung der Klasse *MeshGenerator*, sodass diese mit den Klassen dieses Projektes kompatibel wird und für die erweiterte Darstellung von Farben in den resultierenden 3D-Modellen.
- Die Klassen *CityBuilder*, *CityEditor*, *CityModel*, *CityParams*, *CityScene* und *CityViewer* auf Basis der entsprechenden Klassen des *cgashape*.

Relevante Begriffe, die in diesem Kapitel genannt aber nicht näher beschrieben werden, sollen hier erklärt werden.

- Das Axiom ist ein *PolygonShape*, welches das Startsymbol für eine Grammatik darstellt.
- Ein Shape ist allgemein eine Form. Verschiedene Arten wie *PolygonShape* und *PrismShape* werden im *cgashape* Projekt implementiert und im CityBuilder Projekt genutzt. Im Folgenden werden Shape und Form, *PolygonShape* und Polygon sowie *PrismShape* und Prisma jeweils synonym verwendet.
- Shapeoperationen sind Operationen auf Formen, welche innerhalb der Grammatiken mithilfe von Regeln angewandt werden können. Die Operationen können Shapes verändern oder durch neue Formen ersetzen.

4.1 Architektur und Abhängigkeiten

Die Architektur des Softwareprojektes dieser Bachelorarbeit basiert auf der Architektur des *cgashape* Projekts. Abbildung 4.1 zeigt die im Rahmen dieser Arbeit genutzten Komponenten und Packages. Die Komponente *cgaShapeProject* enthält das Package *cgaShape*, welches den bereits vorhandenen Code des Projekts *cgashape* beinhaltet, und das Package des Projekts *CityBuilder*, welches den während dieser Arbeit implementierten Code enthält. Wie in der Abbildung zu erkennen, nutzt *CityBuilder* *cgaShape*, was hier bedeutet, dass verschiedene Klassen des *cgaShape* genutzt werden oder von diesen geerbt wird. Es wird auch ersichtlich, dass die Komponente *jMonkeyEngine* als eine zweite Komponente nutzt. *jMonkeyEngine* ist eine Open-Source Spiele-Engine für die Entwicklung in Java und dabei leicht zu bedienen. In diesem Projekt wird diese Engine zur Darstellung der resultierenden 3D-Modelle verwendet.

Abbildung 4.2 zeigt die Architektur des Packages *CityBuilder*. Dabei sind vor allem die neu implementierten Klassen *CityEditor*, *CityModel*, *CityParams*, *CityScene*, *CityViewer* und *GeoParser* relevant. Zusätzlich werden im Klassendiagramm die Klassen *Grammar*, *MeshGenerator* und *TriangleMesh* aus dem *cgaShape* dargestellt, da diese zentrale Rollen spielen. Für die Instanziierung der einzelnen Klassen und die Ausführung des Programms ist die hier nicht gezeigte Klasse *CityBuilder* zuständig.

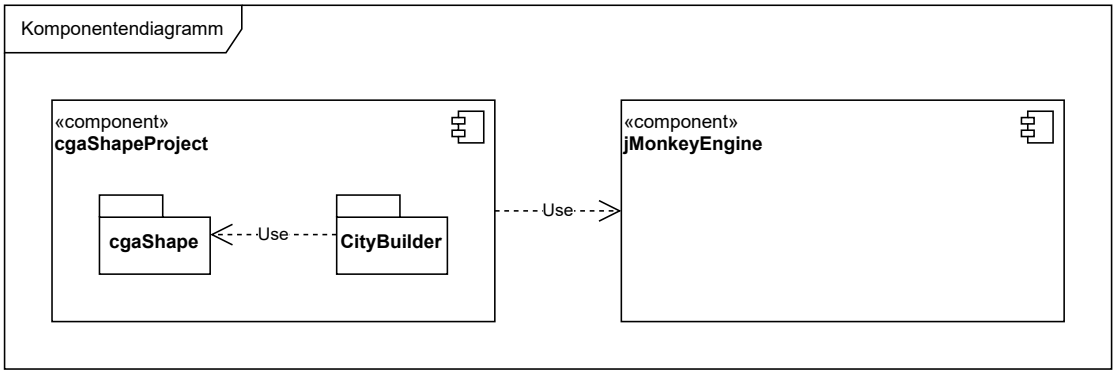


Abbildung 4.1: Komponentendiagramm des Software Projekts der Bachelor Arbeit

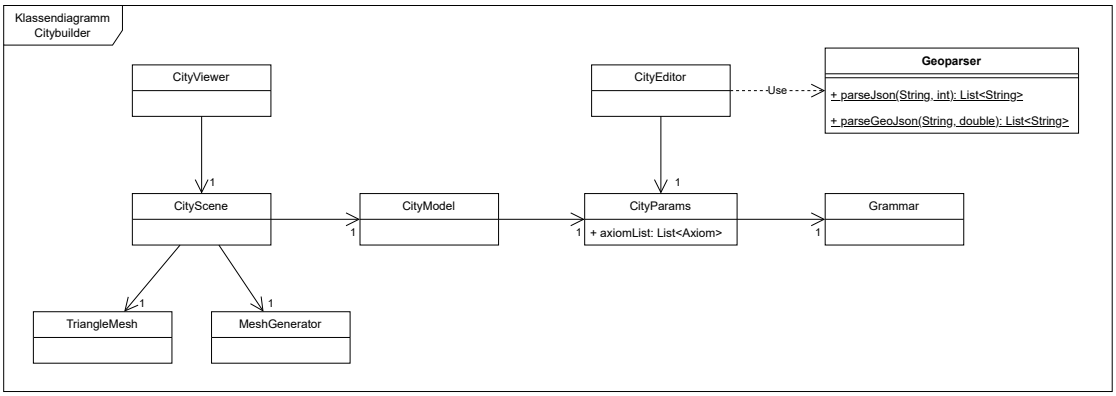


Abbildung 4.2: Klassendiagramm des Software Projekts der Bachelor Arbeit

4.2 Klassen

Im Folgenden werden die implementierten und genutzten Klassen und deren Funktion erklärt.

CityParams

Die Klasse *CityParams* enthält alle Informationen und Objekte, die als Eingabe der Anwendung dienen und somit für die Generierung des 3D Modells benötigt werden. Dazu gehört eine Liste von Axiomen, welche auf dem vom Benutzer eingegebenen Grundriss basiert und die Grundstücke der zu generierenden Gebäude darstellt. Jedes Axiom stellt somit ein Startsymbol für die genutzte Grammatik dar, welche als ein Grammar Objekt ebenfalls in den *CityParams* enthalten ist.

CityModel

Mit der Klasse *CityModel* werden die generierten Formen repräsentiert. Dazu hat das *CityModel* Objekt ein *CityParams* Objekt, aus welchem die Liste von Axiomen und die Grammatik ausgelesen werden. Für jedes Axiom wird mithilfe eines Evaluators aus dem *cgashape* Projekt die Grammatik in einen Shape Tree evaluiert und an die root Shape angehängt, wobei der erste evaluierte Shape Tree die root Shape wird.

CityScene

Die Klasse *CityScene* bekommt als Repräsentation der generierten Form ein *CityModel* Objekt. Sie enthält jeweils ein *MeshGenerator* und *TriangleMesh* Objekt, welche im *cgashape* Projekt implementiert sind und weiter unten näher erläutert werden. Der *MeshGenerator* wird genutzt, um aus der Form im *CityModel* Objekt ein *TriangleMesh* zu erzeugen.

CityEditor

In der Klasse *CityEditor* wird das Fenster der Benutzeroberfläche definiert, in dem der Anwender eine Eingabe Datei im Json- oder GeoJson-Format wählen kann, welche hier Layout genannt wird. Wird ein Layout gewählt, nutzt das *CityEditor* Objekt Methoden der Klasse *GeoParser*, um die Json Datei zu parsen und eine String Liste zu erzeugen, welche die Axiome der zu generierenden Formen enthält. Weiterhin kann der Anwender in diesem Fenster eine Grammatik - hier Grammar genannt - wählen, bearbeiten und speichern, eine maximale Anzahl an generierten Gebäuden - Maximum number of buildings - angeben und mit einem Klick auf den Button Generate die Erzeugung eines 3D-Modells starten. Bei der Instanziierung des *CityEditors* wird das *CityParams* Objekt übergeben, sodass Änderungen an den Eingaben durch den Benutzer an das Backend übergeben werden können.

CityViewer

Für die Anzeige des resultierenden 3D-Modells ist die Klasse *CityViewer* zuständig. Dafür hat diese ein *CityScene* Objekt, aus welchem sie das *TriangleMesh* ausliest und dieses mithilfe von *jMonkeyEngine* in einem zweiten Fenster darstellt, in dem der Benutzer das Modell betrachten kann.

CityBuilder

Die Klasse *CityBuilder* instanziiert nacheinander die zuvor genannten Klassen in der benötigten Reihenfolge und stellt die korrekten Abhängigkeiten her. Hier wird die Main-Methode des Programms ausgeführt.

GeoParser

Da die Eingabedateien im Json- oder GeoJson-Format in das Programm gegeben werden, wird eine Klasse benötigt, welche diese Dateien parst. Dafür ist die Klasse *GeoParser* zuständig, welche jeweils eine Methode für Json- und GeoJson-Dateien implementiert. In diesen Methoden werden die eingegebenen Dateien geparst und nach den relevanten Schlüsselwörtern durchsucht und die Koordinaten der Gebäude Grundstücke werden in

einer String Liste gespeichert. Die Strings in dieser Liste werden weiter verarbeitet, indem sie skaliert, translatiert und in das korrekte Format für Axiome gebracht werden. Weiterhin werden die Koordinaten eines Axioms darauf geprüft, ob sie in Uhrzeigersinn aufgelistet sind und falls das nicht der Fall ist, wird die Reihenfolge umgekehrt. Dies ist notwendig, da die Reihenfolge der Koordinaten angibt, in welcher Richtung das Innere des Axioms liegt.

Die nächsten drei Klassen sind im *cgashape* Package implementiert, werden hier jedoch erwähnt, da sie relevant für die Anwendung sind und teilweise erweitert wurden.

Grammar

Die Klasse *CityParams* besitzt ein Objekt der Klasse *Grammar*, welches aus einem String geparst wird. *Grammar* enthält Variablen und Regeln und wird verwendet, um aus einem Axiom eine neue Form zu evaluieren.

MeshGenerator

Ein Objekt der Klasse *MeshGenerator* ist dafür zuständig aus einem ShapeTree ein *TriangleMesh* zu erzeugen. Dafür wird der Methode des *MeshGenerators* ein ShapeTree übergeben, mit dem rekursiv weitere Methoden aufgerufen werden, bis der gesamte ShapeTree durchlaufen wurde. Dabei gibt es für unterschiedliche Formen innerhalb des ShapeTrees verschiedene Methoden des *MeshGenerators* und weitere Klassen, die wie *MeshGenerator* für bestimmte Arten von Formen funktionieren. Für dieses Projekt wurde die Klasse *MeshGenerator* so erweitert, dass weitere Arten von Formen wie das *FachwerkShape* erzeugt werden können.

Weiterhin wurde die Funktion implementiert, die Farben der resultierenden *TriangleMeshes* je nach Art der Grammatik und Art des Gebäudes einer Grammatik zu beeinflussen. So erhalten Gebäude aller drei Grammatiken verschiedene Wand- und Dach- und Detailfarben, die Fensterfarben von Gebäuden der Cyberpunk und Mittelalter Fantasy Grammatik ändern sich zufallsbasiert innerhalb des *TriangleMeshes* eines Gebäudes und Wandfarben der Mittelalter Fantasy Grammatik ändern sich zufallsbasiert von Gebäude zu Gebäude, wobei die Wandfarben der Mittelalter Fantasy Grammatik von der Gebäudeart abhängen.

TriangleMesh

Objekte der Klasse *TriangleMesh* stellen Dreiecksnetze dar und enthalten alle wichtigen Informationen für diese. Dazu gehören die Vertices, Dreiecke und Farben des Dreiecknetzes. Die Klasse enthält weiterhin Methoden zur Generierung eines Dreiecksnetzes.

4.3 Operationen

Für die Implementation der Grammatiken dieses Projektes wurden mehrere neue Operationen implementiert, welche in diesem Abschnitt erläutert werden. Die bereits vorhandenen Operationen *Detail* und *Split* sowie *RepeatedSplit* des *cgashape* Projektes wurden erweitert.

4.3.1 Shapeoperation Rectify

Die Shapeoperation *Rectify* kann auf Formen der Art *PolygonShape* angewendet werden und erzeugt eine neue Form der gleichen Art, wobei die neue Form immer ein Rechteck ist. Sollte die alte Form bereits ein Rechteck sein, wird dieses unverändert zurückgegeben. Für die Überprüfung, ob es sich bei der Form um ein Rechteck handelt, wird geprüft, ob die Anzahl der Vertices vier beträgt. Dann wird für jede der vier Seiten der Form die Steigung berechnet und mit den Steigungen zweier sich berührenden Seiten wird der Winkel zwischen diesen Seiten berechnet. Falls alle diese Winkel einen rechten Winkel darstellen, handelt es sich um ein Rechteck. Handelt es sich bei der alten Form nicht um ein Rechteck, so wird das umgebende, achsenorientierte Rechteck berechnet und verkleinert, um Überschneidungen mit Nachbargebäuden zu minimieren.

4.3.2 Shapeoperation Square

Die Shapeoperation *Square* funktioniert wie *Rectify*, außer dass die neue Form ein Quadrat darstellt. *Square* sollte nur auf Formen angewendet werden, die rechteckig sind. Um diese Art von Formen in ein Quadrat zu transformieren, werden die Längen der Seiten verglichen. Sollten diese nicht gleich sein, werden neue Vertices berechnet, sodass die längeren Seiten auf die Länge der kürzeren gebracht werden, wobei der Mittelpunkt des Rechtecks der gleiche bleibt.

4.3.3 Shapeoperation Fachwerk

Die Shapeoperation *Fachwerk* erweitert die Operation *Extrude* und arbeitet somit mit *PolygonShapes*. Für diese Operation wurde die neue GeometryShape *FachwerkShape* implementiert. Das *PolygonShape* auf dem diese Operation angewandt wird, wird extrudiert und erhält die Farbe Braun.

4.3.4 Shapeoperation Detail

Shapeoperation *Detail* ist im *cgashape* Projekt implementiert und dient dazu, Details auf *PolygonShapes* zu generieren. Es existieren unterschiedliche Arten von Details, die in diesem Projekt verwendet werden, auf die aber nicht genauer eingegangen werden soll. Um diese Arten von Details auszuwählen, wird der Operation ein Detailname übergeben. Für die Implementation dieses Projekts wurde die Shapeoperation erweitert, sodass eine dunkle Öffnung dargestellt werden kann, wobei der Detailname *dark_ opening* lautet. Dies kann beispielsweise für die Darstellung von Fenstern ohne Glas verwendet werden.

4.3.5 Shapeoperationen Split und RepeatedSplit

Die Shapeoperations *Split* und *RepeatedSplit* sind im *cgashape* implementiert. Sie arbeiten auf *PolygonShapes* oder *PrismShapes*, wobei sie in diesem Projekt ausschließlich auf *PolygonShapes* angewendet werden, und dienen dazu diese Formen zu teilen. Dabei werden die Formen entlang der gewählten Achse X, Y oder Z einmal, mehrfach oder wiederholt geteilt. Durch die Funktionsweise des Programms kann es vorkommen, dass einige Axiome der zu generierenden Gebäude nicht gerade oder achsenorientiert im Koordinatensystem liegen. Das kann zur Folge haben, dass die betroffenen Formen schräg geteilt werden, was nicht der gewollten Funktion entspricht. Aus diesem Grund wurde der Achsenparameter der Operationen für das *CityBuilder* Projekt um weitere mögliche Werte - XNOTALIGNED, YNOTALIGNED und ZNOTALIGNED - erweitert, wobei momentan nur die Erweiterung für die x- und z-Achse implementiert ist, da diese Operationen nur auf *PolygonShapes* angewendet werden und diese in der x-z-Ebene liegen. Wird den Operationen einer dieser Werte als Parameter übergeben, während sie auf eine rechteckige *PolygonShape* angewendet wird, so wird die Form so geteilt, als wäre die Kante zwischen dem ersten und zweiten Vertex der Form entlang der x-Achse ausgerichtet. Dazu wurden folgende Funktionen implementiert:

- Die Ausdehnung des Polygons wird in x-Richtung anhand der Länge der Kante zwischen dem ersten und zweiten Vertex der Form und in z-Richtung anhand der Länge der Kante zwischen dem ersten und dritten Vertex der Form berechnet.
- Die Richtung der Teilung wird für XNOTALIGNED entlang der Kante zwischen dem ersten und zweiten Vertex der Form und für ZNOTALIGNED entlang der Kante zwischen dem ersten und dritten Vertex der Form festgelegt.
- Für die Definition der ersten Teilungsebene in Normalenform wird der erste Vertex der Form als Punkt in der Ebene genutzt und als Normale der Ebene die Richtung der Teilung.
- Die restlichen Teilungsebenen werden hinzugefügt, indem ein Offset auf den ersten Vertex der Form addiert wird und der resultierende Punkt sowie die Teilungsrichtung zur Definition der Ebene in Normalform verwendet werden.

4.4 Grammatiken

Wie in Unterkapitel 3.2.2 beschrieben, wurde für jedes der drei Themen eine Grammatik entwickelt. Dabei folgt der grobe Ablauf der Generierung eines Gebäudes immer dem in Abbildung 3.5 dargestellten, welcher sich in den Grammatiken widerspiegelt.

Alle Grammatiken nutzen bestimmte Variablen, wobei einige dieser Variablen in jeder Grammatik genutzt werden. Dazu gehören:

- buildingHeight: bestimmt die ungefähre Höhe der Gebäude, richtet sich nach den Anforderungen aus dem Unterkapitel 3.1.2 und ist als Intervall definiert.
- floorHeight: legt fest, welche Höhe ein Stockwerk eines Gebäudes besitzt.
- windowWidth und windowHeight: legt die Maße der Fenster eines Gebäudes fest und ist als Intervall definiert.
- doorWidth und doorHeight: legt die Maße der Türen eines Gebäudes fest und ist als Intervall definiert.

Das Definieren von Variablen als Intervall dient dazu, Variationen zwischen verschiedenen Gebäuden eines generierten 3D-Modells zu erhalten. Zusätzlich beinhalten die Grammatiken noch Variablen, die nur für eine Grammatik relevant sind. Um die Unterschiede der Grammatiken darzulegen, werden diese abweichenden Variablen in den Unterpunkten der betroffenen Grammatiken erwähnt und beschrieben, wenn sie relevant sind.

In den folgenden Unterkapiteln wird jede der entwickelten Grammatiken genauer beschrieben und es wird auf Besonderheiten und Abweichungen vom allgemeinen Ablauf hingewiesen. Häufige Begriffe, die hier verwendet werden, sind:

- Lot: das Polygon, welches den Grundriss eines jeden Gebäudes darstellt, aus dem es generiert wird, sowie alle weiteren Polygone, die Grundflächen darstellen.
- Body: Prisma, welches ein Teil des Kernkörpers des Gebäudes ist.
- Facade: die Seiten eines Bodys und somit die Polygone, welche die Außenwände eines Gebäudes darstellen.
- Top: obere Seite eines Bodys und somit das Polygon, welches die Fläche auf einem Gebäude darstellt.
- Roof: Dach eines Gebäudes, welches spitz oder flach sein kann.
- Segment: Abschnitt eines Polygons.
- Slice: Ausschnitt eines Polygons.
- Window: Detail, welches ein Fenster darstellt.
- Door: Detail, welches eine Tür darstellt.

Weitere wichtige Symbolnamen und andere Begriffe werden in den Unterpunkten der einzelnen Grammatiken erklärt. Die Rahmen innerhalb der Diagramme zeigen, welche Schritte ungefähr die einzelnen Schritte des allgemeinen Designs aus dem Unterkapitel 3.2.2 widerspiegeln. Die Notationen an den Pfeilen zeigen, wie viele Instanzen des jeweiligen Objekts erzeugt werden, wenn es mehr als eins ist. Die Erklärungen und Gründe für einzelne Schritte und Entscheidungen innerhalb der Grammatiken finden sich in den Beschreibungen der Themen in Unterkapitel 3.1.2.

4.4.1 Cyberpunk

Abbildung 4.3 zeigt wie die Grammatik implementiert wurde, mit der Gebäude für das Thema Cyberpunk generiert werden. Zusätzlich zu den oben genannten Variablen sind hier doorRoomWidth und doorRoomDepth relevant, welche die Maße des Raumes, der bei manchen Gebäuden als Eingang fungiert, bestimmen.

Lot

Ausgehend vom Lot werden in dieser Grammatik zwei Gebäudetypen unterschieden. Der eine Typ ist ein Gebäude mit einem Erdgeschoss, welches eine größere Ausdehnung hat als der Rest des Gebäudes. Der zweite Typ hat von unten bis oben die gleiche Ausdehnung. Dazu wird das ursprüngliche Lot in ein unteres und ein oberes Lot oder in nur ein oberes Lot aufgeteilt. Weiterhin gibt es Gebäudevarianten, bei denen das Gebäude in zwei Hälften aufgeteilt wird, welche unterschiedlich hoch sind. Dazu wird das entsprechende Lot in zwei SplitLots aufgeteilt.

Body

Die entstandenen Lots werden entsprechend hoch ausgedehnt, um so den Body des Gebäudes zu erhalten. Handelt es sich um ein Gebäude mit geteiltem Body, unterscheiden sich diese Höhen. Es gibt die Möglichkeit, dass die Gebäudehälften skaliert werden, sodass deren Grundflächen kleiner werden.

Facade

Je nach Art des Körpers werden dessen Seiten in null bis eine FrontFacade und ansonsten Facades aufgeteilt. Facades werden je nach Stockwerkhöhe in FloorFacades aufgeteilt. Dabei wird der erste Abschnitt der FrontFacade zu einer BaseFloorFacade. Diese FloorFacades werden wiederum in FloorFacadeSegments unterteilt, welche dabei helfen, die Wände der Gebäude für Details zu unterteilen.

Details

Die Details enthalten in dieser Grammatik Fenster und Türen. Für Fenster werden die FloorFacadeSegments in WindowSegments unterteilt, welche in WindowSlices unterteilt werden, auf denen das Window Detail gesetzt wird. Dabei werden die Variablen der Fenstermaße genutzt. Für die Tür wird zwischen zwei Varianten unterschieden. Das BaseFloorSegment wird neben FloorFacadeSegment auch in entweder ein DoorSegment oder ein DoorRoomSegment unterteilt. Das DoorSegment wird in ein DoorSliceSegment unterteilt, auf dem das Door Detail gesetzt wird. Dabei werden die Variablen für die Maße einer Tür genutzt. Das DoorRoomSegment ist hier vereinfacht dargestellt, da es einem ähnlichen Ablauf wie ein Body folgt. Der DoorRoom ist also ein Raum mit Wänden, Fenstern, Tür und Dach, der als Eingang einen Anbau an der Front des Gebäudes darstellt.

Top und Roof

Der Body des Gebäudes wird neben den Seiten auch in das obere Teil aufgeteilt, also Top. Hier gibt es eine Variante, bei der das obere Teil zu einem FlatRoof gemacht wird. Bei der anderen Variante wird Top aufgeteilt in das BuildingRoof und BuildingTop. BuildingRoof wird wiederum zu einem FlatRoof und BuildingTop wird zu einem RoofRoom gemacht. RoofRoom ist hier vereinfacht dargestellt, da es einem ähnlichen Ablauf wie ein Body folgt. Ein RoofRoom stellt einen kleinen Raum mit Tür und Flachdach auf dem Dach eines Gebäudes dar, der beispielsweise als Zugang zum Gebäudedach dient.

4.4.2 Mittelalter Fantasy

Hier wird die Grammatik für Gebäude des Themas Mittelalter Fantasy beschrieben. Es werden drei verschiedene Varianten von Gebäuden generiert. Eine Variante ist ein Gebäude, dessen Erdgeschoss eine verkleinerte Grundfläche im Vergleich zu den oberen Stockwerken hat. Eine andere Variante ist ein Gebäude, dessen Stockwerke alle die gleiche Grundfläche haben. Die letzte Variante ist ein Turm. Abbildung 4.4 zeigt die Implementation der ersten zwei Gebäudevarianten und Abbildung 4.5 zeigt die Implementation des Turms. Zusätzlich zu den oben genannten Variablen sind die Folgenden von Relevanz:

- firstFloorDownscale: legt fest, wie viel das Erdgeschoss einer Gebäudevariante verkleinert wird.

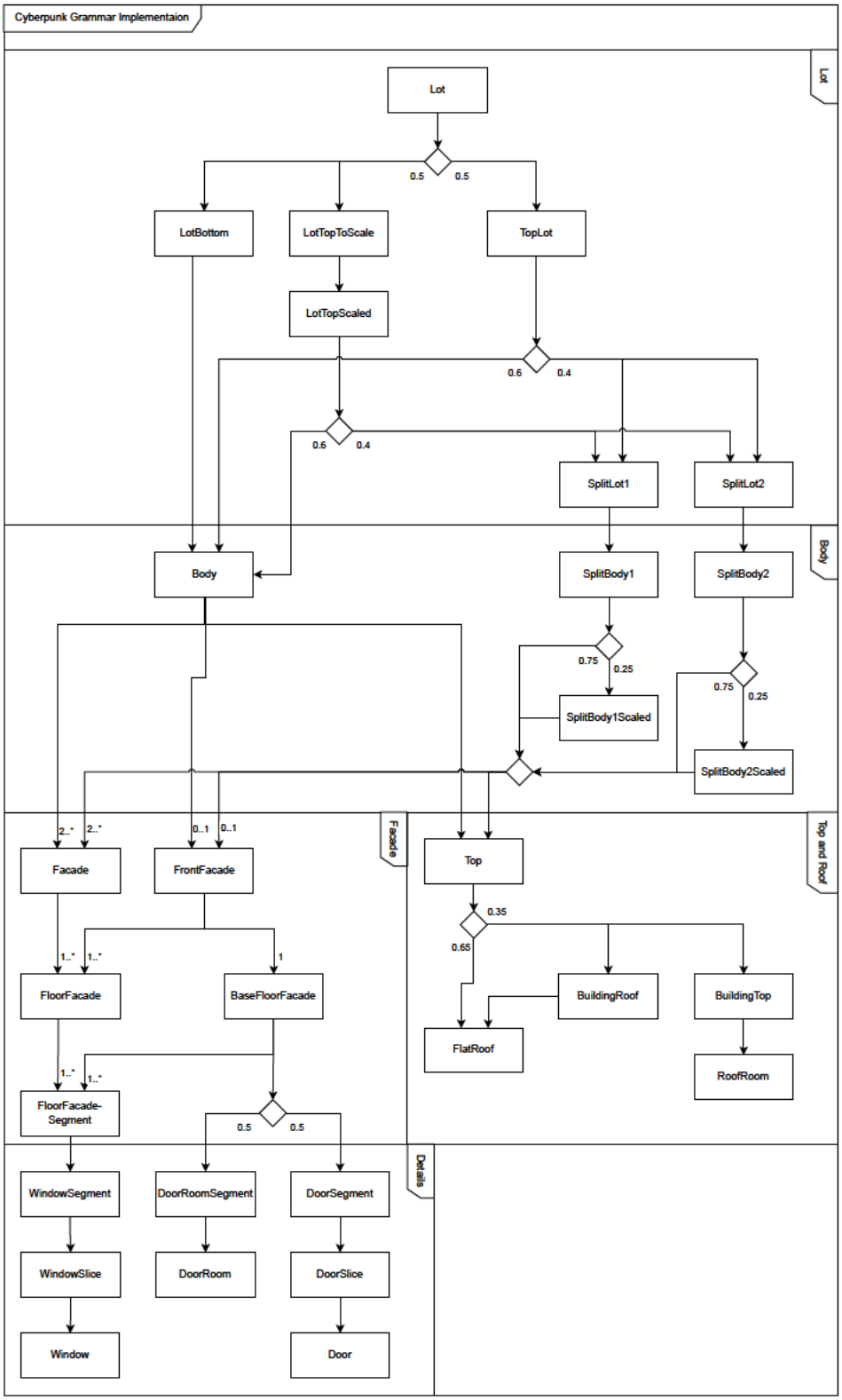


Abbildung 4.3: Design der Cyberpunk Grammatik

- dormerWidth und dormerWindowWidth: legt die Maße der Gauben und Gaubenfenster fest.
- roofAngle: legt den Winkel der Spitzdächer der Gebäude mit Dach fest.
- towerHeight, towerWindowWidth und towerRoofAngle: legt die Maße und Winkel der jeweils entsprechenden Teile eines Turms wie bei den anderen Gebäuden fest.

In dieser Grammatik werden zusätzlich folgende Begriffe verwendet:

- Fachwerk: Detail, welches Fachwerk an einer Außenwand darstellt.
- Dormer: Form, welche eine Gaube an einem Dach darstellt.
- DormerRoof: Satteldach mit Gauben.
- Battlement: Zinnen an der Spitze eines Turms.

Lot

Es wurde entschieden, mit der Grammatik für das Thema Mittelalter Fantasy ausschließlich Gebäude mit viereckigen Grundrissen zu generieren. Diese Entscheidung wurde getroffen, da viereckige Grundrisse besser zum Thema passen und die für dieses Thema geforderten Spitzdächer aus technischen Gründen nur auf Körpern mit viereckigem Grundriss erzeugt werden können. Daher ist der erste Schritt in dieser Grammatik das Lot in ein viereckiges RectangleLot zu transformieren. Danach werden aus dem RectangleLot die Lots für die drei Varianten generiert: OnlyOneLot für das Gebäude mit gleichbleibender Grundfläche, FirstFloorLot und TopLot für das Gebäude mit kleiner Grundfläche im Erdgeschoss und TowerLot für einen Turm. TopLot wird in die Höhe translatiert, sodass es sich über dem Erdgeschoss befindet und die höheren Geschosse darstellen kann. TopLotHigh wird dupliziert und ergibt UpperFloorTop und UpperFloorBottom, wobei UpperFloorBottom den Boden der oberen Stockwerke darstellt. Dies ist notwendig, da ansonsten beim fertigen 3D-Modell die oberen Stockwerke nach unten hin offen wären. Das TowerLot wird zu einem rechteckigen TowerLotSquare gemacht, da entschieden wurde, nur Türme mit rechteckigen Grundflächen zu erzeugen.

Body

Der Körper des Gebäudes mit gleichbleibender Grundfläche OnlyOneLotBody und der Körper für die oberen Stockwerke des Gebäudes mit sich ändernder Grundfläche UpperFloorBody werden erzeugt, indem die entsprechenden Lots extrudiert werden. Für den Körper der unteren Stockwerke des Gebäudes mit sich ändernder Grundfläche FirstFloorBody wird das Lot extrudiert und FirstFloorBodyUnscaled skaliert, sodass die Grundfläche sich verkleinert. Der Körper des Gebäudes vom Typ Turm wird extrudiert, wobei hier die Variable für die Turmhöhe verwendet wird.

Facade

FirstFloorBody wird in die Seitenteile zerlegt, welche zu DoorFacades und Facades werden. Die Seitenteile von UpperFloorBody werden zu mehreren FacadeFachwerk. OneLotBody wird entweder in DoorFacades und Facades oder DoorFacadeFachwerk und mehrere FacadeFachwerk zerlegt. FloorFacadeFachwerk wird vertikal entsprechend der Gebäude- und Stockwerkhöhe in mehrere FloorFacadeFachwerk aufgeteilt. Der TowerBody wird in TowerFacades und TowerDoorFacades zerlegt. Diese werden vertikal entsprechend der Höhe und Variablen in NoWindowsFacades, TowerFacades und im Falle der TowerDoorFacades in eine TowerFirstFloorFacade zerlegt. TowerFloorfacades werden in TowerFloorFacadeSegments unterteilt.

Details

In dieser Grammatik gibt es ein zusätzliches Detail für Fachwerk. Die DoorFacade wird in DoorFacadeSegments und ein DoorSegment aufgeteilt. FloorFacadeFachwerk und DoorFacadeFachwerk werden in FloorFacadeFachwerkSegments unterteilt, wobei DoorFacadeFachwerk zusätzlich in ein DoorSliceFachwerk unterteilt wird. FloorFacadeFachwerkSegment wird entweder zu einem WindowSegment oder einem NoWindowSegment. TowerFloorFacadeSegment wird entweder ein TowerWindowSegment oder ein TowerNoWindowSegment und TowerFirstFloorSegment wird in ein DoorSegment und mehrere TowerNoWindowSegment aufgeteilt. Auf DoorSegment, DoorSliceFachwerk, WindowSegment und TowerWindowSegment werden analog zur Cyberpunk Grammatik die Door beziehungsweise Window Details erzeugt. Das Erzeugen der Fachwerk Details ist in der Abbildung

vereinfacht dargestellt. Es werden DoorSegmentFachwerk, DoorSliceFachwerk, NoWindowSegment und WindowSegment mit mehreren geeigneten Teiloperationen in kleinere Slices geteilt, auf denen das Fachwerk Detail generiert wird.

Top und Roof

Wie bei der Grammatik für das Cyberpunk Thema entstehen aus den Körpern auch die oberen Teile Top. Für die beiden Gebäude Varianten, die keine Türme sind, wird aus Top ein DormerRoof vom Typ ridge also ein Satteldach erzeugt. Auf diesen Dächern werden Dormers also Gauben erzeugt, welche ein Window Detail erhalten. Dieser Prozess ist hier der Übersicht halber vereinfacht dargestellt. Um die Dormer zu erzeugen, wird das Dach in seine Komponenten aufgeteilt. Die Side Komponenten des Dachs werden mit verschiedenen Variablen für die Maße der Gauben in Segmente und Slices aufgeteilt, in denen mit der dormer Operation eine Gaube vom Typ gabled erzeugt wird. An welcher Stelle eine Gaube erzeugt wird und an welcher Stelle nicht, ist zufallsbasiert. Eine Gaube wird wiederum in Komponenten aufgeteilt, welche in Segmente und Slices unterteilt werden, um das Window Detail zu erzeugen.

Bei Gebäuden vom Typ Turm werden zwei Varianten für den oberen Teil unterschieden. Dazu wird TowerTop zufallsbasiert entweder zu TowerTopBodyBattlements für einen Wehrturm mit Zinnen oder zu TowerTopBody für einen Wohnturm mit Dach extrudiert sowie skaliert, sodass der oberste Teil des Turms eine größere Grundfläche hat als der Rest des Turms. TowerTopBody wird aufgeteilt in FacadeFachwerk für die Seiten und TowerTopTop für das obere Teil. Die Seiten werden wie im Punkt Facade beschrieben weiter bearbeitet, was in der Abbildung aus Gründen der Leserlichkeit weggelassen wurde. Aus TowerTopTop wird ein TowerTopRoof vom Typ Skeleton also ein Zeltdach erzeugt. Das Erzeugen der Battlements also Zinnen ist in der Abbildung ebenfalls vereinfacht dargestellt. Zum Erzeugen der Zinnen, wird das obere Teil TowerTopBodyBattlements in das obere Teil TowerTopBattlement aufgeteilt. TowerTopBattlement wird nun in X-Richtung geteilt, wobei die zwei äußeren Ränder mit einer definierten Breite zu Battlements werden. Das mittlere Teil wird in Z-Richtung geteilt und die äußeren Ränder mit der gleichen Breite werden zu BattlementsZ. Battlements werden in Z-Richtung gleichmäßig in fünf Formen mit dem Symbol Battlement und vier Formen mit dem Symbol BetweenBattlement aufgeteilt. BattlementsZ werden in X-Richtung gleichmäßig in drei Formen mit dem Symbol Battlement und zwei Formen mit dem Symbol BetweenBattlement aufgeteilt. Zum Schluss werden alle Formen Battlement um einen definierten Wert

extrudiert und alle Formen BetweenBattlement um einen Wert, der ungefähr der Hälfte des vorherigen Werts entspricht, ebenfalls extrudiert.

4.4.3 Antikes Griechenland

Abbildung 4.6 zeigt die Implementierung der Grammatik zur Erzeugung von Gebäuden des Themas Antikes Griechenland. Mit dieser Grammatik werden zwei Gebäude Arten erzeugt, Wohngebäude und Tempel. Zusätzlich zu den oben genannten Variablen sind die Folgenden von Relevanz:

- templeHeight: legt die Höhe eines Tempels fest.
- columnRadius: legt die Breite von Tempelsäulen fest.

In dieser Grammatik werden zusätzlich folgende Begriffe verwendet:

- Steps: Prisma welches Stufen darstellt.
- Column: Prisma, welches die Säule eines Tempels darstellt.
- RidgeRoof: Satteldach.
- SkeletonRoof: Walmdach.

Lot

Analog zur Grammatik des Themas Mittelalter Fantasy wird das Lot in ein Rectangle-Lot umgewandelt. Aus diesem werden entweder ein StandardLot oder ein TempleLot und TempleTop. Das TempleLot wird dupliziert, sodass ein TempleStepsLot und ein TempleColumnsLot entstehen. Das TempleTop wird mit Teilungs-Operationen so verkleinert, dass es nur den inneren Teil des Tempels und nicht die Stufen abdeckt, und entsprechend der Höhe des Tempels in die Höhe translatiert.

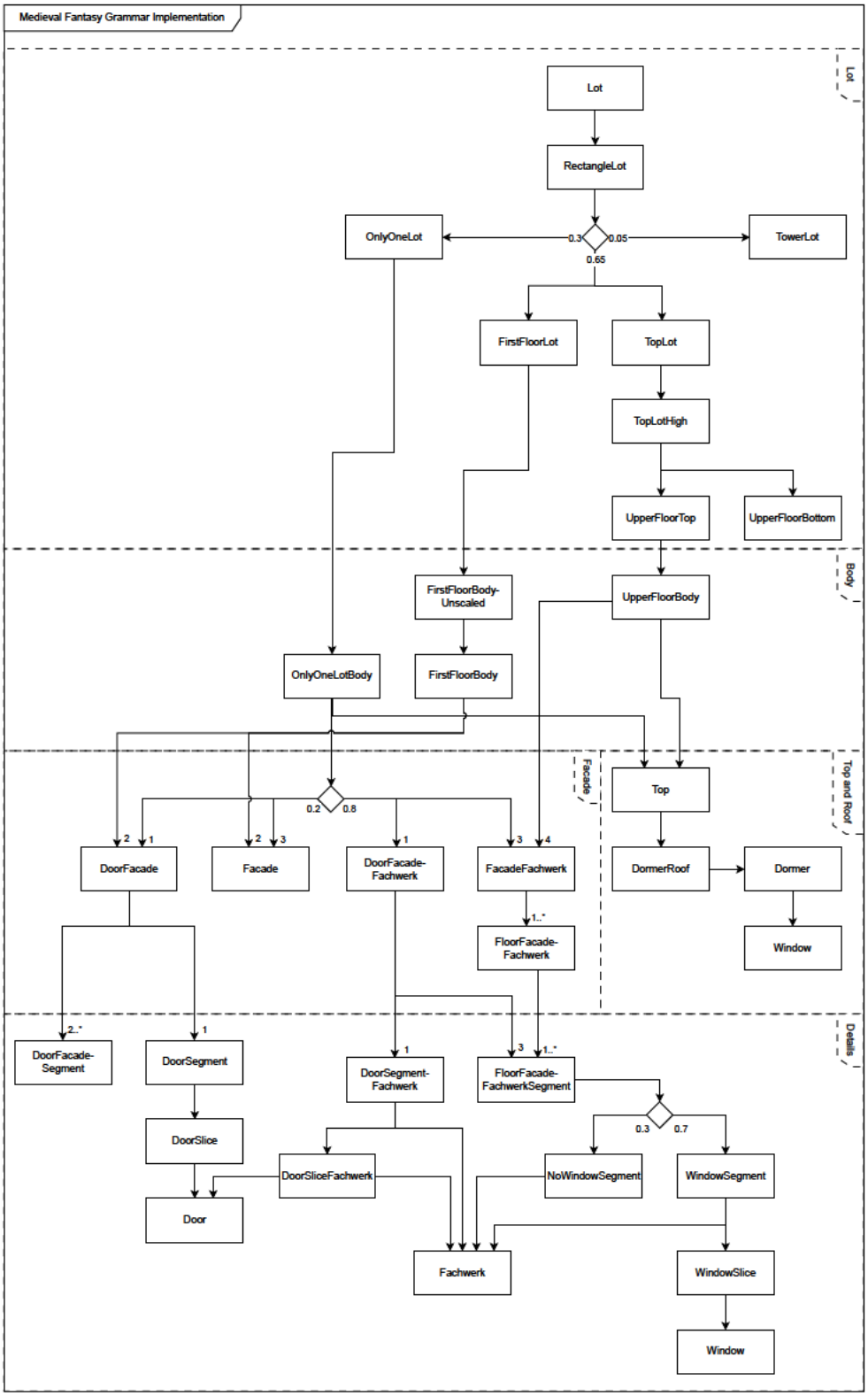


Abbildung 4.4: Design der Mittelalter Fantasy Grammatik

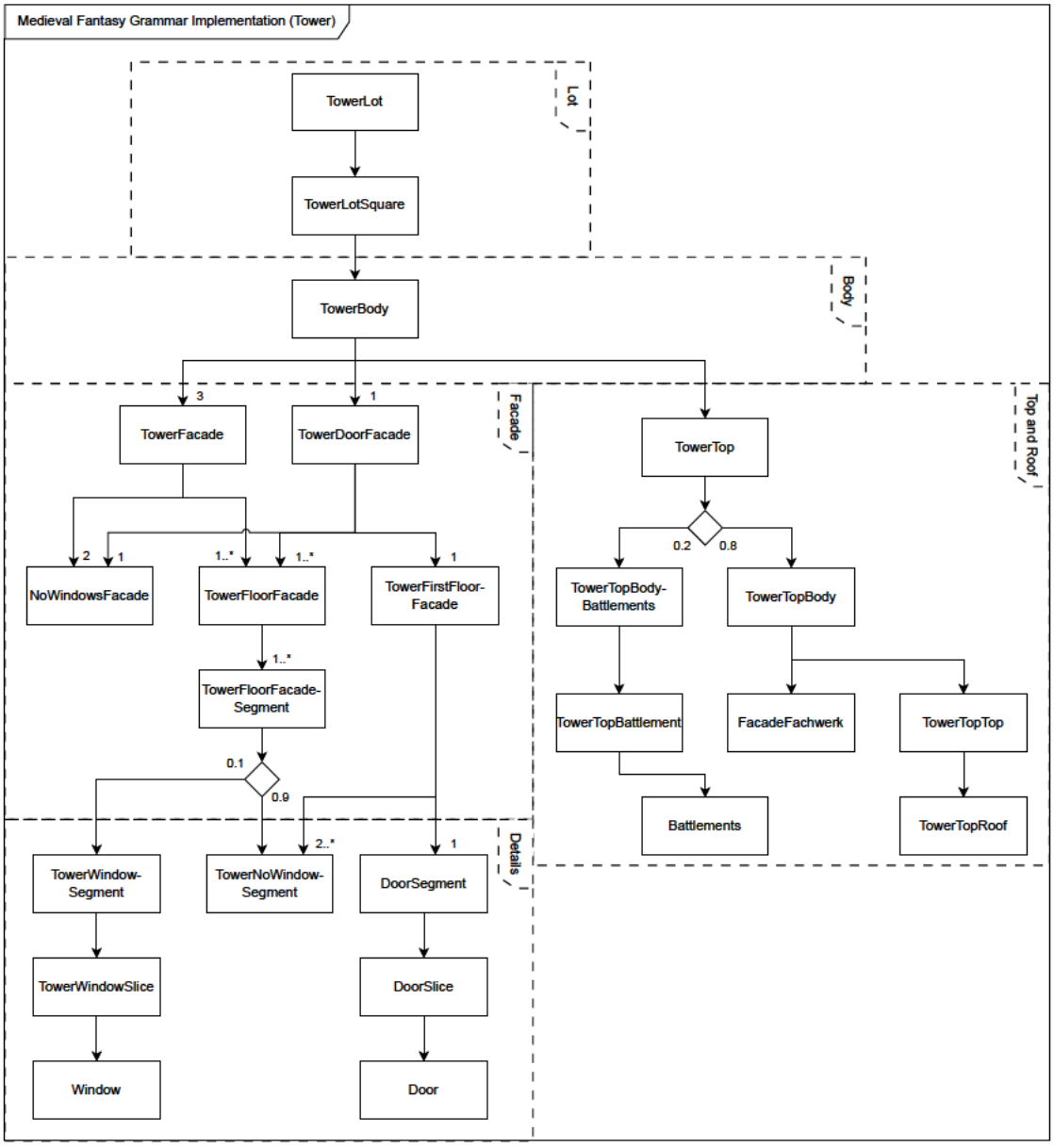


Abbildung 4.5: Design der Mittelalter Fantasy Grammatik für einen Turm

Body

Das StandardLot wird entsprechend der Variablen nach oben ausgedehnt und wird so zu StandardBody. Sowohl die Generierung der TempleSteps als auch die der Columns ist hier vereinfacht dargestellt. Um die TempleSteps zu erhalten, werden mit mehreren geeigneten Teilungs-Operationen die Ränder des TempleStepsLot heraus geteilt. Dies wird mit dem übrig gebliebenen TempleStepsLot zweimal wiederholt, sodass drei Umrandungen entstehen. Jede dieser Umrandungen wird in die Höhe ausgedehnt, wobei die äußerste Umrandung am niedrigsten bleibt und die innerste Umrandung am höchsten wird, und so zu TempleSteps. Die übrig gebliebene Mitte wird auf die Höhe der höchsten Stufe ausgedehnt. TempleColumnsLot wird mit Teilungs-Operationen so verkleinert, dass es sich innerhalb der TempleSteps befindet. Mit mehreren geeigneten Teilungs-Operationen in verschiedene Achsenrichtungen werden die Ränder von TempleColumnsLot heraus geteilt und diese in Columns und Zwischenräume geteilt. Die Columns werden auf die Höhe des Tempels ausgedehnt.

Facade

Der StandardBody wird analog zu den vorherigen Grammatiken weiterverarbeitet, sodass Facades und Top entstehen.

Details

Analog zu den vorherigen Grammatiken werden die Facades so weiterverarbeitet, dass Window und Door Details generiert werden können.

Top und Roof

Analog zu den vorherigen Grammatiken wird Top so weiterverarbeitet, dass ein Roof generiert werden kann, welches hier ein Skeleton-, Ridge- ode FlatRoof sein kann.

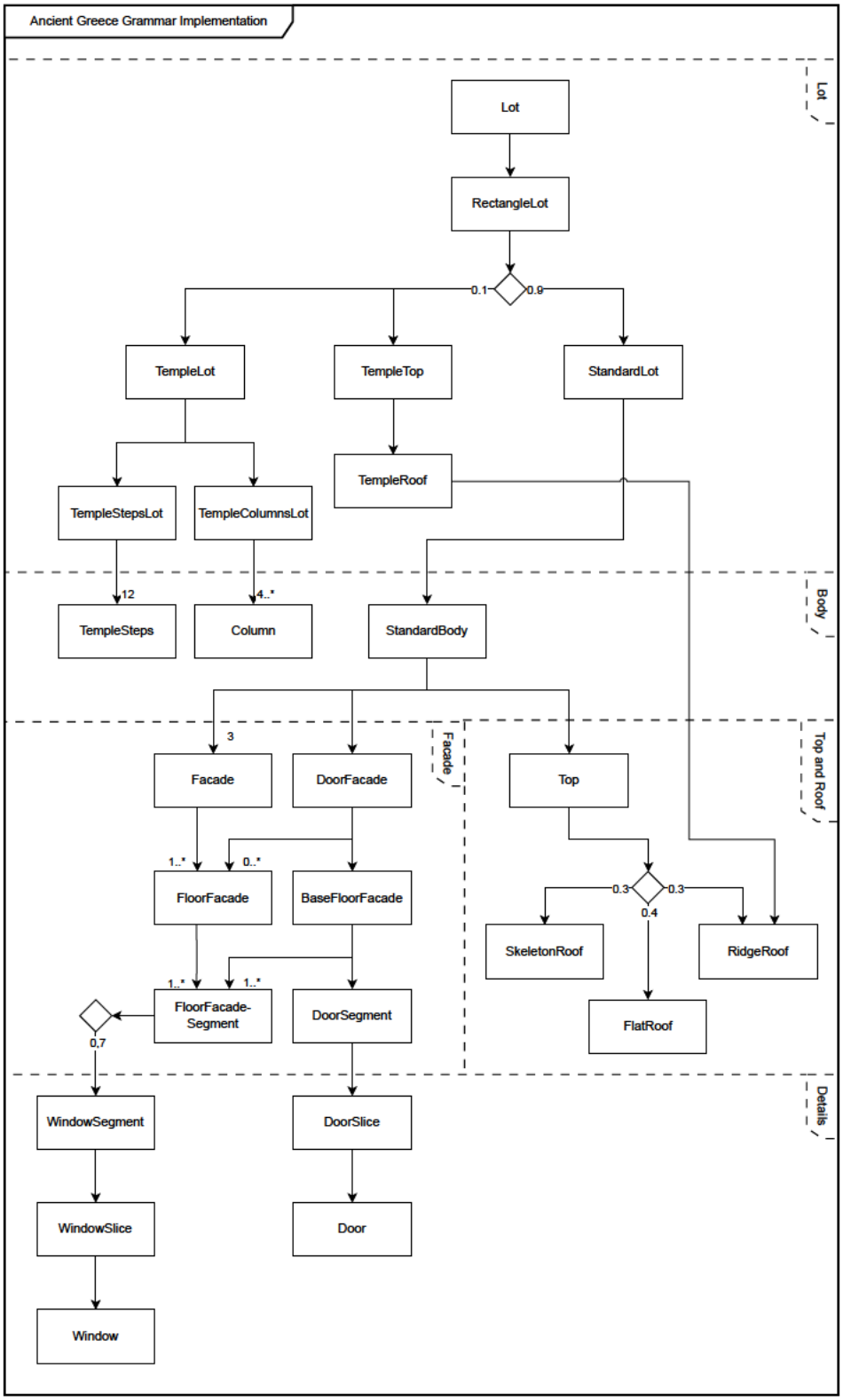


Abbildung 4.6: Design der Antikes Griechenland Grammatik

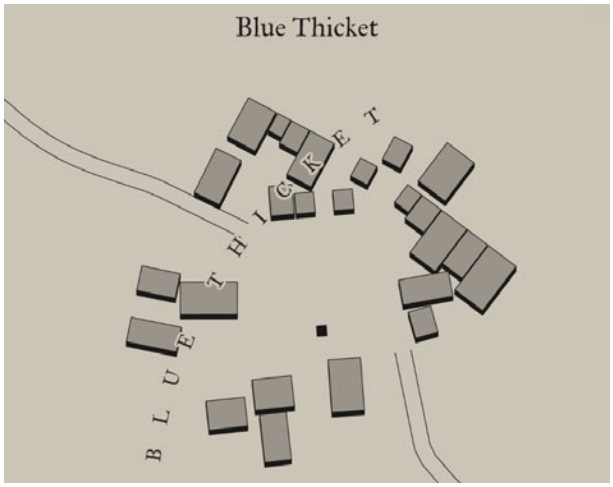


Abbildung 4.7: Grundriss einer zufallsbasiert erzeugten Stadt

4.5 Verwendung des Programms

Hier wird erklärt, was bei der Anwendung der Software zu beachten ist.

Wie in Unterkapitel 3.1.1 beschrieben, existieren die zwei möglichen Datei-Arten JSON und GEOJSON für die Eingabe des Programms. Die hier verwendeten und mitgelieferten JSON-Dateien wurden mit dem zufallsbasierten Generator für mittelalterliche Stadt Grundrisse *Medieval Fantasy City Generator*[1] erzeugt. Abbildung 4.7 zeigt ein Beispiel für so einen Grundriss. Hier können verschiedene Einstellungen vorgenommen werden, anhand derer zufällige Grundrisse erzeugt werden, welche über den Menüpunkt Settlement als JSON-Datei exportiert werden können. Die exportierten Dateien haben das korrekte Format für die Anwendung dieses Projektes.

Die hier verwendeten GEOJSON-Dateien wurden mit der freien, open source Geo-Informationssystem-Software *QGIS*[2] erzeugt. Mithilfe dieser Software können Geodaten, welche unter anderem Informationen über die Grundrisse von Gebäuden enthalten, aus den Karten von *OpenStreetMap*[3] exportiert werden. Dabei ist für diese Anwendung beispielsweise nach den folgenden Punkten vorzugehen:

[1] Entwickler: watabou auf *itch.io*, URL: https://watabou.itch.io/medieval-fantasy-city-generator, Abrufdatum: 01.12.2023
[2] *QGIS*, URL: https://www.qgis.org/de/site/, Abrufdatum: 01.12.2023
[3] *OpenStreetMap*, URL: https://www.openstreetmap.org/, Abrufdatum: 01.12.2023

Abbildung 4.8: Ansicht der Gebäudegrundrisse um Berliner Tor 7 in Hamburg in *QGIS*

- *OpenStreetMap* in *QGIS* öffnen und in den gewünschten Kartenabschnitt reinzoomen.
- In *QGIS QuickOSM* starten und eine neue Abfrage über den Schlüssel building und der Kartenfenster-Ausdehnung als räumliche Ausdehnung für die Abfrage starten.
- Mit dem Cursor alle Gebäude, die generiert werden sollen, auswählen und dabei darauf achten, dass nur Gebäude, dessen Grundrisse ein Polygon darstellen, gewählt werden.
- Mit einem Rechtsklick auf das nach dem zweiten Schritt entstandene Layer building unter dem Kontextmenüpunkt Export die gewählten Objekte speichern und dabei das KBS auf Pseudo-Mercator stellen.

Die entstandene GEOJSON-Datei kann für die Generierung von Gebäudegruppen verwendet werden. Abbildung 4.8 zeigt eine Sicht in *QGIS* mit markierten Gebäudegrundrissen um das Gebäude Berliner Tor 7 in Hamburg herum.

Für die Erzeugung eines neuen 3D-Modells müssen eine Grundriss-Datei und eine Grammatik gewählt werden und der Button Generate geklickt werden. Die gewählte Grammatik-Datei kann bei Bedarf im Editor-Fenster der Anwendung angepasst und gespeichert werden. Weiterhin kann eingestellt werden, wie viele Gebäude maximal generiert werden sollen, wobei jeder Wert unter null so viele Gebäude generiert, wie der Grundriss Gebäude-Grundstücke enthält. Das generierte Modell kann im zweiten Fenster der Anwendung betrachtet werden. Wird eine Layout-Datei gewählt, ohne den Generate-Button zu klicken, so wird der gesamte Grundriss der gewählten Datei angezeigt. Wird der Generate Button gedrückt, während eine Layout-Datei und keine Grammatik gewählt sind,

so wird der Grundriss von so vielen Gebäude-Grundstücken angezeigt, wie die maximale Anzahl an zu generierenden Gebäuden eingestellt ist. Um eine Änderung im Fenster des 3D-Modells angezeigt zu bekommen, muss das Fenster beispielsweise mit einem Mausklick aktiviert werden.

5 Evaluation

In diesem Kapitel werden die Ergebnisse dieser Arbeit dargestellt und bewertet sowie Alternativen für einzelne Punkte der Implementierung erläutert.

5.1 Bewertung der Generierungs-Ergebnisse

Hier werden die Ergebnisse der wiederholten Generierung mit den verschiedenen Grammatiken dargestellt und diskutiert. Es wurde für jede Generierung das gleiche Layout verwendet, welches in Abbildung 5.1 zu sehen ist.

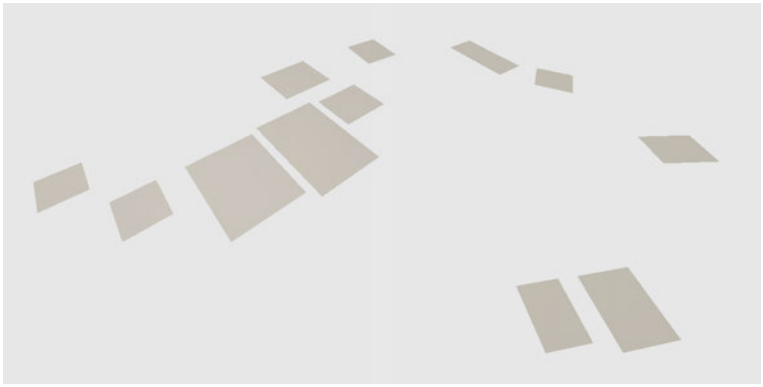


Abbildung 5.1: Layout, welches zur mehrfachen Generierung mit allen Grammatiken verwendet wurde

5.1.1 Cyberpunk

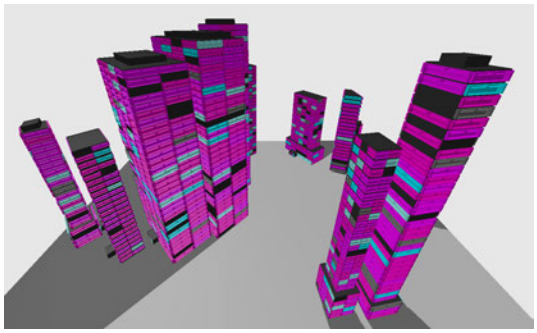
Abbildung 5.2 zeigt die Ergebnisse von drei nacheinander durchgeführten Generierungen einer Gebäudegruppe des Themas Cyberpunk. Es ist zu erkennen, dass jede Generierung ein Modell erzeugt, welches sich von den anderen Modellen unterscheidet. Die in

Abschnitt 3.1.2 festgelegten Charakteristika für das Cyberpunk Thema werden von jedem Modell erfüllt. Die einzelnen Gebäude des Modells sind hoch, sie vermitteln den Eindruck, Wolkenkratzer zu sein, da ihre Höhe einem vielfachen der Breite und Tiefe entspricht, wobei die Höhe hier aus Performance-Gründen auf maximal 100 Meter begrenzt wurde. Die Fassaden der Gebäude bestehen größtenteils aus Fenstern, die Dächer sind flach und teilweise mit einem extra Raum für den Dachzutritt versehen. Farblich besteht das Modell aus dunklen Farben wie Schwarz gemischt mit Neonfarben wie Magenta und Hellblau. Varianz wird erreicht, indem Farben unterschiedlich verteilt werden und weil die Grammatik unterschiedliche Architekturen wie unterschiedliche Höhen innerhalb eines Modells für den gleichen Grundriss erzeugt. Die Gebäude wurden entsprechend dem Layout generiert.

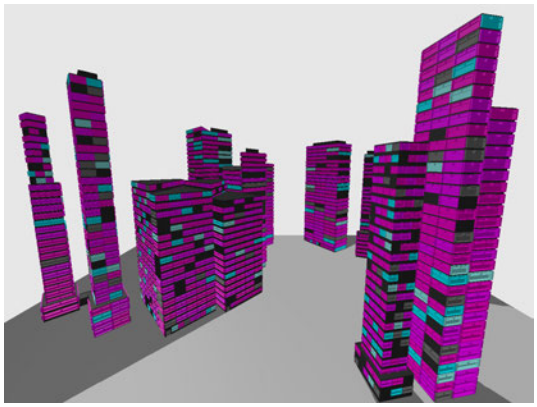
Ein Problem der Grammatik dieses Themas, welches aus Zeitgründen im Rahmen dieser Arbeit nicht gelöst werden konnte, ist, dass der Raum auf dem Dach eines Gebäudes teilweise über den Rand des Gebäudes hängt. Ein Ansatz, dieses Problem zu lösen, wäre, für die Verschiebung des Raums eine neue Operation zu implementieren, welche die Größe des Raums und die Größe des Dachs kennt und den Raum nur soweit verschiebt, dass er nicht über den Rand des Dachs hinaus verschoben wird.

5.1.2 Mittelalter Fantasy

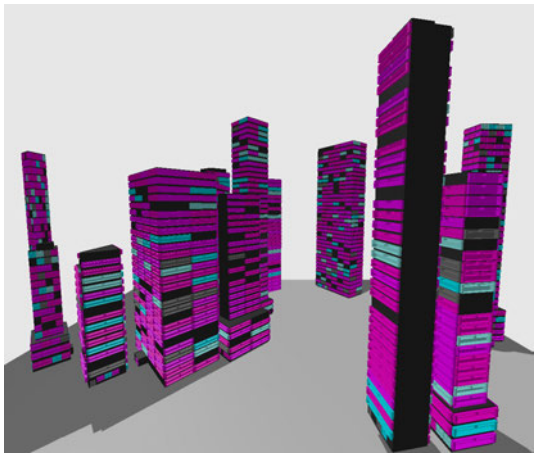
Abbildung 5.3 zeigt die Ergebnisse von drei nacheinander durchgeführten Generierungen einer Gebäudegruppe des Themas Mittelalter Fantasy. Es ist zu erkennen, dass jede Generierung ein Modell erzeugt, welches sich von den anderen Modellen unterscheidet. Die in Abschnitt 3.1.2 festgelegten Charakteristika für das Mittelalter Fantasy Thema werden von jedem Modell erfüllt. Wohngebäude sind zwischen sechs und zehn Metern hoch, während Türme bis zu 30 Meter hoch sind. Die Fassaden der Gebäude weisen Fachwerk und einige Fenster auf und die Dächer sind Satteldächer mit Gauben beziehungsweise Walmdächer für Türme. Die Modelle weisen an den Wänden die Farben Weiß, Gelb, Rot und Grau auf, während Dächer, Fachwerk und Türen braun sind. Unterschiedliche Arten von Gebäuden, wie schlichte Gebäude ohne Fachwerk, Gebäude mit Fachwerk mit oder ohne verkleinertem Erdgeschoss und Türme, Farbverteilungen und beleuchtete Fenster sorgen für Abwechslung in verschiedenen Generierungen. Die Lage und Größe der Gebäude entspricht dem Layout.



(a) Ergebnis der ersten Generierung

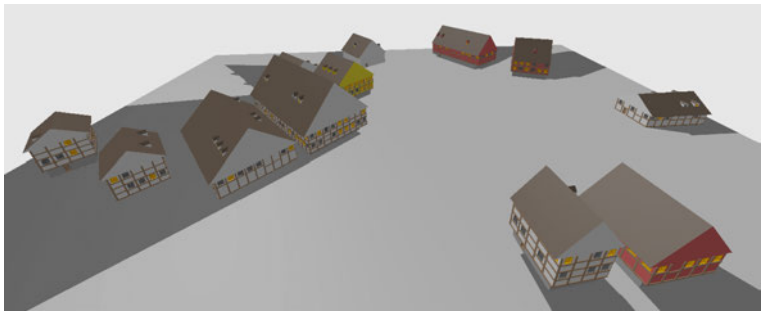


(b) Ergebnis der zweiten Generierung

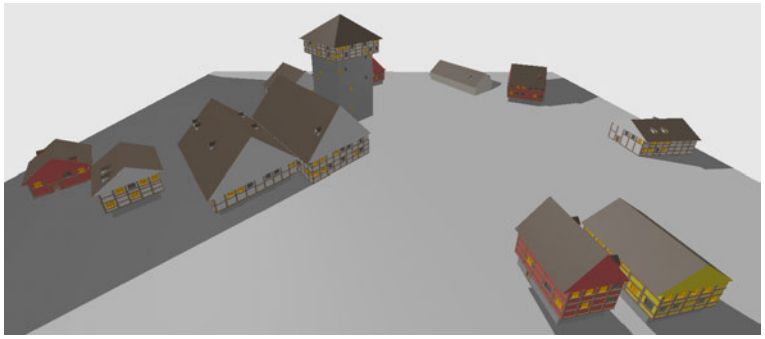


(c) Ergebnis der dritten Generierung

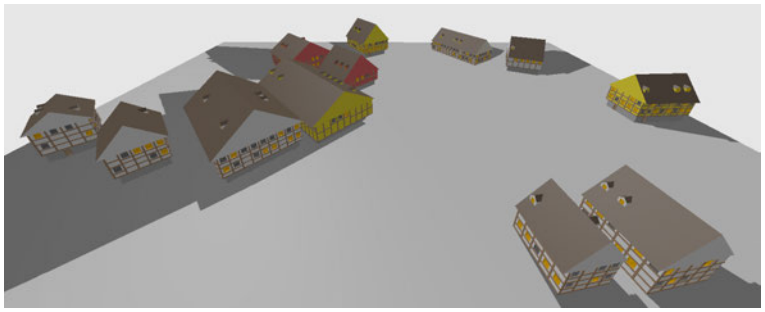
Abbildung 5.2: Drei nacheinander durchgeführte Generierungen einer Cyberpunk Gebäudegruppe



(a) Ergebnis der ersten Generierung



(b) Ergebnis der zweiten Generierung



(c) Ergebnis der dritten Generierung

Abbildung 5.3: Drei nacheinander durchgeführte Generierungen einer Mittelalter Fantasy Gebäudegruppe

5.1.3 Antikes Griechenland

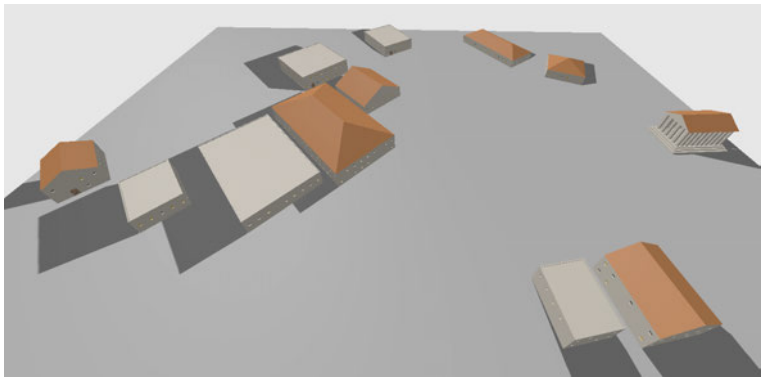
Abbildung 5.4 zeigt die Ergebnisse von drei nacheinander durchgeführten Generierungen einer Gebäudegruppe des Themas Antikes Griechenland. Es ist zu erkennen, dass jede Generierung ein Modell erzeugt, welches sich von den anderen Modellen unterscheidet. Die in Abschnitt 3.1.2 festgelegten Charakteristika für das Thema Antikes Griechenland werden von jedem Modell erfüllt. Die Gebäude sind flach, zwischen drei und sechs Metern

hoch, beziehungsweise etwas höher, bis zu zehn Meter, wenn es sich um einen Tempel handelt. Fassaden sind schlicht gehalten und weisen kleine Fenster ohne Glas auf. Tempel sind von Stufen umgeben, die in das Innere führen und weisen rundherum Säulen auf. Während Tempel Satteldächer haben, können andere Gebäude Sattel-, Walm- oder Flachdächer haben. Farblich sind die Gebäude in den Farben des Baumaterials Sandstein und Terrakotta gehalten. Durch die unterschiedlichen Gebäudearten, Dächer und beleuchteten oder dunklen Fenster wird Varianz zwischen den generierten Modellen erreicht. Die Lage und Größe der Grundrisse im Layout wird von den einzelnen Gebäuden eingehalten.

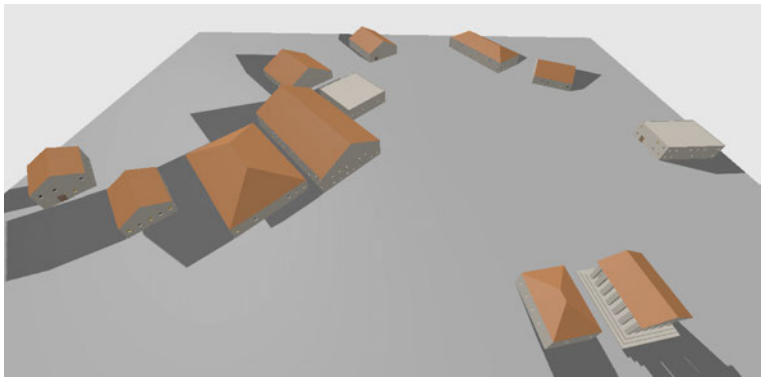
5.1.4 Andere Layouts

Wie im Abschnitt 4.5 beschrieben ist es möglich, Layouts aus der echten Welt in das Programm einzugeben. Abbildung 5.5 zeigt einen Grundriss aus Hamburg, welcher die Gebäude um Berliner Tor 7 beinhaltet und jeweils ein Modell der Themen Cyberpunk und Mittelalter Fantasy, die mit diesem Layout generiert wurden. Es ist zu erkennen, dass mit diesem Layout ein deutlich besseres 3D-Modell für das Thema Cyberpunk erzeugt werden kann. Das liegt zum einen daran, dass die Lots in der Grammatik des Themas Mittelalter Fantasy zu Rechtecken gemacht werden, was teilweise ungewünschte Nebeneffekte hat. So kann es vorkommen, dass Gebäude sich schneiden oder wie in diesem Fall ineinander liegen. Andererseits liegt es daran, dass sich das Layout von vornherein besser für ein Thema wie Cyberpunk eignet, da es Lots für Gebäude mit sehr großen Grundrissen enthält, welche nicht zu der Art von Gebäuden passt, die für Themen wie Mittelalter Fantasy oder Antikes Griechenland erzeugt werden.

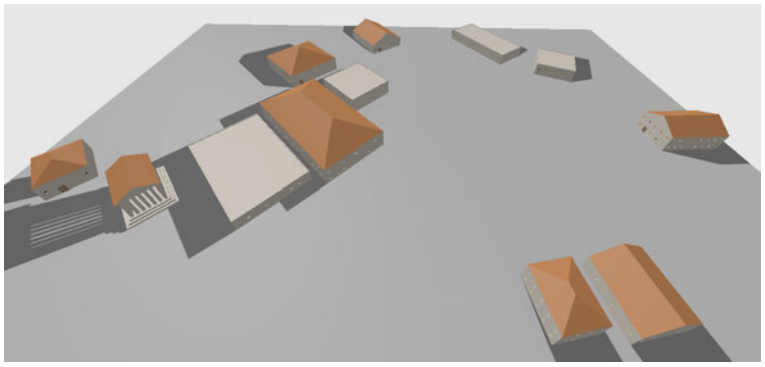
Abbildung 5.6 zeigt einen Grundriss, welcher die Gebäude in und um den Elsternweg in Maschen, Seevetal zeigt und jeweils ein Modell der Themen Cyberpunk und Mittelalter Fantasy, die mit diesem Layout generiert wurden. Im Gegensatz zum Layout des Berliner Tor 7, ist es kein Problem, ein Modell des Themas Mittelalter Fantasy mit diesem Layout zu generieren. Das liegt daran, dass die Grundstücke in diesem Layout größtenteils schon rechteckig sind, sodass keine Überschneidungen bei der Umwandlung der Grundstücke geschehen können und daran, dass die Grundstückgrößen besser zu Themen wie Mittelalter Fantasy und Antikes Griechenland passen. Auch ein Modell für das Thema Cyberpunk kann erzeugt werden, es sieht jedoch nicht so gut aus, wie ein Modell, dass auf einem Layout einer Stadt basiert.



(a) Ergebnis der ersten Generierung



(b) Ergebnis der zweiten Generierung



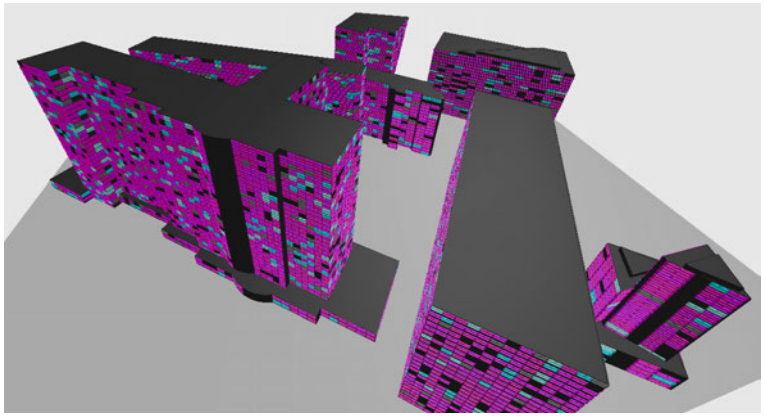
(c) Ergebnis der dritten Generierung

Abbildung 5.4: Drei nacheinander durchgeführte Generierungen einer Antikes Griechenland Gebäudegruppe

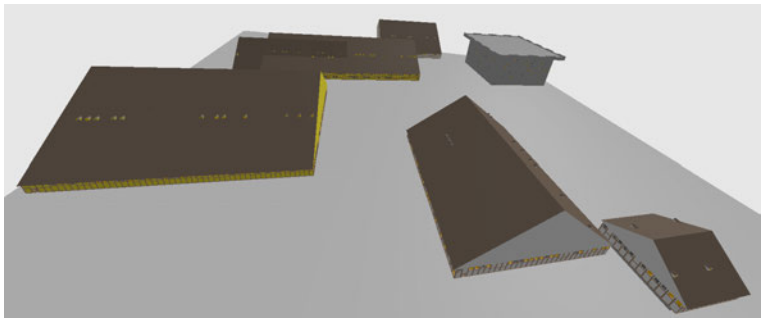
Rechteckige Grundrisse

Wie bereits erläutert, werden die Grundstücke für Themen wie Mittelalter Fantasy und Antikes Griechenland in Rechtecke umgewandelt. Die Gründe dafür sind technisch, da

(a) Layout der Gebäude um Berliner Tor 7



(b) Modell für das Thema Cyberpunk

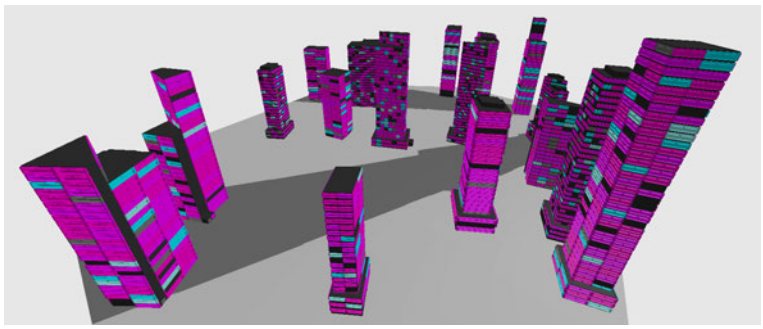


(c) Modell für das Thema Mittelalter Fantasy

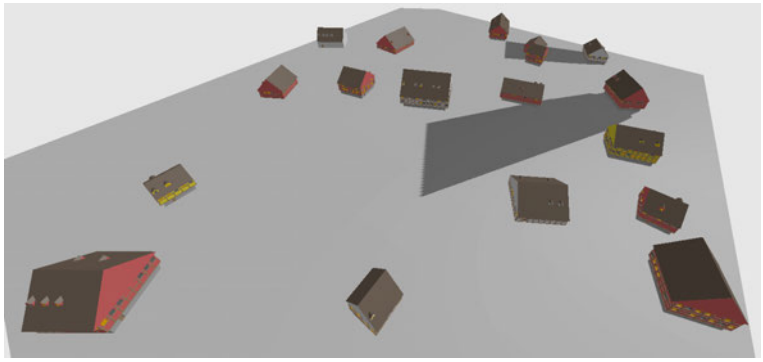
Abbildung 5.5: Mit dem Grundriss der Gebäude um Berliner Tor 7 erzeugte Modelle

in der aktuellen Implementation des *cgashape* Projektes Dächer, die nicht flach sind, nur auf rechteckigen Formen generiert werden können. Dieses Problem könnte mit anderen Implementierungen und Algorithmen wie dem Straight Skeleton Algorithmus [1] behoben werden, welcher in diesem Projekt nicht umgesetzt wurde, da es nicht das Thema der Arbeit war. Es gibt aber auch inhaltliche Gründe für dieses Vorgehen. So wären

(a) Layout der Gebäude um den Elsternweg in Maschen



(b) Modell für das Thema Cyberpunk



(c) Modell für das Thema Mittelalter Fantasy

Abbildung 5.6: Mit dem Grundriss der Gebäude um den Elsternweg in Maschen erzeugte Modelle

Grundrisse aus zusammengesetzten Rechtecken für Gebäude aus dem Mittelalter oder der Antike noch denkbar, komplexere Grundrisse, wie sie Heute teilweise vorkommen, könnten die erzeugten Gebäude jedoch unglaubwürdig machen. Daher wäre ein Ansatz, das Umwandeln der Grundstücke in Rechtecke auf eine bessere Art umzusetzen, indem

beispielsweise das flächenmäßig größte Rechteck in dem ursprünglichen Polygon des Lots gesucht wird und dieses Rechteck das neue Lot wird. Der beschriebene Ansatz für eine Implementation eines solchen Algorithmus von Bernard Chazelle [4] wurde in dieser Arbeit nicht umgesetzt, da es den Umfang dieses Projektes überschreiten würde.

5.2 Performance

5.2.1 Zeitmessungen

Um die Performance der Anwendung zu evaluieren, wurden mehrere Messungen durchgeführt, welche in Abbildung 5.7 dargestellt sind. Es wurden für alle drei Themen jeweils die Dauer der Generierung eines vollständigen 3D-Modells für vier verschiedene Layouts gemessen. Die Dauer für jedes Thema-Layout-Paar wurde dabei mit mehreren Messungen, aus denen der Mittelwert berechnet wurde, ermittelt, da zwei Generierungen eines Modells mit dem gleichen Layout und der gleichen Grammatik unterschiedlich lange dauern können. Diese Unterschiede können mit der zufälligen Natur der prozeduralen Generierung erklärt werden. So macht es einen Unterschied für die Dauer der Generierung, wie hoch die Gebäude eines Modells werden, denn je höher das Gebäude wird, desto mehr Details werden dargestellt und die Erzeugung von Details wie Fenstern benötigt viel Zeit.

Die für die Messungen genutzten Layouts sind das Whit-Widog-Layout aus der Abbildung 5.1, welches ein Layout mit zwölf Gebäuden darstellt, das Berliner Tor-Layout aus Abbildung 5.5 mit sieben Gebäuden und das Maschen-Layout aus Abbildung 5.6, welches 18 Gebäude enthält. Dabei wurde das erste Layout mit dem *Medieval Fantasy City Generator* und die anderen zwei Layouts mit *QGIS* aus der *OpenStreetMap*-Karte exportiert. Abbildung 5.7 zeigt die Ergebnisse der Messungen, wobei die y-Achse die Laufzeit in Millisekunden zeigt.

Es ist leicht zu erkennen, dass die Generierungen der Cyberpunk Modelle bei Weitem die längsten Laufzeiten haben, mit fast 60 Minuten Laufzeit für die Generierung des Berliner Tor-Modells. Am zweitlängsten benötigt die Mittelalter Fantasy Grammatik mit knapp acht Minuten Laufzeit für das Berliner Tor-Modell, währen die Laufzeiten der Grammatik für Antikes Griechenland im Vergleich zu vernachlässigen sind. Diese Laufzeiten für besonders detaillierte Grammatiken wie die Cyberpunk Grammatik sind ungenügend, da dadurch der Einsatz in Echtzeit-Anwendungen unmöglich wird, aber

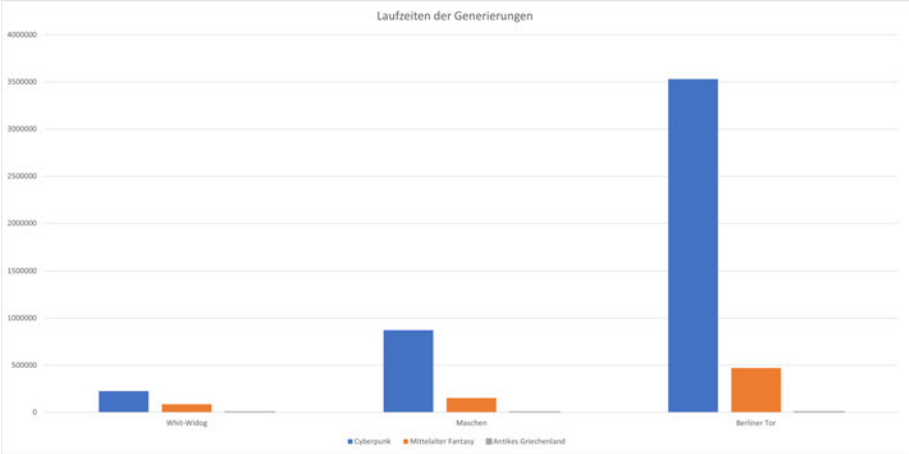


Abbildung 5.7: Laufzeiten der Generierungen mit unterschiedlichen Grammatiken und Layouts im Vergleich

auch für andere Anwedungen möchte man Laufzeiten mit dieser Art von Ausmaßen vermeiden. Es ist denkbar, dass dieses Problem mit einer optimierten Implementierung relevanter Teile des *cgashape* Projekts gelöst werden könnte.

5.2.2 Beeinflussende Faktoren

Hier werden kurz einige der Faktoren, die die Laufzeit des Programms beeinflussen, erläutert und dargestellt.

Gebäudehöhe und -größe

Der deutliche Unterschied in den Laufzeiten lässt sich zum einen mit den Gebäudehöhen und -größen erklären. Dass es länger dauert, eine größere Anzahl von Gebäuden zu generieren ist offensichtlich, da mehr Formen erzeugt und dargestellt werden müssen. Das Berliner Tor-Layout hat jedoch weniger Gebäude als das Whit-Widog-Layout. Gleichzeitig sind die Gebäude des Berliner Tor-Layout jedoch um ein Vielfaches größer im Grundriss als die des Whit-Widog-Layout, wodurch die Gebäude größere Fassaden haben, was dazu führt, dass mehr Formen erzeugt und dargestellt werden müssen. Das erklärt den großen Laufzeit-Unterschied zwischen diesen zwei Layouts. Der noch eindeutigere Unterschied zwischen den Laufzeiten der Mittelalter Fantasy- beziehungsweise Antikes Griechenland- und Cyberpunk-Grammatik liegt in der Gebäudehöhe. Während

die Mittelalter Fantasy- und Antikes Griechenland-Gebäude nur bis zu einer Höhe von zehn Metern gehen, können Cyberpunk-Gebäude bis zu 100 Meter hoch sein, wodurch wiederum mehr Formen darzustellen sind.

Details

Der wichtigere Faktor sind aber die Details, welche ein Gebäude aufweist, denn ohne die Details würde auch die Höhe die Laufzeit der Generierung eines Gebäudes nicht beeinflussen, da ohne Details immer die gleiche Anzahl Formen darzustellen sind, nämlich nur die Grundformen für den Körper und das Dach. So beeinflusst die Höhe eines Gebäudes nur indirekt die Laufzeit der Generierung, da ein Gebäude im Allgemeinen mehr Details hat, wenn es höher ist. Je mehr Details ein Modell am Ende aufweist, desto länger dauert die Generierung, da mehr Formen zu generieren und darzustellen sind. Dies erklärt den Unterschied zwischen den einzelnen Grammatiken weiter, da in der Reihenfolge Antikes Griechenland, Mittelalter Fantasy, Cyberpunk die Anzahl an Details pro Gebäude steigt, was im Falle von Cyberpunk durch die Höhe der Gebäude noch verstärkt wird.

Das lässt sich auch anhand eines anderen Beispiels zeigen. Für die Anwendung dieser Arbeit wurde in die Mittelalter Fantasy Grammatik die Erzeugung einer Mauerstruktur eingebaut. Die Mauern der Türme aus diesem Thema sollten eine Struktur aufweisen, die sie wie in Abbildung 5.8 wie eine grobe Steinmauer aus großen Mauersteinen aussehen lässt. Der große Unterschied in der Laufzeit von 2071 Millisekunden für den schlichten Turm zu 124821 Millisekunden für den strukturierten Turm macht deutlich, welchen Unterschied die Anzahl an Details bewirkt. Deshalb wurde diese Funktion in der Grammatik vorerst deaktiviert.

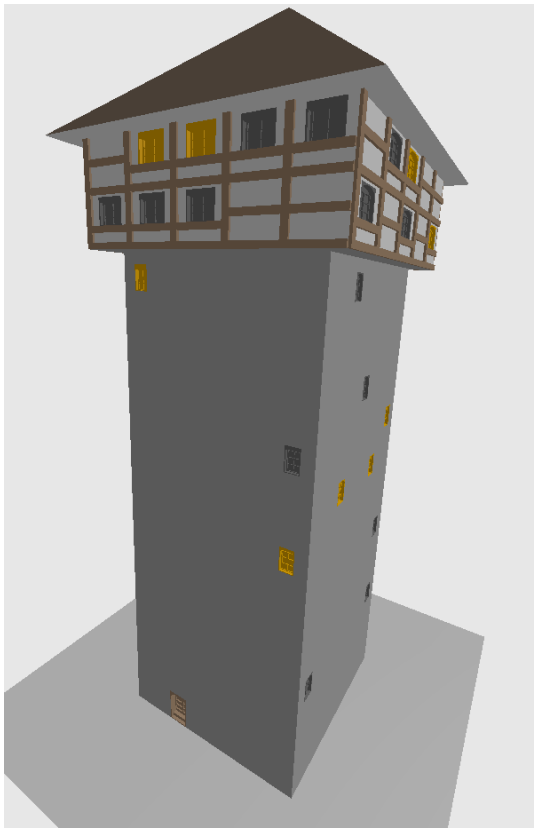
5.3 Zusammenfassung

Die Auswertung der Ergebnisse dieser Arbeit soll hier zusammengefasst werden. Folgende Anforderungen aus Abschnitt 3.1 wurden eingehalten:

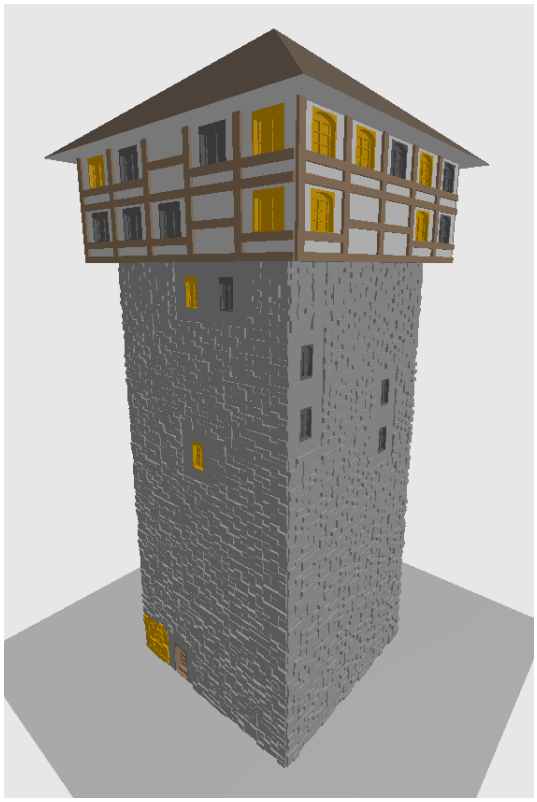
- Die definierten Dateiformate für die Programmeingabe können in das Programm eingegeben werden und führen zu einem Ergebnis.

- Die Ausgabe des Programms ist ein 3D-Modell einer Gruppe von Gebäuden des gewählten Themas, dessen Lage und Ausdehnung mit Einschränkungen dem eingegebenen Layout entsprechen.
- Mehrere nacheinander mit dem gleichen Layout und der gleichen Grammatik erzeugte Modelle unterscheiden sich voneinander.
- Die festgelegten Charakteristika für Gebäude der drei Themen werden von jeweils der zugehörigen Grammatik eingehalten.

Es wurde erarbeitet, dass sich bestimmte Layouts besser für bestimmte Grammatiken eignen und umgekehrt. Die Laufzeit detaillierterer Grammatiken wie Cyberpunk hat sich als ein Problempunkt herausgestellt, da die sehr langen Zeiten die Nutzung dieser Anwendung stark einschränkt.



(a) Modell eines schlichten Turms



(b) Der gleiche Turm mit strukturierter Mauer

Abbildung 5.8: Vergleich eines schlichten Turms mit einem strukturierten Turm der Mittelalter Fantasy Grammatik

6 Fazit

In dieser Arbeit über Prozedurale Content Generierung, Computergrafik und Shape Grammars wurde untersucht, ob und wie gut man aus Grundrissen Gebäude-Gruppen variabler Themen erzeugen kann. Dazu wurden die Grundlagen der genannten Themen erläutert und der Stand der Technik betrachtet. Weiterhin wurde ein Prototyp, der die oben genannte Aufgabe durchführen können soll konzeptioniert, implementiert und evaluiert. Es hat sich herausgestellt, dass man mit Mitteln der PCG und der Computergrafik ohne viel Aufwand im Vergleich zu anderen Lösungen detaillierte Modelle von kleinen bis großen, Thema-variablen Gebäudegruppen auf Basis von vorhandenen Grundrissen erzeugen kann. Weiterhin hat sich gezeigt, dass es auch Probleme gibt, die es zu lösen gilt, wie Grundrisse, die nicht zu allen Themen passen und eine ungenügende Performance. Für die entdeckten Probleme wurden Lösungsansätze vorgeschlagen, sodass das Potenzial dieser Art von Anwendung weiter ausgebaut werden kann. So wäre es möglich, das Problem der unpassenden Grundrisse zu verbessern, indem beispielsweise zu große Grundrisse mehrere Gebäude erhalten. Mit den erwähnten Verbesserungen wäre es auch möglich, dass mit einer besseren Performance noch detailliertere Gebäude erzeugt werden können, wie leuchtende Werbetafeln und große Antennen an den dystopischen Cyberpunk-Wolkenkratzern, Türme mit groben Steinwänden und mittelalterliche Wohnhäuser mit Schornsteinen sowie ein detailliertes Inneres antiker, griechischer Tempel mit Altären und Statuen. Aufbauend auf diesem Projekt wären verschiedene Schritte zur Weiterentwicklung denkbar. So könnten Methoden entwickelt werden, mit denen für Gebäude des Mittelalter Fantasy Themas komplexere Grundrisse ermöglicht werden. Es könnte mehr Variation erschaffen werden, indem, wenn es passt, Formen und Details unregelmäßiger gesetzt werden oder mehr Details, wie neue Fenster- oder Türarten, implementiert werden. Die Umgebung der Gebäude könnte ebenfalls prozedural generiert werden. Die Eingabeformate der Anwendung dieses Projektes lassen auch Informationen zu der Lage von Straßen, Stadtmauern oder anderen Objekten zu beziehungsweise beinhalten diese bereits. Mit diesen Informationen lassen sich 3D-Modelle für diese Objekte einer Stadt erzeugen. Zusätzlich könnten prozedural Vegetation oder andere Details an

Positionen erzeugt werden, welche nicht von anderen Objekten belegt sind. Es wäre also denkbar auf Basis dieser Arbeit prozedural komplette, detaillierte Stadtmodelle zu erzeugen.

Abschließend ist zu sagen, dass die Prozedurale Content Generierung im Feld der Computergrafik ein großes Potenzial hat, welches in Zukunft bestimmt noch weiter ausgebaut und genutzt werden wird und gleichzeitig viele Möglichkeiten zur weiteren Forschung bietet.

Literaturverzeichnis

[1] Aichholzer, Oswin ; Aurenhammer, Franz ; Alberts, David ; Gartner, Bernd: A Novel Type of Skeleton for Polygons. In: *Journal of Universal Computer Science* 1 (1995), Nr. 12

[2] Azad, Mir M. ; Barua, Abhik ; Sultana, Shrmin: A Review Analysis Of Ancient Greek Architecture. In: *International Journal of Civil Engineering, Construction and Estate Management* 3 (2015), Nr. 2

[3] Barriga, Nicolas A.: A Short Introduction to Procedural Content Generation Algorithms for Videogames. In: *International Journal on Artificial Intelligence Tools* 28 (2019), Nr. 2

[4] Chazelle, Bernard: *The Polygon Containment Problem.* In: *Advances in Computing Research, Volume 1*, JAI Press Inc., 1983. – ISBN 0-89232-356-6

[5] Christian Wagenknecht, Michael H.: *Formale Sprachen, abstrakte Automaten und Compiler.* Vieweg+Teubner, 2009. – ISBN 978-3-8348-0624-6

[6] Dylla, Kimberly ; Frischer, Bernard ; Mueller, Pascal ; Ulmer, Andreas ; Haegler, Simon: Rome Reborn 2.0: A Case Study of Virtual City Reconstruction Using Procedural Modeling Techniques. In: *Computer Graphics World* (2010)

[7] Ebert, David S. ; Musgrave, F. K. ; Peachey, Darwyn ; Perlin, Ken ; Worley, Steven ; Mark, William R. ; Hart, John C. ; Musgrave, F. K. ; Peachey, Darwyn ; Perlin, Ken ; Worley, Steven: *Texturing and Modeling: A Procedural Approach.* Morgan Kaufmann, 2003. – ISBN 978-1-55860-848-1

[8] Halatsch, Jan ; Kunze, Antje ; Schmitt, Gerhard: Using Shape Grammars for Master Planning. In: *Design Computing and Cognition '08*, 2008

[9] Hendrikx, Mark ; Meijer, Sebastiaan ; Velden, Joeri Van D. ; Iosup, Alexandru: Procedural content generation for games: A survey. In: *ACM Transactions on Multimedia Computing, Communications and Applications* 9 (2013), Nr. 1

[10] Michael Bender, Manfred B.: *Computergrafik: Ein anwendungsorientiertes Lehrbuch.* Carl Hanser Verlag, 2006. – ISBN 3-446-40434-1

[11] Mueller, Pascal ; Wonka, Peter ; Haegler, Simon ; Ulmer, Andreas: Procedural modeling of buildings. In: *ACM Transactions on Graphics* 25 (2006), Nr. 3

[12] Noor Shaker, Mark J. N.: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer, 2016. – ISBN 978-3-319-42714-0

[13] Politopoulos, A. ; Mol, A.A.A. ; Boom, K.H.J. ; Ariese, C.E.: "History Is Our Playground": Action and Authenticity in Assassin's Creed: Odyssey. In: *Advances in Archaeological Practice* 7 (2019), Nr. 3

[14] Ruiz-Montiel, Manuela ; Boned, Javier ; Gavilanes, Juan ; Jiménez, Eduardo ; Mandow, Lawrence ; Cruz, José-Luis P. de-la: Design with shape grammars and reinforcement learning. In: *Advanced Engineering Informatics* 27 (2013), Nr. 2

[15] Stiny, G: Introduction to Shape and Shape Grammars. In: *Environment and Planning B* 7 (1980), Nr. 3

[16] Stiny, George ; Gips, James: Shape Grammars and the Generative Specification of Painting and Sculpture. In: *Information Processing 71* (1972)

[17] Vollborn, Constanze: *Verziertes Fachwerk in Lüneburg*, Christian – Albrechts – Universität zu Kiel, Dissertation, 2007

[18] Wang, Xiao ; Song, Yacheng ; Tang, Peng: Generative urban design using shape grammar and block morphological analysis. In: *Frontiers of Architectural Research* 9 (2020), Nr. 4

[19] Yoav I. H. Parish, Pascal M.: Procedural modeling of cities. In: *SIGGRAPH01: The 28th International Conference on Computer Graphics and Interactive Techniques* (2001)

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
Dall-E 3 im Bing Image Creator von Microsoft Designer	Erzeugung eines Bildes zur Veranschaulichung des Themas Cyberpunk
Medieval Fantasy City Generator von watabou auf itch.io	Erzeugung von Grundrissen für Gebäude Gruppen
QGIS und OpenStreetMap	Export von Geodaten für Grundrisse von Gebäude Gruppen

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original