HAW HAMBURG

BACHELOR THESIS
Michael Dornhof

Prozedurale Generierung von 3D Fahrzeugmodellen mit einer Form Grammatik

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

Michael Dornhof

Prozedurale Generierung von 3D Fahrzeugmodellen mit einer Form Grammatik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
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: 18. Mai 2023

Michael Dornhof

Thema der Arbeit

Prozedurale Generierung von 3D Fahrzeugmodellen mit einer Form Grammatik

Stichworte

Prozedurale Contentgenerierung, Computer-Grafik, 3D, Fahrzeuge, Form Grammatik

Kurzzusammenfassung

Mittels eines grammatikalischem Ansatzes zur Prozeduralen Contentgenerierung soll ein Prototyp entwickelt werden, welcher die Generierung von 3D-Fahrzeugmodellen ermöglicht. Dieser soll mittels Zufallswertern agieren, um so Varianz zwischen den Fahrzeugen zu bieten. Darüber hinaus ist vorgesehen, dass die Evaluation der Grammatik parametrisierbar ist, so dass der Nutzer das Resultat beeinflussen kann.

Michael Dornhof

Title of Thesis

Procedural generation of 3D vehicle-models using a shape grammar

Keywords

Procedural contentgeneration, Computer-graphics, 3D, Vehicles, Shape Grammar

Abstract

Using a grammatical approach for procedural content generation, a prototype is to be developed that generates 3D vehicle models. This prototype will operate using randomness to provide variation among the resulting vehicle models. Additionally, the possibility of parameterizing the evaluation of the grammar is intended to be implemented so that the user is able to influence the outcome of the resulting models.

Inhaltsverzeichnis

Abbildungsverzeichnis **vi**

Tabellenverzeichnis **viii**

1 Einleitung **1**

2 Grundlagen **3**
2.1 Computergrafik . . . 3
2.1.1 Polygonale Netze . . . 4
2.2 Formale Grammatik . . . 5
2.3 Prozedurale Contentgenerierung . . . 6
2.3.1 Such-Basierter Ansatz . . . 7
2.3.2 Konstruierende Ansätze . . . 8
2.3.3 Grammatik basierende Ansätze . . . 9
2.4 Shape Grammar . . . 11

3 Stand der Technik **13**
3.1 CGA Shape: Generierung von Gebäuden . . . 13
3.2 Generierung von Cross-Over Fahrzeugen . . . 15

4 Konzept **17**
4.1 Ansatz . . . 17
4.1.1 Implementierungsansatz . . . 18
4.1.2 Aufbau Fahrzeug . . . 18
4.2 Funktionale Anforderungen . . . 21
4.2.1 Variationen . . . 21
4.2.2 Realismus . . . 21
4.3 Nicht-Funktionale Anforderungen . . . 21

5 Implementierung **23**
5.1 Einführung . . . 23
5.2 Architektur . . . 23
5.2.1 Einordnung der Abhängigkeiten . . . 23
5.2.2 Grammatik . . . 25
5.2.3 ShapeTree . . . 26
5.2.4 MeshGenerator . . . 27
5.3 Modifikationen der Grammatik-Syntax . . . 27
5.3.1 Setup Variables . . . 28
5.3.2 Statische Evaluation von Zufallswerten . . . 29
5.4 Operatoren und Shapes . . . 30
5.4.1 duplicate-Operator . . . 30
5.4.2 ellipse-Operator . . . 31
5.4.3 repeated_split-Operator . . . 31
5.4.4 square-Operator . . . 32
5.4.5 padding-Operator . . . 32
5.4.6 vehiclecabin-Operator und VehicleCabin-Shape . . . 34
5.4.7 vehicle_base-Operator . . . 34
5.5 Realisierung der Grammatik . . . 35

6 Evaluation **40**
6.1 Ergebnisse . . . 40
6.2 Anforderungen . . . 42
6.3 Metriken . . . 44

7 Fazit und Ausblick **45**

Literaturverzeichnis **47**
Selbstständigkeitserklärung . . . 49

Abbildungsverzeichnis

1.1 Exemplarische Generierung von fünf verschiedenen Fahrzeugmodellen mittels des Implementierten Ansatzes 2

2.1 Bestandteile von Polygonalen Netzen 4
2.2 Visualisierung von Dreickecksnetzen und Verfeinerung der runden Geometrie mittels kleinerer Dreiecke [16]. 5
2.3 Erzeugung einer Spielfläche mittels Space-Partitioning unter Verwendung einer Quadtree Datenstruktur [12] 9
2.4 Beispiel eines einfachen L-Systems zur Anwendung einer Turtle Graphic [12] 11
2.5 Beispiel eines erweiterten L-Systems zur Darstellung einer Pflanzen ähnlichen Grafik [12] 11
2.6 Beispiel Shape-Rules (a) und einer initialen Shape (b) [14] 12
2.7 Ausführung der Shape Rules aus Abbildung 3.1 auf die initiale Shape [14] 12

3.1 Darstellung der geometrischen Eigenschaften einer Shape [7] 14
3.2 Beispiele von prozedural generierten Modellen mittels CGA Shape. Links Pompeii, Rechts generisches Gebäude [7]. 15
3.3 Regel 4 (Front, Side, Rear) [9] 16
3.4 Prozedural erzeugtes Cross-Over-Vehikel [9] 16

4.1 Fahrzeugklassen - Links Tesla Cybertruck (SUV), Mitte 3D Modell eines VW Golf 2 (Kleinwagen), Rechts VW T4 (Transporter) 17
4.2 Hierarchie der aufeinander aufbauenden Fahrzeugkomponenten 18
4.3 Verwendete Dimensionen eines VW T4s 20
4.4 Verwendete Dimensionen eines Tesla Cybertrucks 20
4.5 Verwendete Dimensionen eines VW Golf 2 20

5.1 Komponentendiagramm der Grundarchitektur des zu betrachtenden Unterprojekts 24

5.2 Klassendiagramm im Kontext des Auswertens einer Grammatik-Datei . . 25
5.3 Klassendiagramm im Kontext des Generierens eines ShapeTrees 26
5.4 Klassendiagramm im Kontext des Generierens eines ShapeTrees 27
5.5 Setup Variable „SCALE_VARIABLES“ am Beispiel zweier Grammatiken 28
5.6 Statische Evaluation am Beispiel zweier Grammatiken 29
5.7 Duplicate-Operator am Beispiel einer Grammatik 30
5.8 ellipse-Operator am Beispiel zweier Grammatiken. Links mit 10 Vertices Präzision, Rechts mit 100 Vertices 31
5.9 repeated_split-Operator am Beispiel zweier Grammatiken. Oben mittels ursprünglicher Zuordnung, unten mittels wiederholender Zuordnung 32
5.10 padding-Operator am Beispiel einer Grammatik 33
5.11 Algorithmus des padding-Operators am Beispiel eines Pentagons 33
5.12 Darstellung des entstehenden Modells des VehicleCabin-Operator 34
5.13 Fahrzeug-Rumpf und Reifen 37
5.14 Fahrerkabine und Fenster (Tesla Cybertruck, VW Golf 2, VW T4) 37
5.15 Exemplarische Generierungen der Front eines Tesla Cybertrucks, VW Golf 2, VW T4 38
5.16 Exemplarische Generierungen der verschiedenen Variationen von einem Auspuff 39

6.1 Drei exemplarische Generierungen der VW Golf 2 Inspiration 41
6.2 Drei exemplarische Generierungen der VW T4 Inspiration 41
6.3 Drei exemplarische Generierungen der Tesla Cybertruck Inspiration 42
6.4 Hierarchischer Aufbau der Komponenten zuzüglich der Anzahl der implementierten Variationen 43

Tabellenverzeichnis

6.1 Zeitmessungen verschiedener Operationen und Generierungen. Werte ergeben sich jeweils aus fünf Ausführungen 44

1 Einleitung

Die prozedurale Contentgenerierung erweist sich als ein äußerst aufregender Forschungsbereich der Computergrafik und der Künstlichen Intelligenz. Dabei ermöglicht die prozedurale Generierungen von Inhalten, realistische und vielfältige 3D-Modelle ohne stetig manuelles entwerfen dieser. Lediglich die Voraussetzungen sind einmalig zu schaffen, so dass das System basierend auf diesem Ausgangspunkt effizient verwendet werden kann. Sind diese Voraussetzungen geschaffen, so besteht die Möglichkeit, das System in verschiedenen Anwendungsgebieten anzuwenden, wie in etwa Computerspielen, Simulationen oder VR-/AR-Applikationen.

Die herkömmlichen Modellierungstechniken von 3D-Modellen erweisen sich als äußerst Zeitaufwendig und sind somit mit hohen Kosten verbunden, insbesondere in Hinsicht auf den Trend und dem Wunsch nach stetig höherer Komplexität und detailreicheren 3D-Modellen. Zudem besteht die Gefahr von Duplikaten und ein Mangel an Varianz zwischen generierten 3D-Modellen im Fall einer manuellen Generierung. Ein prozeduraler Ansatz bietet hierbei eine vielversprechende Lösung und ersetzt im besten Fall viele manuelle Schritte gänzlich. Dies ermöglicht relativ simpel, mittels voneinander verschiedener Verfahren, Varianz in der Generierung. Mittels entsprechender Verfahren ist es so möglich, Fahrzeugmodelle automatisch zu erzeugen, wobei Form, Details und Variationen kontrolliert werden können.

Ziel dieser Arbeit ist das Entwickeln eines Prototyps zur prozeduralen Generierung von 3D-Fahrzeugmodellen unter Verwendung eines regelbasierten Ansatzes, um die Frage zu beantworten, inwieweit ein solcher prozedurale Ansatz sinnvoll im Kontext der Fahrzeugmodelle ist. Das regelbasierte Verfahren setzt sich durch verschiedene Regeln zusammen, welche iterativ aufeinander aufbauen und somit unter sinnvoller Verknüpfung komplexe Modelle generieren können. Zusätzlich soll der Anwender in der Lage sein, dass generierende System zu parametrisieren, so dass diese Parameter direkten Einfluss auf die Generierung der 3D-Modelle haben, um so unterschiedliche Stile und Fahrzeugtypen zu

ermöglichen.

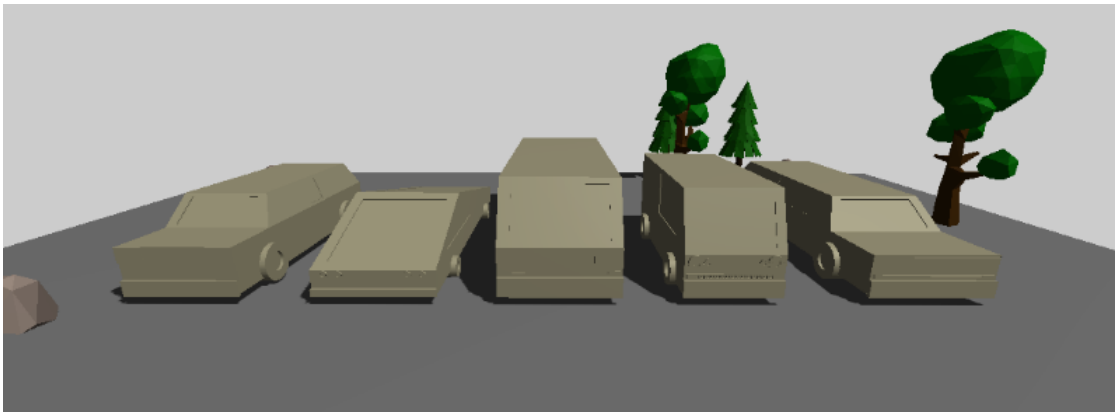


Abbildung 1.1: Exemplarische Generierung von fünf verschiedenen Fahrzeugmodellen mittels des Implementierten Ansatzes

Die folgende Arbeit ist in sechs unterschiedliche Kapitel gegliedert. Zunächst werden im Kapitel „Grundlagen“, benötigte grundlegende Verfahren beleuchtet und grob beschrieben. Weiterführend werden bestehende Projekte und Wissenschaftliche Arbeiten im Kontext der prozeduralen Contentgenerierung unter Verwendung eines Grammatikalischen Ansatzes im Kapitel „Stand der Technik“ betrachtet. Diese dienen als Inspiration und Basis der eigenständigen Forschung. Aufbauenden auf diesen Konzepten, wird ein eigenes Konzept im gleichnamigem Kapitel „Konzept“ vorgeschlagen, welches Anforderungen und Verfahren einführt. Schließlich werden die einzelnen Elemente des Konzepts in der „Implementierung“ umgesetzt und die entsprechenden Ergebnisse final im abschließendem Kapitel „Evaluation“ ausgewertet, um der Frage der Sinnhaftigkeit nachzukommen.

2 Grundlagen

2.1 Computergrafik

Der Begriff der Computergrafik beschäftigt sich mit der grafischen Darstellung von digitalen Szenen im zwei- oder dreidimensionalem Raum. Dies kann einfache statische Bilder umfassen, aber auch die Darstellung von komplexen dreidimensionalen Objekten und Modellen. Dabei haben die 2D- und 3D-Szenen gemeinsam, dass grundlegend sämtliche resultierenden Bilder auf einer Pixel Ebene ausgegeben werden. Im Kontext von 3D-Szenen ist dies ausschlaggebend für den Aufwand in Hinsicht der Rechenleistung, denn hier muss eine komplexe abstrakte Datenstruktur so heruntergebrochen werden, dass diese samt ihrer Schatten, Lichteinfälle, Tiefen, etc. als 2D-Bild darstellbar ist. Diese Arbeit bezieht sich auf den 3D Aspekt der Computergrafik, welche teils weiterführende Grundlagen gegenüber denen der 2D-Computergrafik voraussetzt. Im Folgenden werden relevante, benötigte Grundlagen zur prozeduralen Generierung von 3D-Fahrzeugen weiterführend geklärt.
Man unterscheidet bei der Computergrafik zwischen der interaktiven Computergrafik, auch Echtzeit Computergrafik und der Nichtechtzeit-Computergrafik. Essenziell ist hierbei, dass die interaktive Computergrafik, wie der Name schon sagt, eine Interaktion zwischen dem Benutzer und dem System ermöglicht. So können verschiedene Simulationen oder eine dynamisch steuerbare Kamera realisiert werden. Dies setzt eine äußerst performante Berechnung der einzelnen Szenen voraus, da der Benutzer eine möglichst flüssige Ausgabe erhalten soll. Um diese hohe Bildwiederholrate gewährleisten zu können, müssen hierfür häufig Abstriche hinsichtlich der Detailtiefe und dem Realismus erfolgen. Hingegen setzt die Nichtechtzeit-Computergrafik auf maximale Detailtiefe und Realismus. Hier spielt es grundsätzlich keine große Rolle, wie schnell das System das Bild generiert, da der Benutzer dieses nicht unmittelbar wahrnehmen muss. Ein Beispielfall einer solchen Computergrafik stellen unter anderem die Animationsfilme dar [8].

2.1.1 Polygonale Netze

Dreidimensionale Objekte werden in der Computergrafik anhand ihrer Oberflächen dargestellt, welche eine Datenstruktur voraussetzt, die dieses Vorhaben effizient und zuverlässig realisieren kann. Der meist verbreitetste Ansatz für eine solche Datenstruktur ist das Triangle-Mesh (dt. Dreickecksnetz) [5]. Ein Triangle-Mesh besteht aus einer zusammenhängenden Menge von aufeinander aufbauenden Komponenten in einem 3D-Raum. Grundlage für jegliche 3D-Objekte sind Punkte (auch Knoten oder Vertex), die mittels Kanten verknüpft werden. Drei Punkte, die mittels entsprechender Kanten miteinander Verbunden werden stellen ein Dreieck dar, welches auch spezifischer Facette genannt wird. Mehrere Facetten, welche miteinander verknüpft sind, bilden ein Dreiecksnetz/Polygonnetz.

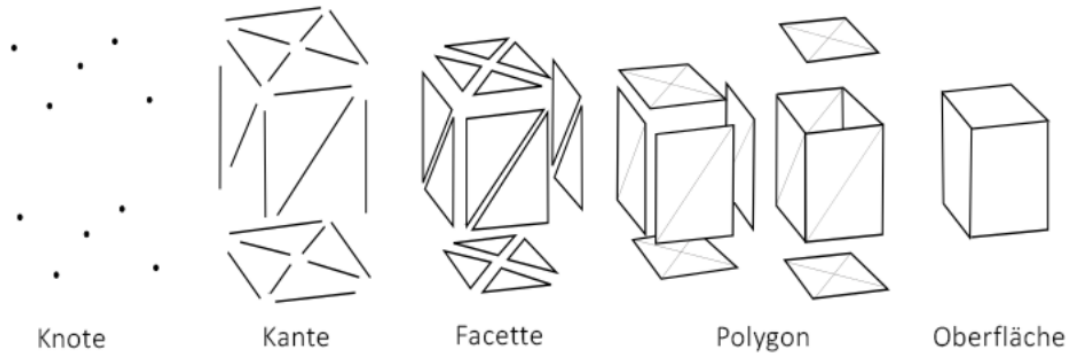


Abbildung 2.1: Bestandteile von Polygonalen Netzen

Polygonale Netze sind in der Lage alle geometrischen Formen dreidimensional darzustellen oder können sich zumindest den Formen annähern. Runden Formen, wie in etwa einer Kugel, kann sich daher durch entsprechend kleine Facetten lediglich angenähert werden. Dabei gibt die Größe der einzelnen Facetten den Detailgrad der runden Form an [1]. So ist eine deutlich rundere Geometrie in Abbildung 2.2 (c) zu entnehmen als in Abbildung 2.2 (a).

Abbildung 2.1: Commons, Wikimedia: Mesh overview. 2009. commons.wikimedia.org; abgerufen am 17.11.2022

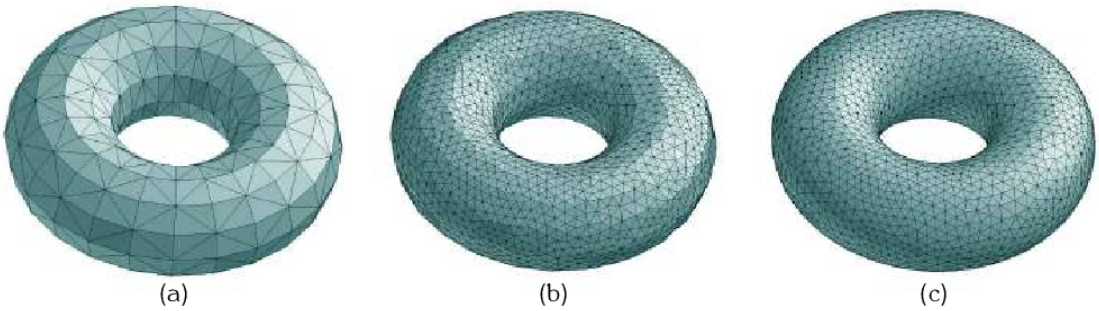


Abbildung 2.2: Visualisierung von Dreickecksnetzen und Verfeinerung der runden Geometrie mittels kleinerer Dreiecke [16].

Eine weitere wichtige Eigenschaft von Polygonalen Netzen sind Normalvektoren. Diese stellen einen Richtungsvektor dar, der orthogonal zu einer Oberfläche steht und somit die Richtung beschreibt, zu der eine Oberfläche gerichtet ist. Diese Eigenschaft hat eine große Bedeutung für verschiedene Aspekte, wie dem Anwenden eines Beleuchtungsmodells [4].

2.2 Formale Grammatik

Formale Grammatiken wurden erstmals von dem Linguisten Noam Chomsky in den 1950er Jahren beschrieben und verfolgen das Ziel natürliche Sprache zu modellieren [2]. Dieses Modell ist allerdings nicht nur in den Sprachwissenschaften nützlich, sondern findet auch große Anwendung in der Informatik. Der Kerngedanke der von N. Chomsky definierten Grammatiken stellt ein Konzept dar, welches Wörter bildet, indem ein Symbol auf ein oder mehrere darauf folgende Symbole ableitet. Dieses Ableiten von Symbolen geschieht iterativ, bis das resultierende Wort vollständig durch Terminalsymbole beschrieben ist, sprich keine weitere Regel auf den Symbolen des Wortes anwendbar ist.
Man unterscheidet bei Grammatiken zwischen Regulär, Kontextsensitiv und Kontextfrei. Der Rahmen dieser Arbeit beschränkt sich jedoch auf die Kontextfreien Grammatiken, so dass lediglich diese hier näher betrachtet werden.
Bei kontextfreien Grammatiken handelt es sich, um eine äußerst mächtige Sprachfamilie, die nicht nur in der Theorie Anwendung findet. So setzt sich nahezu jede höhere Programmiersprache technisch aus einer kontextfreien Grammatik zusammen, indem der Parser auf Basis einer Eingabe mittels einer kontextfreien Grammatik die Bedeutung des Codes extrahiert und schließlich diesen kompiliert oder interpretiert [13].

Kontextfreie Grammatiken zeichnen sich dadurch aus, dass deren Produktionsregeln nicht abhängig eines Kontexts sind und somit stets Produktionsregeln von einem einzigen Nonterminalsymbol ausgehen. Kontextfreie Grammatiken lassen sich wie folgt formal definieren [3]:
Eine kontextfreie Grammatik stellt ein Vier-Tupel der Form (Σ, V, S, P) dar, mit:

- Σ: Endliche Menge von Terminalsymbolen
- V: Endliche Menge von Nonterminalsymbolen mit $V \cap \Sigma \emptyset$
- S: Einem Startsymbol $S \in V$
- P: Einer endlichen Menge von Produktionsregeln

Weiterführend unterscheidet man eine Grammatik zwischen deterministisch und nichtdeterministisch. Eine deterministische Grammatik setzt voraus, dass zu jedem Nonterminalsymbol exakt eine Regel existiert, in der das Nonterminalsymbol auf der linken Seite der Regel steht, sprich ein Nonterminalsymbol wird stets zu einem identischen Resultat abgeleitet. Hingegen kann eine nichtdeterministische Grammatik ein Nonterminalsymbol mittels verschiedenerer Produktionsregeln, in denen das entsprechende Nonterminalsymbol auf der linken Seite der Regel vorkommt, auf verschiedene Resultate ableiten [12]. Die Auswahl einer Produktionsregel bei nichtdeterministischen Grammatiken kann entweder zufällig oder nach Wahrscheinlichkeiten erfolgen. Besonders unter Berücksichtigung von prozeduraler Contentgenerierung, haben nichtdeterministische Grammatiken den Vorteil, Varianz in das Endresultat eines Eingabeworts zu bringen und sind daher besonders attraktiv.

2.3 Prozedurale Contentgenerierung

Julian T., Emil. K, David S und Georgios Y. definierten die prozedurale Contentgenerierung im Rahmen der Spieleentwicklung als „algorithmical creation of game content with limited or indirect user input“ [17] (dt. Algorithmische Kreierung von Spieleinhalten mit eingeschränkter oder indirekter Benutzereingabe). Ausgehend von dieser Definition beschäftigt sich die prozedurale Contentgenerierung mit der computergesteuerten Generierung von Inhalten anhand einer Vielzahl von Einsatzmöglichkeiten. Im Kontext der Spieleentwicklung kann sich dies über die Generierung von 3D-Modellen, wie Gegenständen, Fahrzeugen, Charakteren und vielem mehr beziehen. Auch grundlegendere Elemente

eines Spieles, wie ein Spiele-Level lassen sich meist vollständig prozedural generieren, so dass der Spieler nach jedem Start des Spieles ein neues, prozedural generiertes Level zu spielen hat. Ein bekanntes Beispiel eines Spieles, welches von der prozeduralen Contentgenerierung profitiert ist „Rogue“ (1980, „Epyx, Inc“), welches dessen Level prozedural generiert. Auch aktuelle Spiele wie „Minecraft“[1] setzen auf prozedurale Contentgenerierung. Bei Minecraft handelt es sich um ein Open-World-Spiel, in dem die gesamte Welt und dessen Inhalt zur Laufzeit prozedural generiert wird.

Bei dem Begriff der prozeduralen Contentgenerierung handelt es sich um einen Oberbegriff für verschiedene Algorithmen und Ansätze. Die folgenden Unterkapitel beleuchten grob einige Ansätze die für die prozedurale Contentgenerierung Anwendung finden.

2.3.1 Such-Basierter Ansatz

Der Such-Basierte Ansatz zur prozeduralen Contentgenerierung verwendet einen iterativen Algorithmus, welcher aus einer Vielzahl von möglichen Resultaten, das am besten geeignete auswählt. Dabei besteht der Such-Basierte Ansatz aus drei fundamentalen Komponenten [12].

- Such-Algorithmus: Der Such-Algorithmus stellt den Kern des Ansatzes dar und beschreibt, wie das beste Resultat ausgewählt wird. Häufig wird hierfür ein evolutionärer oder stochastischer Ansatz angewandt.
- Darstellung des Inhalts: Die Darstellung des generierten Objekts muss in eine entsprechende Art übersetzt werden, so dass dieser vom Algorithmus interpretierbar und auswertbar ist. Dies kann im Fall eines Labyrinths, eine Matrix mit einem binären Wert in den einzelnen Zellen sein, welche die Pfadwege des Labyrinths darstellen.
- Evaluationsfunktion: Das generierte Resultat wird mittels verschiedener Metriken bewertet, um so schließlich das bestmögliche Resultat zu erzielen. Im Fall des Labyrinths kann dies die Komplexität (Anzahl an Verzweigungen, minimale Länge, Anzahl an Zyklen, uvm.) beschreiben, aber auch ob ein generiertes Labyrinth überhaupt abschließbar ist.

[1] Spiel: Minecraft - https://www.minecraft.net/; (erstmals von Mojang in 2011 veröffentlicht)

Evolutionärer Such-Algorithmus

Der evolutionäre Algorithmus hat seinen Ursprung in der von Darwin aufgestellten Evolutionstheorie. Hier wird nach dem Prinzip der natürlichen Selektion vorgegangen. Kern ist das Generieren einer Population, in der die Individuen mittels der Evaluationsfunktion gefiltert werden. Individuen die schlecht bewertet werden, werden fallengelassen und Individuen die eine gute Bewertung erzielen, werden für die nachfolgende Population mutiert. Das Mutieren geschieht dabei insofern, dass entsprechende Eigenschaften eines Individuums zufällig minimal verändert werden. So wird im Durchschnitt jede nachfolgende Generation von der Evaluationsfunktion besser bewertet als die vorangehende Generation, bis schließlich eine gewünschte Bewertung erreicht wurde oder die maximale Anzahl von Generationen überschritten wird [12].

Stochastischer Such-Algorithmus

Abhängig davon, wie komplex das Ziel der prozeduralen Contentgenerierung ist, kann auch ein stochastischer Such-Algorithmus angewandt werden. Rückführend auf das Beispiel des Labyrinths, kann ein stochastischer Ansatz z.B. eine beliebige Anzahl an zufällig generierten Labyrinthen erzeugen und aus dieser Menge das bestbewertete Labyrinth auswählen.

2.3.2 Konstruierende Ansätze

Im Kontext der prozeduralen Contentgenerierung versteht man unter den konstruierenden Ansätzen (engl. Constructive Approaches), jene die meist in einer festen Zeit laufen und die Ausgabe nicht evaluieren, um diese möglicherweise zu verwerfen und neu zu generieren. Dieser Ansatz findet häufig Anwendung in sogenannten Dungeon Spielen, wo dem Spieler zwar eine Open-World ähnliche Erfahrung geboten wird, aber vom Spiel bzw. vom PCG Algorithmus feste Wege generiert werden [12].

Space-Partitioning Ansatz

Einen Algorithmus der konstruierenden Ansätze stellt das Space-Partitioning dar. Dabei teilt der Algorithmus Bereiche rekursiv auf, welche final miteinander verknüpft werden. So können verschiedene Räume prozedural erzeugt werden, welche mit Wegen zueinander

verbunden werden. Häufig wendet man hier das Prinzip von Quadtrees für 2D-Szenen oder auch Octrees für 3D-Szenen an. Zur Generierung eines 2D-Dungeons kann so die gesamte Spielfläche als Baum in Form eines Quadtrees interpretiert werden. Jeder partitionierte Knoten des Quadtrees besitzt vier weitere Kindsknoten, die eine bestimmte Fläche in der Gesamtfläche darstellen.

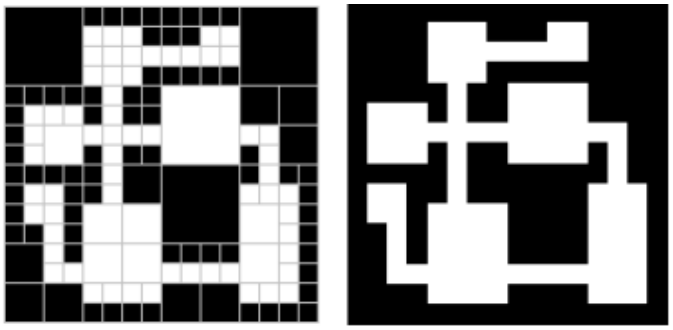


Abbildung 2.3: Erzeugung einer Spielfläche mittels Space-Partitioning unter Verwendung einer Quadtree Datenstruktur [12]

Ein beispielhafter Algorithmus zur Erzeugung einer Spielfläche eines Dungeon-Spiels könnte so aussehen, dass die gesamte Spielfläche als Wurzel des Octrees interpretiert wird. Daraufhin wird beliebig häufig, bis eine Abbruchbedingung erfüllt ist, ein Blattknoten ausgewählt und in vier weitere Blattknoten partitioniert, welche alle einen Teilraum des ursprünglichen Blattknotens darstellen. Ist die Abbruchbedingung erfüllt und somit die Partitionierung des Quadtrees abgeschlossen, so werden alle Blattknoten zufällig als Raum oder leere Fläche definiert und miteinander verbunden. Dieses Vorgehen ermöglicht es Räume unterschiedlicher Größe und ohne Überdeckungen prozedural aufzubauen [12].

2.3.3 Grammatik basierende Ansätze

Grammatiken, welche bereits in Kapitel 2.2 grob beleuchtet wurden, finden auch in der Computergrafik einen häufigen Einsatz, insbesondere dann, wenn diese eine Rolle zur prozeduralen Contentgenerierung einnehmen. Mittels dieser lassen sich viele repetitive Formen einfach und effizient darstellen.

L-Systeme

Bei den L-Systemen handelt es sich um ein besonderes Beispiel eines Grammatik basierenden Ansatzes. Erstmalig wurde diese Grammatik von A. Lindenmayer im Jahr 1968 vorgestellt, mit dem Ziel ein Modell zu entwickeln, welches das Wachstum von organischen Systemen widerspiegelt [6]. Vorwiegend wurden die L-Systeme entwickelt, um bestehende dynamische Prozesse zu modellieren, entgegen herkömmlicher Grammatiken, welche sich auf Sprache fokussieren. L-Systeme oder auch Lindenmayer-Systeme sind eine spezielle Form von Grammatiken, welche sich dadurch auszeichnen, dass deren Produktionsregeln nicht sequentiell, sondern von einer Iteration zur nächsten parallel ausgeführt werden. Formal definieren lässt sich ein L-System G als 3-Tupel (Σ, R, ω), bestehend aus [11]:

- Σ : Das Alphabet mit sämtlichen verwendeten Symbolen der Grammatik, welches potentiell eine Untermenge C für Konstanten enthält, zu denen keine Regel Anwendung findet.
- R: Die Menge aller Regeln der Grammatik
- ω: Die initiale Zeichenkette, bestehend aus einem Element aus Σ^*

Jede Regel in R besitzt die Form $\alpha A \beta \rightarrow \gamma$ für die gilt:

- $A \in \Sigma$: A ist ein Element aus dem übergeordnetem Alphabet des L-Systems.
- $\alpha, \beta \in \Sigma^*$: α und β beschreiben den optionalen Kontext einer Regel und müssen in der behandelten Zeichenkette in der entsprechenden Reihenfolge vorkommen, um ein Ausführen der Regel zu ermöglichen.
- $\gamma \in \Sigma^*$: beschreibt die resultierende Zeichenkette, mit der A ersetzt wird.

Wird ein L-System so konstruiert, dass eine sogenannte Turtle einer Turtle Graphic (man stelle sich eine Schildkröte vor, die einen Pfad nachzeichnet während die Symbole der Grammatik die Richtung beeinflussen) ermöglicht wird, so kann man mittels einer simplen Grammatik, bestehend aus gerade mal einer Regel, bereits komplexe Formen generieren [10]. Betrachte man das L-System mit den Symbolen/Instruktionen „F“ (Gehe nach vorne), „+“ (Drehe 90 Grad gegen den Uhrzeigersinn) und „-“ (Drehe 90 Grad in Uhrzeigersinn) und der einzigen Regel „$F \rightarrow F + F - F - F + F$“, so erhält man nach drei Iterationen die in Abbildung 2.4 aufgeführte Form [12].

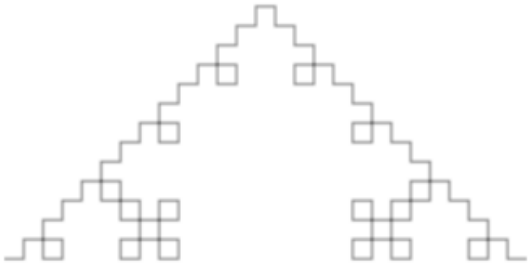


Abbildung 2.4: Beispiel eines einfachen L-Systems zur Anwendung einer Turtle Graphic [12]

Modifiziert man die Richtungswechsel auf 30 Grad, statt 90 Grad und erweitert man dieses L-System um einen Stack mit den Symbolen „[“, welches die aktuelle Position und Orientierung der Turtle speichert und dem Symbol „]“, welches den aktuellsten Wert des Stacks wiederherstellt, so kann man die Regel derart anpassen (z.B. „$F \rightarrow F[-F]F[+F][F]$“), dass bereits erste Pflanzen zu erkennen sind [12].

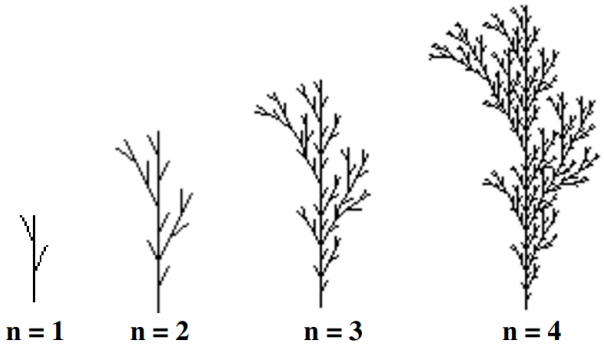


Abbildung 2.5: Beispiel eines erweiterten L-Systems zur Darstellung einer Pflanzen ähnlichen Grafik [12]

2.4 Shape Grammar

Der Begriff der Shape Grammars wurde erstmals in der Arbeit „Shape Grammars and the Generative Specification of Painting and Sculpture“ [15] von James Gips und Goerge Stiny im Jahr 1971 eingeführt, welcher weiter von Goerge Stiny in der Arbeit „Introduction to Shape and Shape Grammars“ (1980) [14] konkretisiert wurde. Der Begriff der Shape Grammars behandelt die Generierung von Formen mittels Grammatiken. So

wird der Begriff der Shapes, als intuitive 2D-Form definiert, welche sich aus einer endlichen Menge von Linien zusammensetzen. Darüber hinaus wird der Begriff der labeled Shapes (im folgenden als Shape benannt) eingeführt, welche aus einer Shape und einem Namen/Symbol bestehen. Eine Subshape „a“ von „b“ ist eine solche Form, deren Linien, vollständig in „b“ vorkommen. Formal definiert G. Stiny eine Shape Grammar, als ein Vier-Tupel der Form:

- S: Endliche Menge von Shapes
- L: Endliche Menge von Symbolen
- R: Endliche Menge von Shape-Rules mit der Form $a \rightarrow b$, wobei a und b Shapes sind
- P: Die Initiale Shape

Eine Shape-Rule wird auf eine Shape angewandt, sofern das Element auf der linken Seite des Pfeils vollständig in der übergeordneten Shape vorkommt. Die Shape auf der linken Seite, wird dann vollständig von der Shape auf der rechten Seite ersetzt.

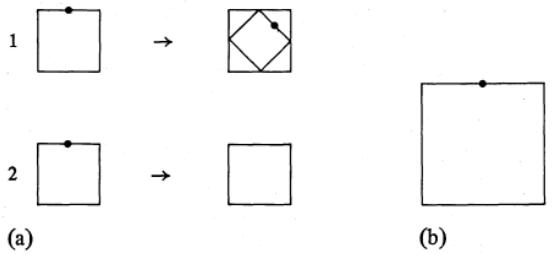


Abbildung 2.6: Beispiel Shape-Rules (a) und einer initialen Shape (b) [14]

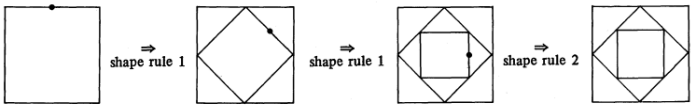


Abbildung 2.7: Ausführung der Shape Rules aus Abbildung 3.1 auf die initiale Shape [14]

Eine Veranschaulichung einer Shape Grammar und deren Produktionen kann der Abbildung 2.6 entnommen werden. Dabei stellt die Abbildung die Produktionsregeln R in (a) und (b) die initiale Shape I der Shape Grammar dar. Die Abbildung 2.7 führt dann die Produktionsregeln der Shape Grammar iterativ aus. Je nach Ansatz zur Auswahl der Regeln, hier zwei mal Regel 1, dann Regel 2, können sich die resultierenden Formen unterscheiden.

3 Stand der Technik

Aufgrund von ausgiebig wachsender Nachfrage an prozedural generierten Modellen, wird stets an weiteren Optimierungen und Konzepten zu diesem Themenbereich der Computergrafik geforscht. Im Folgenden werden einige Ausarbeitungen zum Thema der prozeduralen Contentgenerierung näher betrachtet.

3.1 CGA Shape: Generierung von Gebäuden

Die in der Arbeit „Procedural Modeling of Buildings“ vorgestellte CGA (abk. für Computer Generated Architecture) Shapes ist eine von Peter Wonka, Pascal Müller et al. entwickelte Grammatik, welche auf Shape Grammars basiert, die das prozedurale Generieren von 3D-Modellen ermöglicht. Entwickelt wurde diese Grammatik im Rahmen der Softwareentwicklung des Programms „ArcGIS CityEngine“ der Firma ESRI [1], welches das Ziel verfolgt einen prozeduralen Ansatz zur Stadtgestaltung zu ermöglichen.
Ziel der CGA Shape ist es, mittels einem prozeduralen Ansatz, Geometrie mit hoher visueller Qualität und geometrischem Detail zu generieren. Dies wird erreicht, indem eine geometrische Form iterativ modifiziert wird, bis ein gewünschtes detailreiches Objekt entsteht. Ein allgemeiner Ansatz, um dies mittels einer CGA Shape zu erreichen sieht so aus, dass ausgehend von einer einfachen geometrischen Form ein sogenanntes Mass Model (volumetrisches Objekt) erzeugt wird. Dieses Mass Model wird weiterführend strukturiert und verschiedene Verkettungen von Operationen bringen Details in das entsprechende Objekt [7].
Geometrische Formen (im folgenden Shapes) bestehen dabei aus verschiedenen Eigenschaften. Diese umfassen ein Symbol, welches Nutzung in der aufbauenden Grammatik findet und geometrische Attribute. Die wichtigsten Eigenschaften dieser Attribute sind die Position P im globalen Koordinatensystem, drei Orthogonalen Vektoren X, Y und Z,

[1] ArcGIS CityEngine. https://www.esri.com/de de/arcgis/products/arcgis-cityengine/overview. – Abgerufen: 5.12.2022

die das lokale Koordinatensystem definieren und einen Größen-Vektor S. Diese Attribute stellen eine Bounding-Box, auch Scope genannt, des Objektes dar.

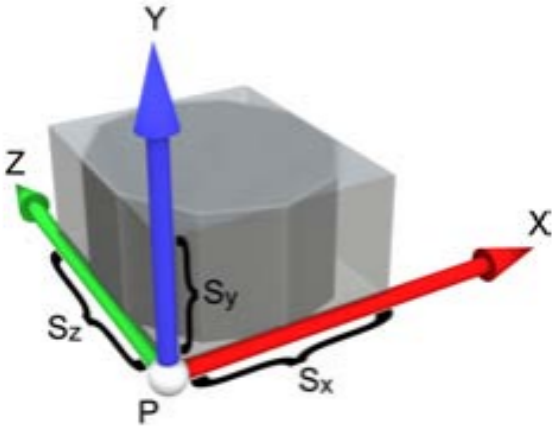


Abbildung 3.1: Darstellung der geometrischen Eigenschaften einer Shape [7]

Im Grunde ist der CGA Shape Ansatz eine Abstraktion einer kontextfreien Grammatik, die anhand von Shapes arbeitet. Jene Symbole der Grammatik identifizieren hierbei eindeutige Shapes und können auf andere Shapes abgeleitet werden. Das Startsymbol ist hierbei eine einfache Shape, meist ein planares Rechteck, aus welcher sich die komplexe finale Shape aufbauen lässt. Hingegen sind die Terminalsymbole jene Shapes, deren Konstruktion abgeschlossen ist und die gewünschte (Teil-)Form darstellen. Kern der Shape Grammars und entsprechend der CGA Shape stellen die Produktionsregeln dar. Diese definieren den hierarchischen Ablauf der Ableitung von Shapes. Eine zu Shape Grammars erweiterte Definition dieser Produktionsregeln kann wie folgt aussehen. $id : predecessor : cond \rightarrow successor : prob$ mit „id“ ein eindeutiger Bezeichner einer Regel, „predecessor“ dem Symbol einer Shape auf die diese Regel angewandt werden soll, „cond“ eine Bedingung unter der diese Regel angewendet werden kann, „sucessor“ dem Nachfolger-Symbol und „prob“ eine Wahrscheinlichkeit zu der diese Regel ausgeführt werden soll. Produktionsregeln wenden Operationen auf den predecessor an, die die Shapes modifizieren. Man unterscheidet hier zwischen Erzeugenden Regeln und Scope Regeln. Scope Regeln sind jene Produktionsregeln, die eine Shape lediglich anhand ihres Scopes modifizieren, wie etwa Skalieren, Transformieren oder Rotieren. Hingegen sind erzeugende Regeln jene Regeln die ausgehend aus einer Shape, neue Shapes erzeugen und die zugrundeliegende geometrische Form modifizieren [7]. Unter Anwendung einer passenden Grammatik, lässt sich so schnell und effizient etwa ein Gebäude oder gar eine ganze Stadt

erzeugen. Folgend zwei resultierende Beispiele, die vollständig mittels der vorgestellten CGA Shape generiert wurden.

Abbildung 3.2: Beispiele von prozedural generierten Modellen mittels CGA Shape. Links Pompeii, Rechts generisches Gebäude [7].

3.2 Generierung von Cross-Over Fahrzeugen

Die Arbeit „Creating Cross-Over Vehicles Defining and Combining“ von Sets Orsborn, Randall C. Smith und Jonathan Cagan aus dem Jahr 2006 [9] beschäftigt sich mit der Generierung von Cross-Over Fahrzeugmodellen. Intention der Arbeit war es, den Fahrzeug-Design Prozess mittels prozeduraler Contentgenerierung zu beschleunigen. Als Cross-Over Fahrzeug bezeichnet man ein solches Fahrzeug, welches sich aus zwei verschiedenen Fahrzeugklassen zusammensetzt. So können verschiedene charakteristische Merkmale, beispielsweise eines Pickups und einer Limousine, kombiniert werden, um schließlich ein Cross-Over Fahrzeug zu generieren. Der Ansatz hinter der Arbeit beruht auf den Shape Grammars, welche in den Grundlagen beschrieben sind. Diese wurden im Vergleich zu Stiney's Arbeit [15] allerdings insofern erweitert, dass die Grammatik auf verschiedenen Koordinatensystemen arbeitet, um entsprechende Regeln zur Anwendung auszuwählen. Diese Erweiterung ermöglicht es, einen Ansatz zur Generierung von dreidimensionalen Objekten zu erzeugen. Konkret wird die Struktur der Produktionsregeln so erweitert, dass die Ein- und Ausgaben sich aus drei Perspektiven des Fahrzeugs zusammensetzen. (Front, Side, Rear). Abbildung 3.3 verdeutlicht diese Änderung. Regel 4S stellt einen Blickwinkel auf die Seite des Fahrzeuges dar, während 4F einen frontalen Blickwinkel darstellt.

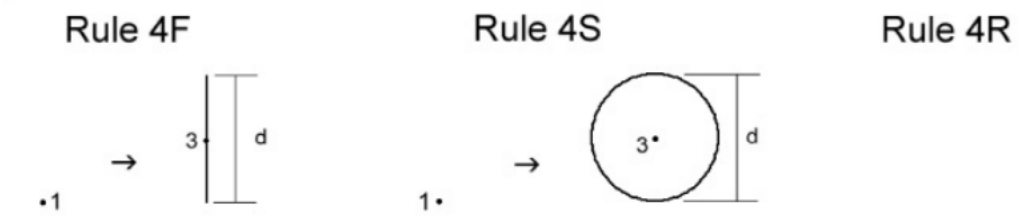


Abbildung 3.3: Regel 4 (Front, Side, Rear) [9]

Im Allgemeinen wird in der Arbeit so vorgegangen, dass sich zunächst passende Fahrzeuge herausgesucht werden, die näher zu betrachten sind. Jene Fahrzeuge untersuchen die Autoren folgend auf räumliche Faktoren und parametrische Gegebenheiten anhand von bestimmten, für das Fahrzeugdesign, essentiellen Charakteristiken. Konkret genannt sind hierbei Reifen, Radhäuser, Dach, Kotflügel, Stoßstangen, etc... [9]. Die identifizierten Merkmale werden folgend mithilfe von Bézierkurve (mit 4 Kontrollpunkten) nachgezeichnet, welche in das Vokabular der Shape Grammar aufgenommen werden. Mittels des aufgestellten Vokabulars wird eine Grundlinie iterativ prozedural erweitert, so dass ein Cross-Over-Vehikel erzeugt wird.

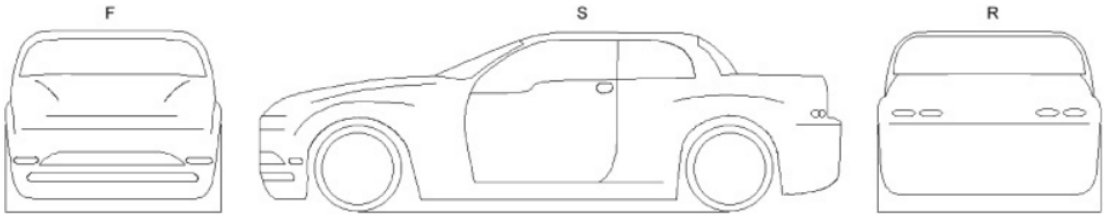


Abbildung 3.4: Prozedural erzeugtes Cross-Over-Vehikel [9]

4 Konzept

Das Ziel dieser Arbeit ist es, einen Prototypen zur prozeduralen Contentgenerierung von 3D-Fahrzeugmodellen zu entwickeln. In den folgenden Unterkapiteln, werden die zu erfüllenden Eigenschaften an die Software näher betrachtet und ein Vorgehen in der Implementierung beschrieben.

4.1 Ansatz

Verallgemeinernd ist zu erreichen, dass verschiedene Klassen von Fahrzeugen generiert werden können. Unter konkreter Anwendung von bestimmten Regeln der Grammatik, steht dabei in Aussicht grob ähnliche Fahrzeugmodelle zu einem Tesla Cybertruck für die Klasse der SUVs, VW Golf 2 für die Klasse der Kleinwagen und einem VW T4 für die Klasse der Transporter generieren zu können.
In Hinsicht auf den Aspekt der prozeduralen Contentgenerierung sollen darüber hinaus verschiedene charakteristische Merkmale dieser Fahrzeuge universell kompatibel zueinander sein. Der verwendete Ansatz zur Generierung der prozeduralen Fahrzeuge ist ein konstruierender, konkret ein auf Grammatiken basierender Ansatz. Die Grammatik soll gewisse Varianzen erlauben, um sogenannte Cross-Over Fahrzeugmodelle erzeugen zu können.

Abbildung 4.1: Fahrzeugklassen - Links Tesla Cybertruck (SUV), Mitte 3D Modell eines VW Golf 2 (Kleinwagen), Rechts VW T4 (Transporter)

Abbildung 4.1 - VW Golf 2 turbosquid.com, Tesla Cybertruck carwow.de, VW T4 auf Basis von automodels.fi; abgerufen am 11.04.2023

4.1.1 Implementierungsansatz

Die Implementierung beschäftigt sich primär mit der Entwicklung einer geeigneten Grammatik, die die Zielstellung dieser Arbeit ermöglicht. Um diese entsprechende Grammatik zu entwickeln, sind verschiedene Gegebenheiten des Systems nötig. Hierunter zählen interne Verfahren, die grundsätzlich das Evaluieren der passenden Grammatik ermöglichen, aber auch verschiedene Operatoren innerhalb der Grammatik, die benötigt werden, um diese entsprechende Grammatik zu entwickeln. Diese Verfahren und Operatoren sind in der Phase der Implementierung auszuarbeiten und zu entwickeln. Besteht dieses Konstrukt, so werden sämtlichen Bausteine in der zu entwickelnden Grammatik sinnvoll miteinander verknüpft, um schließlich die prozeduralen Fahrzeuge zu generieren.

4.1.2 Aufbau Fahrzeug

Um einen besseren Eindruck davon zu bekommen, was das Konzept konkret bieten soll, ist es vorweg wichtig zu beleuchten, welche Komponenten eines Fahrzeugs generiert werden sollen und wie diese miteinander zusammenhängen. Der Kern dieser Arbeit beschränkt sich auf die Karosserie von Fahrzeugen, so dass das Interieur oder nicht zusehende technische Elemente, wie in etwa der Motor, außer Acht gelassen werden.

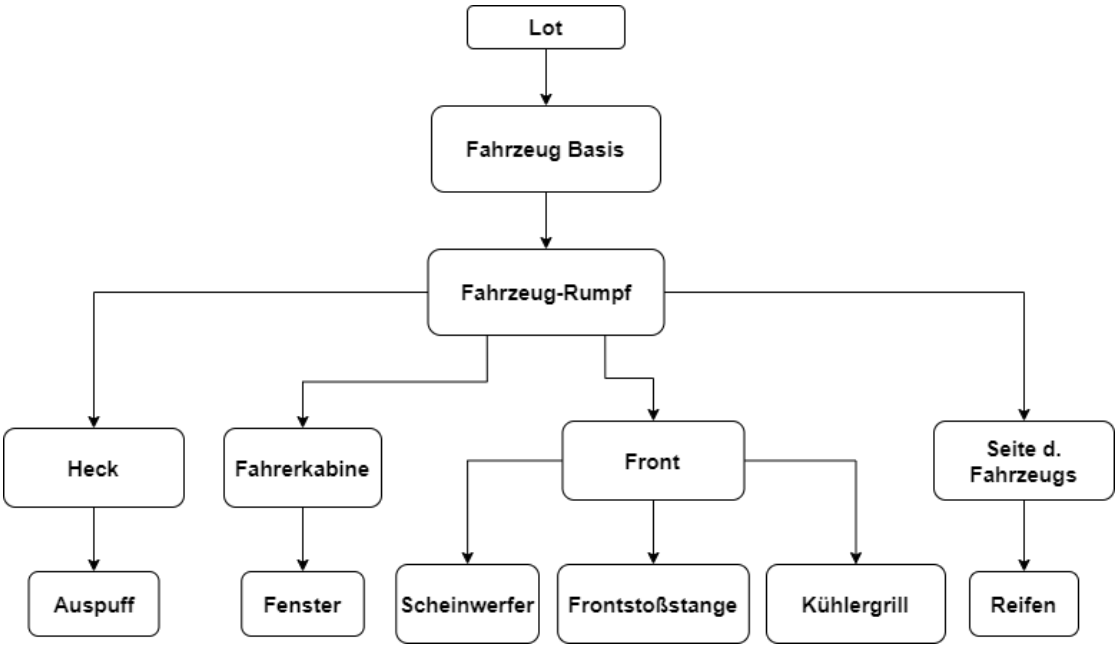


Abbildung 4.2: Hierarchie der aufeinander aufbauenden Fahrzeugkomponenten

Kern Komponenten der zu erstellenden Fahrzeuge sind die folgenden

- Fahrzeug Basis: Die Basis des Fahrzeugs stellt die grundlegende Dimension hinsichtlich Breite und Länge des Fahrzeugs dar. Konkret bildet diese die Grundfläche für sämtliche weitere Komponenten.
- Fahrzeug-Rumpf: Bezeichnet alles des Fahrzeugs bis zur Höhe des sichtbaren Bereichs der Fahrerkabine. Zusätzlich stellt der Fahrzeug-Rumpf die Front und das Heck des Fahrzeugs dar, an welche verschiedene Details anzubringen sind.
- Heck: Das Fahrzeug Heck bezeichnet die nach hinten gerichtete Seite des Fahrzeug-Rumpfs. Dieses kann mit verschiedenen Details erweitert werden.
- Fahrerkabine u. Dach: Sitzt auf dem Fahrzeug-Rumpf und stellt gleichzeitig die Basis für weitere Komponenten des Fahrzeugs dar.
- Front: Die Front bezeichnet die nach vorne gerichtete Seite des Fahrzeugs und wird mittels verschiedener Details erweitert.
- Fahrzeugseiten: Die Fahrzeugseiten bezeichnen jene Flächen des Fahrzeugrumpfs, welche zur Seite gerichtet sind. Diese bieten die Basis für die Reifen des Fahrzeugs.

Aufgrund dessen, dass verschiedene existierende Fahrzeugmodelle als Inspiration dieser Arbeit dienen, wird sich an den Dimensionen dieser orientiert. Ausschlaggebend sind hier die Maße des Fahrzeug-Rumpfs und der Fahrerkabine in absoluten Einheiten. Darüber hinaus sind verschiedene charakteristische Merkmale der Fahrzeuge in relativen Werten anzugeben, so dass diese unabhängig des Fahrzeug-Rumpfs sinnvoll anzuwenden sind. In den folgenden Abbildungen 4.3, 4.4 und 4.5 [1] sind diese relativen Maße mittels eines Suffixes „r“ gekennzeichnet.

[1] Die folgenden Abbildungen dienen lediglich als Orientierung für die Grammatik, keine Gewähr auf präzise Korrektheit!

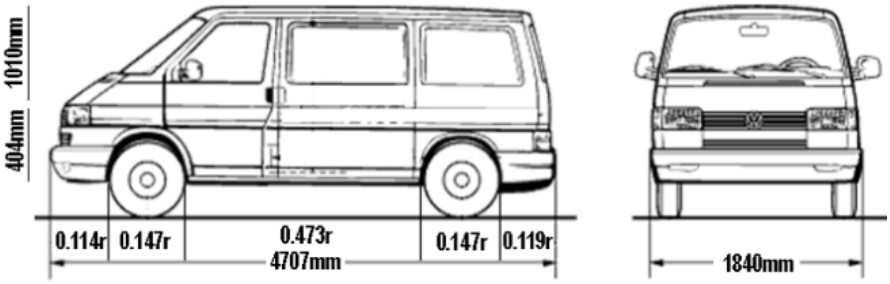


Abbildung 4.3: Verwendete Dimensionen eines VW T4s

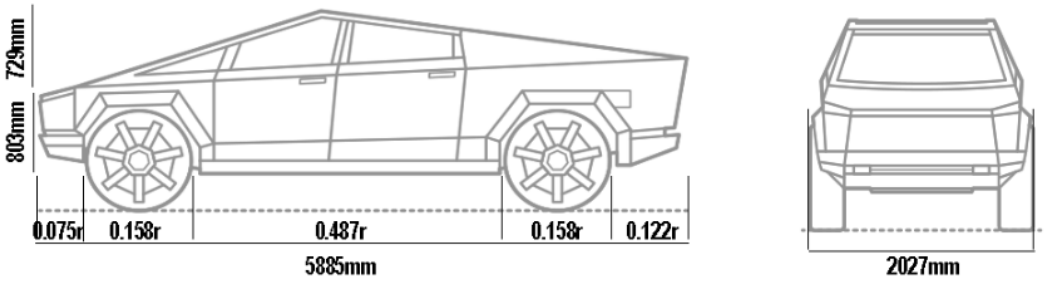


Abbildung 4.4: Verwendete Dimensionen eines Tesla Cybertrucks

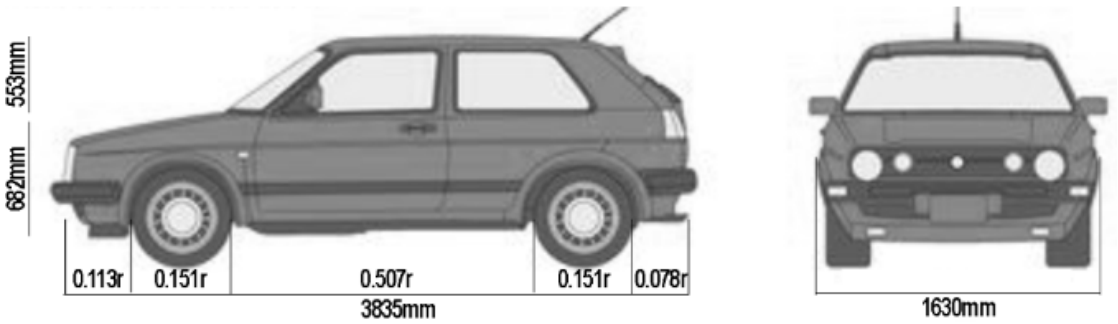


Abbildung 4.5: Verwendete Dimensionen eines VW Golf 2

Abbildung 4.3: Auf Basis von https://www.vwt4forum.co.uk/media/58221/full - abgerufen am 24.04.2023

Abbildung 4.4: Auf Basis von https://www.dimensions.com/element/tesla-cybertruck - abgerufen am 24.04.2023

Abbildung 4.5: Auf Basis von https://fi.pinterest.com/pin/586945763913296662/ - abgerufen am 24.04.2023

4.2 Funktionale Anforderungen

4.2.1 Variationen

Das Ziel dieser Arbeit ist ein prozeduraler Ansatz zur Generierung von Fahrzeugen. Dies impliziert einen gewissen Grad an Zufall, bzw. Varianz. Die Generierung von Cross-Over Fahrzeugen mittels ihrer Dimensionen und Details steht hierbei im Mittelpunkt. Als Teilziel gilt es, dass sich zwei nacheinander generierte Fahrzeuge optisch unterscheiden, voneinander differenzierbar sind und möglichst viele Varianten ermöglichen.

4.2.2 Realismus

Um ein Fahrzeug realistisch darzustellen, sind gewisse Komponenten obligatorisch. Im Kontext der Karosserie eines Fahrzeugs sind die folgenden Komponenten zwingend darzustellen.

- Reifen (Reifen, Felge)
- Front (Scheinwerfer, Stoßstange, Kühlergrill)
- Heck (Auspuff)
- Fahrerkabine (Fenster, ABC-Säule erkennbar)

Zu Erfüllen ist somit ein hoher Grad an Übereinstimmung der generierten Fahrzeugmodelle hinsichtlich den obligatorischen Komponenten eines Fahrzeugs.

4.3 Nicht-Funktionale Anforderungen

Die Generierung soll verschiedene nicht funktionale Anforderungen erfüllen. Vorwiegend soll die Implementation folgende Aspekte erfüllen.

- Effizient und Performant: Die resultierende Software sollte möglichst effizient und performant agieren und Resultate in angemessener Zeit ausgeben, so dass die Generierung potentiell unbemerkt zur Laufzeit geschehen kann.

- Skalierbarkeit: Generierte Fahrzeuge sollten stets unabhängig von einem Vorgänger-Fahrzeug generiert werden können. Darüber hinaus, sollen keine Merkmale des Vorgänger-Fahrzeugs Einfluss auf das nachfolgende Fahrzeug haben.
- Konfigurierbarkeit: Die Software soll parametrisierbar sein, so dass der Anwender der Software Möglichkeiten hat, konkrete Wunschvorstellungen mithilfe der Software effizient zu realisieren.

5 Implementierung

5.1 Einführung

Auf Basis des bereits existierenden Java „cgashape“ Projekts der PCG-Gruppe, geführt von P. Jenke, wird das Softwareprojekt insofern erweitert, dass das Entwickeln eines Prototyps zur prozeduralen Contentgenerierung von Fahrzeugen ermöglicht wird. Dieses Vorhaben lässt sich mittels eines konstruierenden PCG-Algorithmus realisieren. Verwendet wird in dieser Arbeit ein grammatikalischer Ansatz, welcher bestehende und zu implementierende Operatoren entsprechend miteinander verknüpft. Diese Operatoren modifizieren die zugrunde liegenden Polygonalen Netze, so dass schließlich ein 3D-Fahrzeug-Modell resultiert.

5.2 Architektur

Das Kapitel der Architektur behandelt für die Arbeit elementare Komponenten des bestehenden Softwareprojekts. Diese Komponenten werden anhand ihrer funktionalen Rolle oberflächlich beleuchtet.

5.2.1 Einordnung der Abhängigkeiten

Das bestehende Softwareprojekt bietet verschiedene Applikationen und Verwendungszwecke. Die folgende Einordnung der Grundarchitektur beschränkt sich auf das Unterprojekt „cgashape“, welches die Basis für diese Abschlussarbeit bildet.

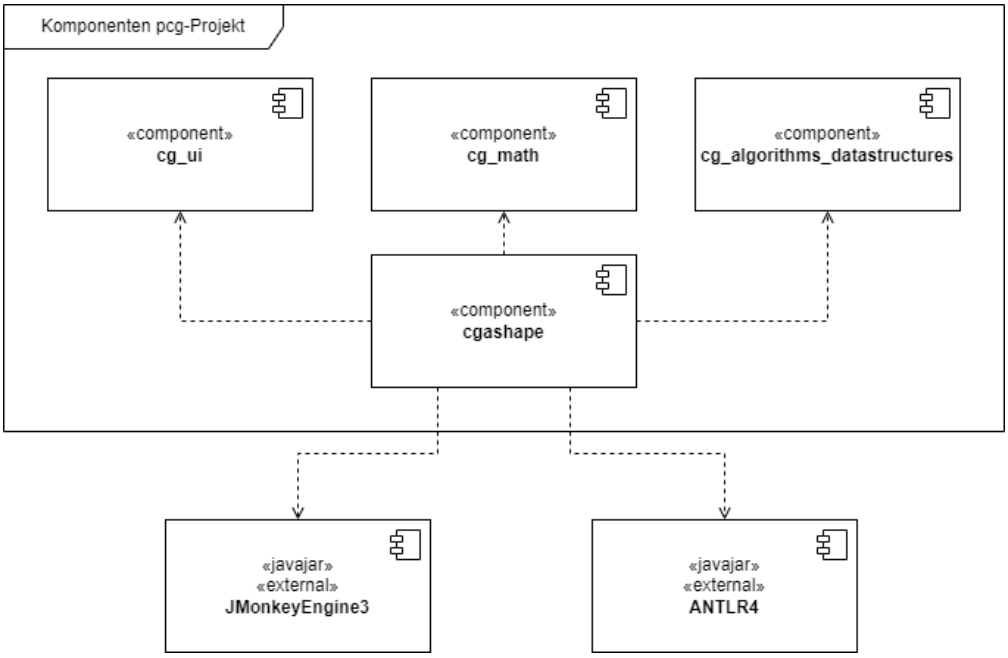


Abbildung 5.1: Komponentendiagramm der Grundarchitektur des zu betrachtenden Unterprojekts

Im Zentrum des zu betrachtenden Projekts steht die Applikation „cgashape“. Diese Applikation ist Hauptaugenmerk dieser Abschlussarbeit und dient als Grundlage jeglicher nötigen Anpassungen und Implementierungen. Darüber hinaus verwendet diese Applikation verschiedene Elemente der Komponenten „cg_ui“, „cg_math“ und „cg_algorithms_datastructures“, welche Teil der PCG-Gruppe und als gegeben zu betrachten sind.
Die „cg_ui“-Komponente bietet Elemente zum realisieren der Benutzeroberfläche für alle Applikationen innerhalb des Projekts. So verwendet auch die „cgashape“-Applikation Elemente der „cg_ui“-Komponente, die zur Visualisierung von Editoren oder der 3D-Szene dienen.
Die „cg_algorithms_datastructures“-Komponente hält insbesondere Datenstrukturen, die für die Evaluation der Grammatik-Regeln nötig sind.
Schließlich bietet die „cg_math“-Komponente verschiedene mathematische Bausteine, die für Berechnungen verschiedener Zusammenhänge der Evaluation Voraussetzungen sind.

Um die generierten 3D-Modelle darzustellen wird die 3D-Engine „JMonkeyEngine3“[1] eingesetzt. Dabei handelt es sich um eine etablierte Open-Source Java Spiele-Engine mit

[1] Spiele Engine - jmonkeyengine.org

großzügigem Funktionsumfang, welche Anwendung zum lösen der Zielsetzung findet. So werden in dieser Arbeit neben der grundlegend benötigten Fähigkeit zur Darstellung von 3D-Inhalten, weitere Funktionen genutzt, wie in etwa Beleuchtungsmodelle, um Schatten zu werfen.

Zum Einlesen der Grammatiken(-Dateien) findet das Framework „ANTLR4“[2] Anwendung. Dabei handelt es sich um einen Open-Source Parser-Generator, welcher eine textuell beschriebene formale Sprache einlesen und einen passenden Parser generieren kann. Dieser generierte Parser findet Anwendung für die Auswertung der Grammatiken des cgashape Projekts. Anzumerken ist hierbei, dass ANTLR4 abhängig der Definition der formalen Sprache eine Klasse generiert, die für das Parsen der entsprechenden Grammatik-Dateien verwendet wird. Einen genaueren Überblick vermittelt das Unterkapitel Grammatik.

5.2.2 Grammatik

Um eine Grammatik-Datei auf eine formale Sprache abzubilden und diese zu nutzen, finden verschiedene Bausteine innerhalb der Applikation Anwendung. Ein Klassendiagramm, welches sich auf die wesentlichen Bestandteile beschränkt ist in Abbildung 5.2 zu entnehmen.

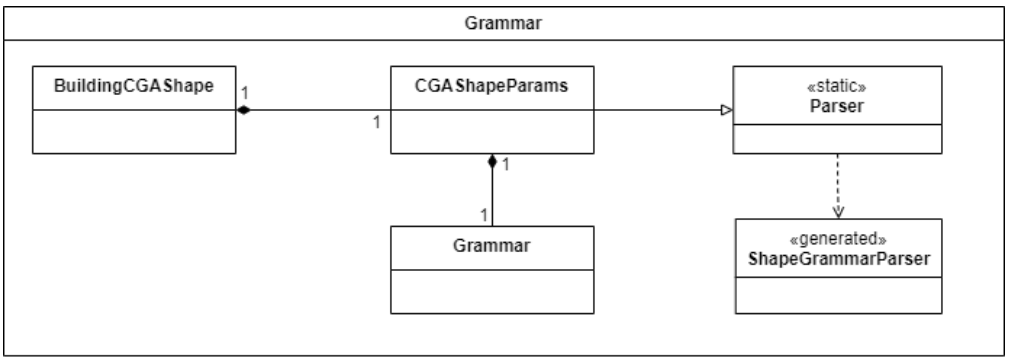


Abbildung 5.2: Klassendiagramm im Kontext des Auswertens einer Grammatik-Datei

Bei Erzeugung eines Objekts der Klasse BuildingCGAShape wird ein Objekt der Klasse CGAShapeParams erzeugt. Die Klasse CGAShapeParams ist dafür verantwortlich, sämtliche für die Applikation relevanten Daten zu speichern. So hält diese Klasse zum

[2] Parser-Generator, „ANother Tool for Language Recognition“ - ANTLR

einen, Standardparameter zur Auswahl einer Grammatik-Datei, als auch ein exemplarisches Axiom, welches als Lot dient. Darüber hinaus ist diese Klasse dafür verantwortlich, den Inhalt der Grammatik-Datei einzulesen und als Zeichenkette an den Parser zu übergeben. Beim Parser handelt es sich um eine statische Klasse, die mittels der von ANTLR4 generierten ShapeGrammarParser Klasse, Zeichenketten auswertet und schließlich als Objekt der Klasse Grammatik zurück gibt, welche zur Laufzeit im Objekt der CGAShapeParams-Klasse referenziert wird. Das CGAShapeParams-Objekt wird für sämtliche folgenden Operationen verwendet.

5.2.3 ShapeTree

Das Softwareprojekt arbeitet intern mit verschiedenen Shapes, welche abhängig von der Grammatik generiert werden. Auf Basis dieser Grammatik lässt sich ein ShapeTree generieren, welcher Metadaten zur Auswertung des finalen 3D-Modells hält.

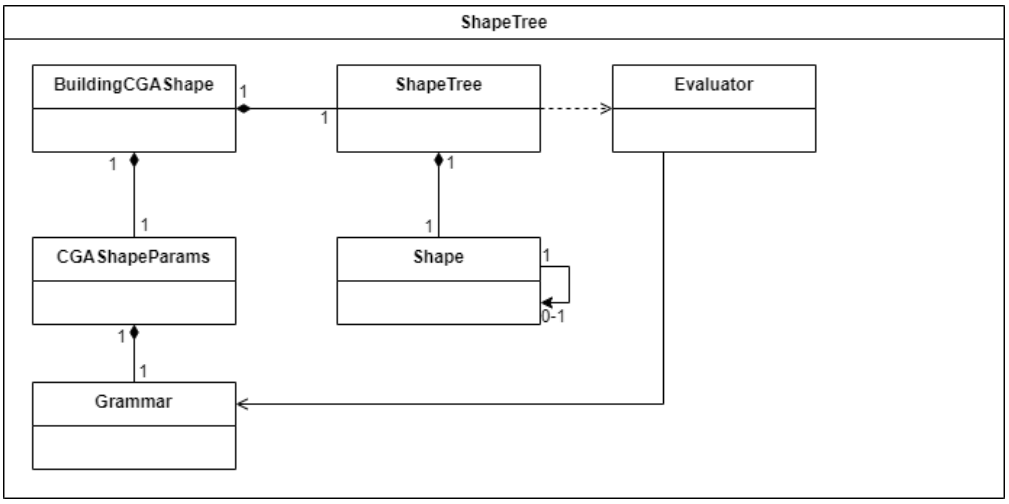


Abbildung 5.3: Klassendiagramm im Kontext des Generierens eines ShapeTrees

Nach Auswertung der Grammatik wird ein Objekt der Klasse ShapeTree erzeugt. Dieser initiiert schließlich ein Objekt der Klasse Evaluator, welches zunächst Variablen und Regeln anhand deren Wahrscheinlichkeiten auswertet. Ist dies erfolgt, wird der ShapeTree rekursiv abgeleitet in dem die successor-ids mittels der ausgewerteten Regeln auf Subshapes abgebildet werden. Diese Shapes werden mittels Nachfolger-Beziehungen verknüpft und die Wurzel-Shape wird schließlich dem ShapeTree zur Referenzierung ausgegeben.

5.2.4 MeshGenerator

Der MeshGenerator stellt die elementare Funktionalität der Applikation dar. Ausgehend aus dem ShapeTree wird hierbei ein TriangleMesh erzeugt, welches die Szene auf der Benutzeroberfläche bildet und somit, final das resultierende prozedural generierte Fahrzeug beschreibt.

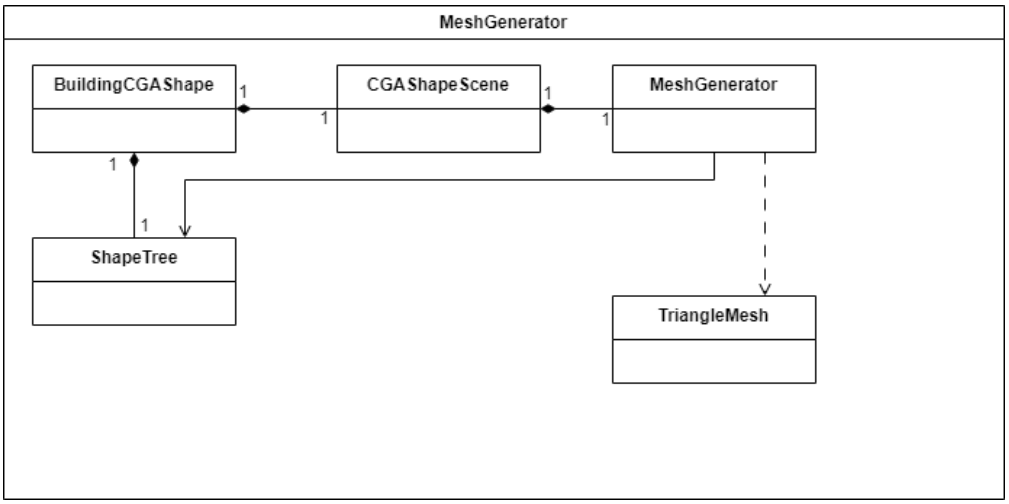


Abbildung 5.4: Klassendiagramm im Kontext des Generierens eines ShapeTrees

Ist der ShapeTree erzeugt, so wird ein Objekt der Klasse CGAShapeScene erzeugt, welches für die Generierung der Triangle-Meshes des 3D-Modells verantwortlich ist. Der Klasse wird dabei der entsprechende ShapeTree übergeben, welches mittels einer neuen Instanz eines MeshGenerators die TriangleMeshes generiert. Hierfür betrachtet der MeshGenerator jede Shape in den Blattknoten und erzeugt passend zu der jeweiligen Shape ein eigenständiges TriangleMesh die final zusammengetragen und als geschlossenes 3D-Objekt ausgegeben werden.

5.3 Modifikationen der Grammatik-Syntax

Im Rahmen der Realisierung der prozeduralen Generierung von Fahrzeugmodellen sind Modifikationen an der formalen Sprache, bzw. der Grammatik-Syntax für den ANTLR4 Parser-Generator vorgenommen worden. Diese werden im Folgenden näher beschrieben.

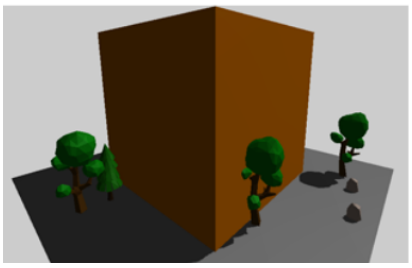
5.3.1 Setup Variables

Aufgrund dessen, dass verschiedene Quellen die echten Maße der zu betrachtenden Fahrzeuge in Millimeter angeben und eine Einheit innerhalb der Szene 10 Meter darstellt, wurde der Parser-Generator um eine weitere optionale Sektion in der Grammatik Datei erweitert. Bei dieser Sektion handelt es sich um sogenannte Setup Variablen, welche der Intention nachgehen, den Evaluator aktiv zu beeinflussen.
Im Kontext der Fahrzeugmaße wurde so die Möglichkeit geschaffen, sämtliche Variablen innerhalb der entsprechenden Sektion in der Grammatik zu skalieren, in dem die Setup-Variable „SCALE_VARIABLES“ definiert wird.
Um dies zu realisieren muss die Sprach-Datei, die von ANTLR4 genutzt wird, um den Parser zu generieren, entsprechend um die gewünschte Sektion erweitert werden. Da sich reguläre Variablen und Setup Variablen technisch nicht unterscheiden, wurden die Setup Variablen sowohl in der Sprach-Datei, als auch im vorangehenden Parser analog zu den regulären Variablen implementiert. Relevant ist hierbei, dass der Evaluator nun jedoch nicht nur die regulären Variablen evaluiert und zuordnet, sondern auch die Setup Variablen behandelt und je nach Vorkommen der „SCALE_VARIABLES“ Setup Variable, die regulären Variablen mit dem evaluiertem Wert multipliziert.

```
Setup:
SCALE_VARIABLES = 0.1;

Variables:
X = 1;
Y = 1;
Z = 1;

Rules:
Lot --> setLot(X,Z) ModifiedLot
ModifiedLot --> extrude(Y) ExtrudedLot
```



```
Variables:
X = 1;
Y = 1;
Z = 1;

Rules:
Lot --> setLot(X,Z) ModifiedLot
ModifiedLot --> extrude(Y) ExtrudedLot
```

Abbildung 5.5: Setup Variable „SCALE_VARIABLES“ am Beispiel zweier Grammatiken

So ist es nun möglich den Evaluator zur Laufzeit zu modifizieren und unter Angabe einer entsprechenden Skalierung jegliche Darstellungen einer Größeneinheit zu realisieren.

5.3.2 Statische Evaluation von Zufallswerten

Kommt es zu der Situation, dass mehrere Shapes mit dem selbem Symbol abgeleitet werden und die Regeln der Ableitungen alternativen besitzen, so besteht die Möglichkeit, dass diese Shapes unterschiedliche Regeln verwenden, was zu unterschiedlichen Resultaten führt. Mittels einer statischen Evaluation von Zufallswerten, wird ein neuer Operator in die Sprach-Syntax aufgenommen, der eine bereits ausgelöste Regel für darauf folgende Evaluierungen forciert. Dies hat den Effekt, dass unter Anwendung dieses Operators auf eine Wahrscheinlichkeit sämtliche Ableitungen desselben Symbols stets zum selben Resultat führt.

```
Rules:
Lot --> split("X", 1r, 1r){X,X}
X --> extrude(0.1):0.2
X --> extrude(0.2):0.2
X --> extrude(0.3):0.2
X --> extrude(0.4):0.2
X --> extrude(0.5):0.2
```

```
Rules:
Lot --> split("X", 1r, 1r){X,X}
X --> extrude(0.1):0.2!
X --> extrude(0.2):0.2!
X --> extrude(0.3):0.2!
X --> extrude(0.4):0.2!
X --> extrude(0.5):0.2!
```

Abbildung 5.6: Statische Evaluation am Beispiel zweier Grammatiken

Unter Berücksichtigung dieses Operators setzt sich die neue Syntax für die Definition einer Regel nun wie folgt zusammen. Zu beachten ist, das optionale Ausrufezeichen folgend auf die Wahrscheinlichkeit.

$$pre[: cond] - - > succ[: prob[!]]$$

Technisch realisiert ist dies zunächst mittels einer Anpassung an der Sprach-Datei für den Parser-Generator. Zudem wurde sowohl der übergeordnete Parser, als auch der Evaluator insofern modifiziert, dass mit einer Wrapper-Klasse „Probability‘, bestehend aus einem Float und einem Flag, hinsichtlich dem Aktivieren einer statischen Evaluation von Zufallswerten, statt lediglich einem Float, gearbeitet wird. Darüber hinaus werden Regeln, die statisch evaluiert werden sollen anhand ihres Hash-Codes persistiert, sofern

diese Regel ausgewählt wurde. Wurde ein Hash-Code einer in Frage kommenden Regel bereits zuvor persistiert, so wird die entsprechende Regel direkt ausgegeben, statt die Wahrscheinlichkeiten auszuwerten und eine Regel auszuwählen.

5.4 Operatoren und Shapes

Zusätzlich zu den Anpassungen an die Grammatik-Syntax, wurden sowohl neue Operatoren implementiert, als auch bestehende zum Teil modifiziert. Die folgenden Unterkapitel gehen auf diese Operatoren und Shapes ein.

5.4.1 duplicate-Operator

Mittels dem duplicate-Operator wird eine bestehende Shape dupliziert. Die Anzahl an angegebenen Nachfolger-Symbolen gibt dabei an, wie viele Duplikate erstellt werden sollen, so dass die ursprüngliche Shape beibehalten werden kann und beliebig viele Duplikate für nachfolgende Regeln verwendet werden können. Aus technischer Sicht werden die Duplikate realisiert, in dem das zugrunde liegende Shape geklont und das Duplikat als Kinds-Knoten der ursprünglichen Shape angehangen wird. In Abbildung 5.7 wird der duplicate-Operator angewandt, um das Lot zu duplizieren und eines der Duplikate entlang der Y-Achse zu transformieren. Nützlich ist dieser Operator insbesondere für Vorhaben, wo ausgehend von einer Shape eine neue Shape abgeleitet werden soll, ohne die ursprüngliche Shape zu modifizieren.

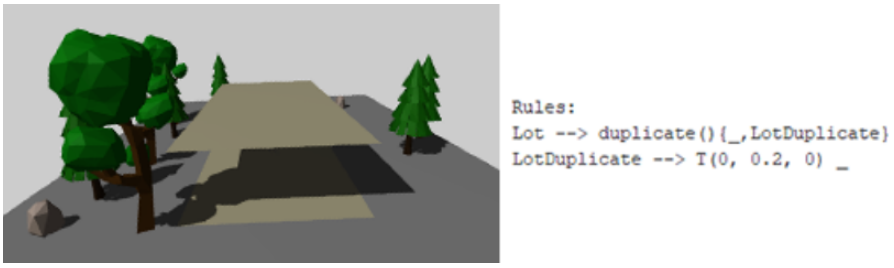


Abbildung 5.7: Duplicate-Operator am Beispiel einer Grammatik

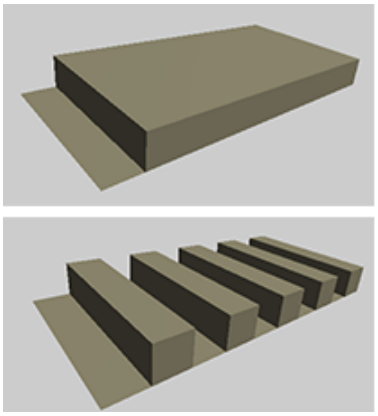
5.4.2 ellipse-Operator

Der hinzugefügte ellipse-Operator bildet aus einem Polygon mit vier Vertices eine Ellipse unter Berücksichtigung der Dimensionen des ursprünglichen Polygons. Dabei erhält dieser Operator einen Parameter zur Angabe der Präzision, welcher angibt, wie viele Vertices für die Generierung der Ellipse verwendet werden sollen. Je höher der Wert, umso weicher die Ellipse, auf Kosten von zusätzlichen Berechnungen und somit Zeit. Unter Anwendung einer Laufvariable, welche die Anzahl der Eckpunkte darstellt, lässt sich der Winkel bestimmen, auf dem der zu betrachtende Punkt liegt. Dieser Winkel und die Dimension der Ellipse wird verwendet, um die konkrete Position des Punktes zu bestimmen. Kumuliert man diese berechneten Punkte, so lässt sich schließlich ein geschlossenes Polygon bilden, welches die Ellipse darstellt.

Abbildung 5.8: ellipse-Operator am Beispiel zweier Grammatiken. Links mit 10 Vertices Präzision, Rechts mit 100 Vertices

5.4.3 repeated_split-Operator

Der bereits bestehende repeated_split-Operator, welcher ein Polygon entlang einer gegebenen Achse möglichst oft splittet, wurde um einen weiteren optionalen Parameter erweitert. Dieser Parameter gibt an, wie die Zuordnung der Nachfolger-Symbole erfolgt. Die ursprüngliche Implementierung hat jedes Segment eindeutig einem Symbol zugeordnet. Sind mehr Segmente durch die Splits entstanden, als es Nachfolger-Symbole gibt, so wurden alle zusätzlichen Segmente dem zuletzt definierten Nachfolger-Symbol zugeordnet. Dies hat gewisse Vorhaben für die Fahrzeug Generierung erschwert. Nunmehr ist es möglich, durch Angabe des „Repeating“-Parameters, die Zuordnung der Nachfolge-Symbole wiederholend stattfinden zu lassen. Dabei werden die Segmente auf die gegebenen Nachfolger-Symbole abwechselnd und wiederholend zugeordnet.



```
Rules:
Lot --> repeated_split("X", 0.1){X,Y}
Y --> extrude(0.1) _
```



```
Rules:
Lot --> repeated_split("X", 0.1, "REPEATING"){X,Y}
Y --> extrude(0.1) _
```

Abbildung 5.9: repeated_split-Operator am Beispiel zweier Grammatiken. Oben mittels ursprünglicher Zuordnung, unten mittels wiederholender Zuordnung

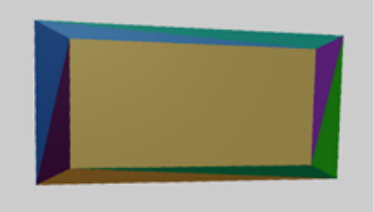
5.4.4 square-Operator

Der neu implementierte square-Operator agiert auf einem beliebigem Polygon mit 4 Vertices und bildet eine neue Shape, welche ein maximales Quadrat innerhalb dieses Rechtecks darstellt und positioniert dieses zentriert innerhalb des ursprünglichen Polygons. Realisiert wird dies indem, die kürzeste Seite des Polygons ermittelt wird und vier neue Vertices definiert werden, welche jeweils mittels der Hälfte dieser Länge entfernt vom Zentrum positioniert sind.

5.4.5 padding-Operator

Der neu implementierte padding-Operator teilt ein Polygon in zwei Flächen auf, wovon eine die innere Fläche und eine die äußere Fläche bildet. Davon stellt die innere Fläche quasi eine runter skalierte Kopie des ursprünglichen Polygons dar, womit die äußere Fläche die Differenz der inneren Fläche und des ursprünglichen Polygons bildet, so dass beide resultierenden Teil-Flächen vereint wiederum das ursprüngliche Polygon darstellen.

Dieses Vorgehen wurde realisiert, indem zunächst die bestehenden Vertices des ursprünglichen Polygons dupliziert und entsprechend einer Eingabe skaliert werden. Die duplizierten und skalierten Vertices bilden hierbei bereits die innere Fläche. Hinsichtlich der äußeren Fläche des Paddings ist nun eine Vereinigung der ursprünglichen Vertices und der neuen Vertices vorzunehmen. Jeder ursprüngliche Vertex wird iterativ betrachtet und



```
Rules:
Lot --> padding(0.2){Outer,Inner}
```

Abbildung 5.10: padding-Operator am Beispiel einer Grammatik

ausgehend aus den bekannten Vertices werden jeweils zwei Dreiecke/Polygone gebildet, welche sich durch die folgenden Vertices zusammensetzen:

- 1) $o_i, \quad o_{(i+1)\ mod\ |o|}, \quad i_i$
- 2) $o_i, \quad i_i, \quad i_{(i-1)\ mod\ |o|}$

Als Beispiel betrachte man Abbildung 5.11, in der die erste Iteration exemplarisch gekennzeichnet ist. Begonnen wird mit der Laufvariable $i = 1$. Es werden zwei Polygone/Dreiecke gebildet. Das erste Polygon, in grün eingezeichnet, setzt sich zusammen aus den Vertices o_1, o_2, i_1 und das zweite aus den Vertices o_1, i_1, i_5, welches orange eingezeichnet ist.

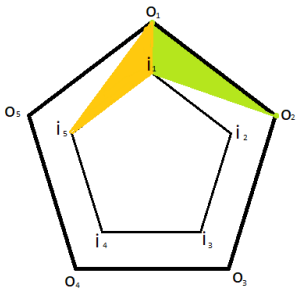


Abbildung 5.11: Algorithmus des padding-Operators am Beispiel eines Pentagons

Führt man dieses Vorgehen für jeden der 5 äußeren Vertices durch, so ergibt sich der äußere Ring bzw. die äußere Fläche des Pentagons und die beiden Flächen sind ermittelt. Dabei wurde sich hier bewusst für Dreiecke als Datenstruktur der äußeren Polygone entschieden, da diese unabhängig der Struktur des ursprünglichen Polygons stets zuverlässige Resultate liefern.

5.4.6 vehiclecabin-Operator und VehicleCabin-Shape

Der vehicleCabin-Operator ist der einzige Operator der implementiert wurde, der sich spezifisch auf das Vorhaben der Generierung von Fahrzeugmodellen beschränkt. Dieser Operator agiert auf einem rechteckigem Polygon und erzeugt ein VehicleCabin-Shape. Diese VehicleCabin-Shapes stellen hierbei die Fahrerkabinen der drei zu betrachtenden Fahrzeugklassen VW Golf 2, VW T4 und dem Tesla Cybertruck dar.

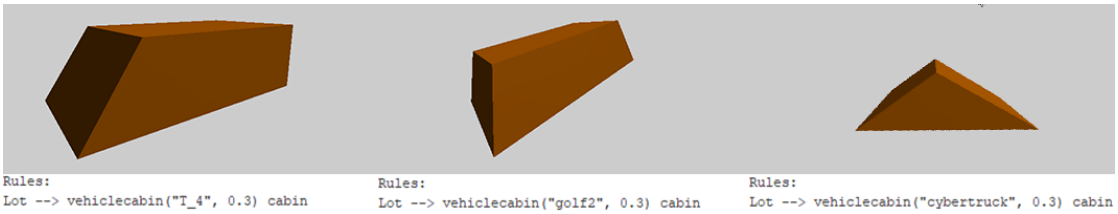


Abbildung 5.12: Darstellung des entstehenden Modells des VehicleCabin-Operator

Aufgerufen wird der Operator mittels zwei Parametern. Der erste Parameter gibt den Typ der Fahrerkabine an und der zweite Operator definiert wie hoch sich die Fahrerkabine erstreckt. Die zu erstellenden Meshes werden mittels eines für die VehicleCabins vorgesehenem MeshGenerators erstellt, indem die Charakteristika der Fahrerkabinen-Typen berücksichtigt werden. Bei der VW Golf 2 Fahrerkabine werden alle vier oberen Vertices sowohl in der X, als auch Z Achse in die Mitte eingelassen, so dass die typisch Trapezprisma Form entsteht. Hingegen werden die vorderen zwei obigen Vertices bei der VW T4 Fahrerkabine lediglich in der X-Achse eingelassen. Die Fahrerkabine des Tesla Cybertruck wird generiert indem zwei weitere Vertices der Grundfläche hinzugefügt werden und nahezu mittig entlang der X-Achse, allerdings näher zu den vorderen Vertices, als den hinteren Vertices, platziert werden. So entsteht die kürzere Fläche für die vorgesehene Windschutzscheibe und der flache Sturz der Heckscheibe.

5.4.7 vehicle_base-Operator

Der verhicle_base-Operator dient lediglich dazu innerhalb der Grammatik die Basis des Fahrzeugs dynamisch zu modifizieren bzw. zu setzen. Im Fall einer Generierung eines einzelnen Fahrzeugs kann dies z.B. das Lot sein, für welches dieser Operator primär implementiert wurde. Als Parameter werden zwei absolute Werte, X und Z, akzeptiert. Aus technischer Sicht wird ein neues Polygon erzeugt mit den Maßen der eingegebenen Parameter, welches im ShapeTree an die ursprüngliche Shape gehangen wird. Zusätzlich

hat dieser Operator noch einen weiteren Zweck. Um mehrere Fahrzeuge zu generieren, die voneinander unabhängig evaluiert werden, muss sichergestellt werden, dass die Liste der statischen Evaluationen leer ist. Der verhicle_base-Operator hat somit Zugriff auf den EvaluationContext und kann die Liste leeren.

5.5 Realisierung der Grammatik

Auf Basis der vorangehenden Kapitel in dieser Arbeit ist es nun möglich, sämtliche implementierte Verfahren, Operatoren und Shapes so miteinander zu verknüpfen, dass die Grammatik, die als Teil der Zielstellung dieser Arbeit fungiert, sich entwickeln lässt und somit Fahrzeug Modelle prozedural zu generieren. Den formalen Grammatiken liegt eine gewisse Modularität in der Natur. Diese Modularität wurde weitestgehend in den verschiedenen Komponenten der Fahrzeuge angestrebt, so dass diese Komponenten überwiegend voneinander unabhängig behandelt werden.

Wie bereits im Kapitel 5.3.1 Setup Variables vorweg genommen, ist vorgesehen, dass die Grammatik mit Millimetern als Einheit in deren Variablen statt mit 10m pro Einheit arbeitet. Um dies zu berücksichtigen setzt die Grammatik auf die „SCALE_VALUES“ Setup-Variable mit einer Skalierung von 0.0001 um die Werte von Millimeter auf 10 Meter Einheiten umzurechnen.

Ferner sind sämtliche grundlegende Charakteristika der drei verschiedenen Fahrzeugmodelle, als Variablen definiert. Konkret handelt es sich hier um acht Werte pro Fahrzeug. Diese Werte beschreiben die folgenden Maße:

- *_BODY_HEIGHT, *_BODY_WIDTH, *_BODY_LENGTH: Der Rumpf eines Fahrzeuges erstreckt sich vom unteren Beginn der Karosserie, bis hoch zur Fahrerkabine. Darüber hinaus befindet sich sowohl das Heck, als auch die Front samt dessen Details am Rumpf des Fahrzeuges.
- *_CABIN_HEIGHT: Dieser Wert beschreibt die Höhe der Fahrerkabine des jeweiligen Fahrzeugmodells.
- *_WHEEL_DIAMETER: Beschreibt den Durchmesser eines einzelnen Reifens, entsprechend der Charakteristik des Fahrzeugmodells.

- *_FWHEEL_TO_FRONT: Beschreibt den Abstand der Front des Fahrzeug-Rumpfs zum Beginn des vorderen Reifens entlang der X-Achse.
- *_BWHEEL_SPACE_INB: Beschreibt den kürzesten Abstand zwischen den beiden Reifen entlang der X-Achse.
- *_BWHEEL_TO_BACK: Beschreibt den Abstand des Hecks des Fahrzeug-Rumpfs zum Ende des hinteren Reifens entlang der X-Achse.

Ausgehend von der Grundfläche, dem Lot, wird mittels eines Aufrufs des vehicle_base-Operators die Dimension der Grundfläche angepasst und diese dann extrudiert. Als Resultat erhält man schließlich den Fahrzeug-Rumpf. Sowohl bei der Breite und Länge, als auch der Höhe verfügt die Grammatik über alternative Regeln, um Varianz bezüglich des Fahrzeug-Rumpfs zu ermöglichen. Somit besteht für jedes Fahrzeugmodell jeweils eine Option hinsichtlich Breite und Länge, als auch eine Option für die Höhe des Rumpfs. Darüber hinaus wird der Fahrzeug-Rumpf entlang der Y-Achse transformiert, um ihn vom Koordinatenursprung abzuheben, so dass Platz für die Reifen zwischen Boden und Fahrzeug-Rumpf ist.
Folgend wird der Fahrzeug-Rumpf in dessen Fassaden unterteilt, um Flächen zu schaffen, die für weiterführende Komponenten genutzt werden.
Die Fassaden, welche die Seiten der Karosserie darstellen sollen, werden entsprechend der Maße der Fahrzeugmodelle entlang der X-Achse partitioniert, um zunächst Segmente zu erzeugen, für welche die Reifen vorhergesehen sind.
Die entsprechenden Segmente für die die Reifen vorhergesehen sind, werden zunächst dupliziert, um so eine Lücke in der Karosserie zu vermeiden. Mittels dem square-Operators wird sichergestellt, dass es sich bei den Segmenten um ein Quadrat handelt, welches schließlich mittels dem ellipse-Operators zu einem Kreis modifiziert wird. Dieser Kreis wird extrudiert, so dass ein Zylinder entsteht. Entsprechender Zylinder wird folgend um den selben Wert, der zum abheben des Rumpfs verwendet wurde, in Richtung Y-Achse transformiert, so dass die Reifen vollständig auf dem Boden stehen. Final werden die Zylinder so modifiziert, dass ein Padding auf die nach außen gerichteten Fassaden gebildet wird, wovon die Innenseiten des Paddings negativ extrudiert werden. Dies resultiert in einem Reifen mit passender zu erahnender Felge.

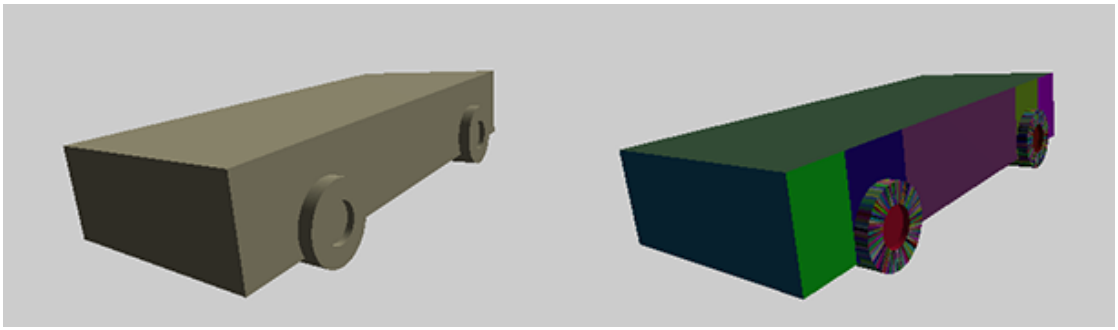


Abbildung 5.13: Fahrzeug-Rumpf und Reifen

Die obere Fassade des Fahrzeug-Rumpfs wird mittels des VehicleCabin-Operators und einer, je nach Fahrzeugmodell spezifischen Höhe, um eine Fahrerkabine ergänzt. Um die Fahrerkabine mit Fenstern auszustatten, werden auch von dieser VehicleCabinShape Fassaden extrahiert und Paddings auf verschiedene Fassaden, je nach Fahrzeugmodell, angewandt. Resultierende innere Flächen werden mit minimal negativem Wert extrudiert, so dass die Fenster zu erahnen sind. Bereits nun sind die zu modellierenden Fahrzeuge wahrzunehmen.

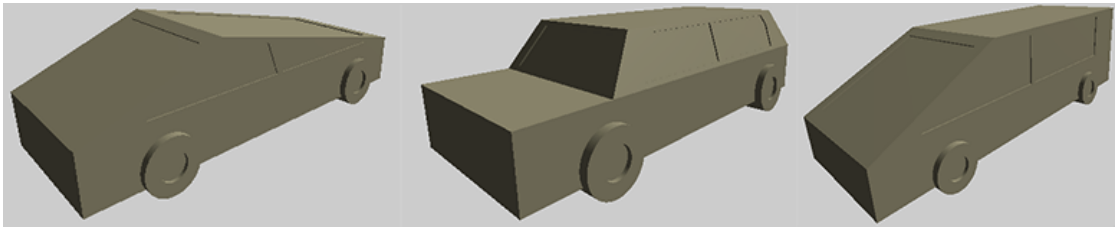


Abbildung 5.14: Fahrerkabine und Fenster (Tesla Cybertruck, VW Golf 2, VW T4)

Um die Details an die Front des Fahrzeugs anzubringen, wird die Front entlang der Vertikalen vorerst partitioniert und die resultierenden Segmente, den entsprechenden Details zugeordnet. Implementiert sind aktuell insbesondere die Details Kühlergrill (Horizontal und Vertikale Lüftungsschlitze), verschiedene Stile von Frontscheinwerfern und eine untere Stoßstange.
Die untere Stoßstange setzt sich zusammen aus einer Duplizierung des entsprechenden Segments und einer Extrusion des Duplikats, um eine gewisse Tiefe zu erlangen. Dieses Prisma wird schließlich in Richtung Boden transformiert, um sich von der Front abzuheben.
Der Kühlergrill wird realisiert indem das vorgesehene Segment mittels eines repeated_-split-Operators und eines gesetzten „REPEATING“-Flags entlang der Horizontalen oder

Vertikalen erneut partitioniert wird. Schließlich wird jedes zweite Segment aus dem repeated_split-Operator extrudiert, so dass eine Riffelung entsteht, welche den Kühlergrill darstellen soll.
Die Frontscheinwerfer sind in drei verschiedenen Varianten implementiert. Die Frontscheinwerfer des VW T4 sind realisiert mittels jeweils eines Rechtecks. Hierfür ist die für die Scheinwerfer vorhergesehene Spalte in ihre Zuständigkeiten unterteilt und die jeweiligen Spalten der Scheinwerfer extrudiert. Die Scheinwerfer des VW Golf 2 setzen sich durch jeweils zwei Zylinder pro Seite der Front zusammen. Auch hier ist das für die Scheinwerfer vorhergesehene Segment in ihre Zuständigkeiten unterteilt. Dabei wird dieses Segment erneut entlang der Horizontalen in sieben Segmente unterteilt. Vier Segmente sind für die eigentlichen Scheinwerfer vorhergesehen, zwei für einen Abstand zwischen Karosserie Ende und Scheinwerfer und ein Segment für einen Abstand zwischen den inneren Scheinwerfern. Zunächst ist mittels des square-Operators sicherzustellen, dass es sich bei den Scheinwerfersegmenten um Quadrate handelt, da die Scheinwerfer Kreise darstellen sollen und keine Ellipsen. Mittels des ellipse-Operators werden die Quadrate nun zu Kreisen transformiert, welche ähnlich zum Vorgehen der Reifen nun extrudiert und mittels eines weiteren Paddings und einer negativen Extrusion ausgestanzt werden. Bei den Scheinwerfern des Tesla Cybertrucks handelt es sich um eine schmale Einlassung in die Karosserie. Hierfür wird das für die Scheinwerfer vorhergesehene Segment erneut in der Vertikalen partitioniert, um ein entsprechend schmales Segment gewährleisten zu können. Dieses schmale Segment wird schließlich negativ extrudiert, so dass die Einlassung realisiert wird.

Abbildung 5.15: Exemplarische Generierungen der Front eines Tesla Cybertrucks, VW Golf 2, VW T4

Schließlich wird der Auspuff realisiert, in dem das Heck des Fahrzeug-Rumpfs partitioniert wird. Eine entstehende unten befindliche Zeile wird erneut entlang der Horizontalen partitioniert, um Flächen für den vorhergesehenen Auspuff zu realisieren. Diese Flächen werden im Fall eines rechteckigen Auspuffs zunächst extrudiert, mit einem Padding versehen und die innere Fläche negativ extrudiert, um ein Rohr zu erzeugen. Bei einem

runden Auspuff wird analog vorgegangen, nur dass zuvor die für den Auspuff vorhergesehenen Segmente mittels duplicate-Operator dupliziert werden, mittels square-Operator zu einem Quadrat und schließlich mittels ellipse-Operator zu einem Kreis transformiert werden.

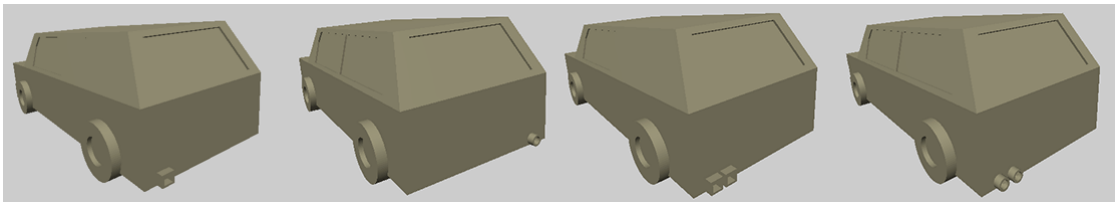


Abbildung 5.16: Exemplarische Generierungen der verschiedenen Variationen von einem Auspuff

Viele der genannten Komponenten des Fahrzeugmodells sind mit alternativen Regeln und Wahrscheinlichkeiten ausgestattet, so dass die Cross-Over Fahrzeuge ermöglicht werden. Insbesondere ist hierbei der Fokus auf Dimensionen und Details der Fahrzeuge gelegt. Die verschiedenen Varianzen und Möglichkeiten der Grammatik werden im Kapitel Evaluation konkretisiert.

6 Evaluation

In diesem Kapitel werden die verschiedenen Verfahren und Ergebnisse des Prototyps zur prozeduralen Generierung von Fahrzeugmodellen unter Anwendung einer Grammatik beleuchtet. Zunächst werden die Ergebnisse vorgestellt und beurteilt. Folgend wird auf die verschiedenen Anforderungen des Konzepts eingegangen und betrachtet, ob diese erfüllt sind. Abgeschlossen wird die Evaluation mittels verschiedener Metriken hinsichtlich z.B. der Performance.

6.1 Ergebnisse

Die prozedurale Contentgenerierung von Fahrzeugmodellen mittels eines Grammatikalischen Ansatzes erweist sich als äußerst gelungen. Durch den modularen Aufbau, den die Grammatik samt ihrer Operatoren ermöglicht, ist es vergleichsweise einfach, verschiedene Fahrzeuge zu generieren. Insbesondere bei der Anbringung verschiedener Details lässt sich dieser Ansatz wunderbar anwenden, da diese voneinander unabhängig zu betrachten sind und sich entsprechend ausgehend einer planaren Facade generieren lassen.
In den folgenden Abbildungen sind jeweils drei verschiedene Fahrzeugmodelle zu entnehmen, welche mittels der Grammatik generiert wurden. Gruppiert sind diese anhand der Fahrerkabinen, die die jeweiligen echten Fahrzeuge (VW Golf2, VW T4, Tesla Cybertruck) abbilden sollen. Wahrzunehmen sind verschiedene Dimensionen der Fahrzeuge, als auch verschiedene Details des Fahrzeuges. Eine genauere Auflistung der verschiedenen möglichen Details ist im Unterkapitel Anforderungen zu entnehmen.
Um einen Größenvergleich zu ermöglichen sind die Garten Assets der bestehenden Software Implementation aktiviert, deren Dimensionen als grobe Vergleichswerte dienen sollen.

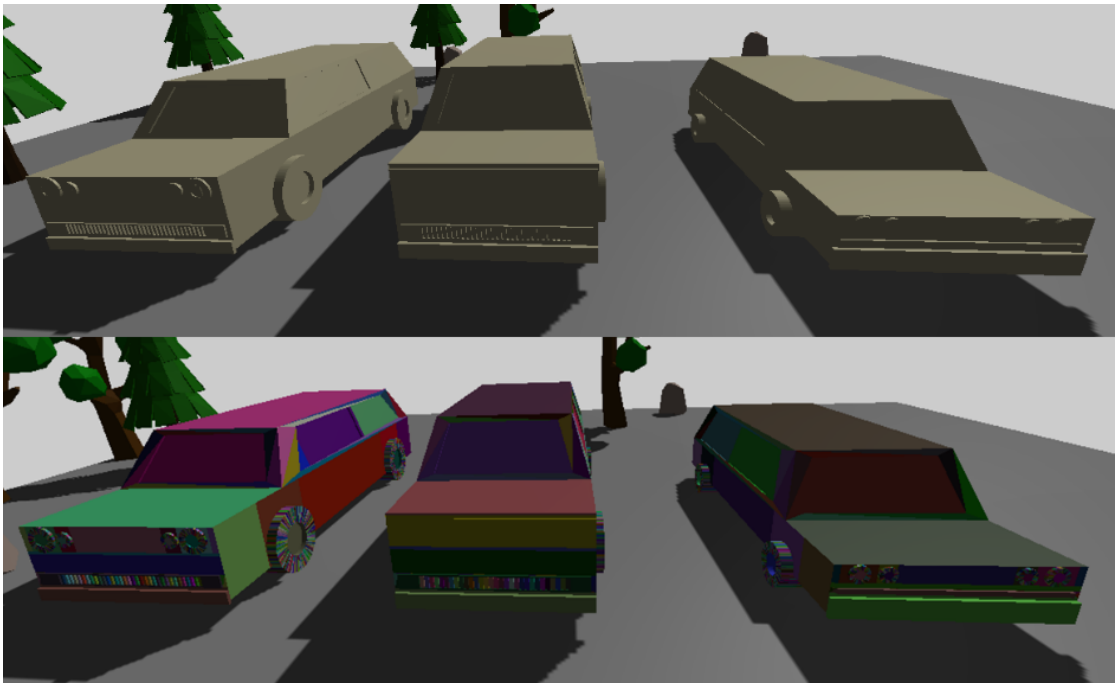


Abbildung 6.1: Drei exemplarische Generierungen der VW Golf 2 Inspiration

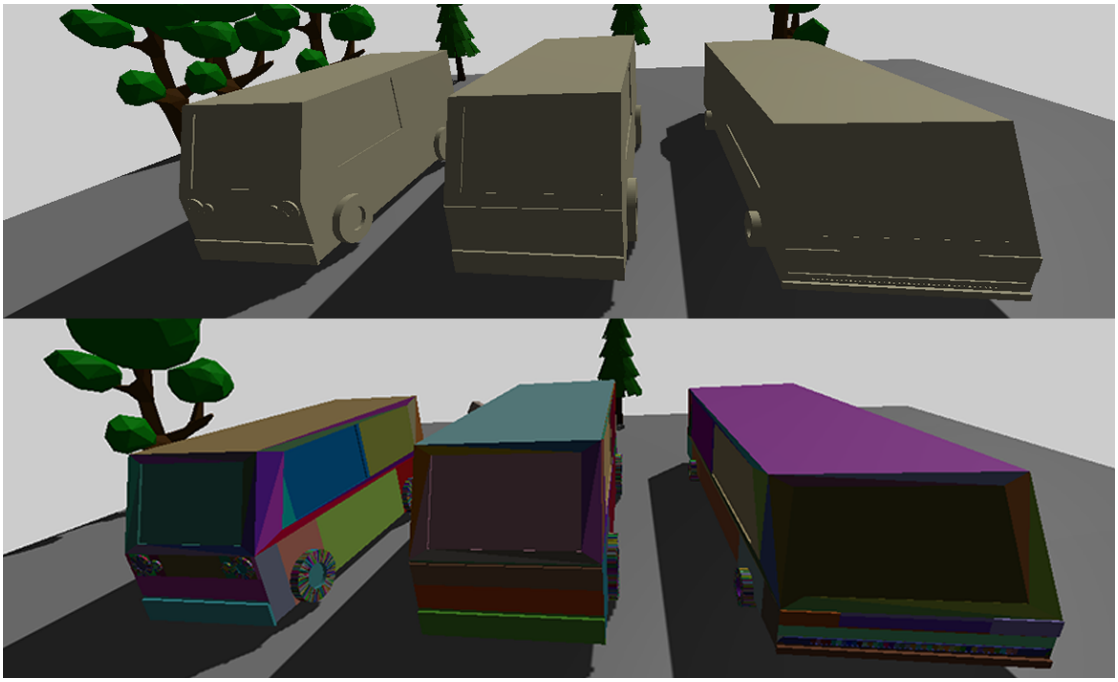


Abbildung 6.2: Drei exemplarische Generierungen der VW T4 Inspiration

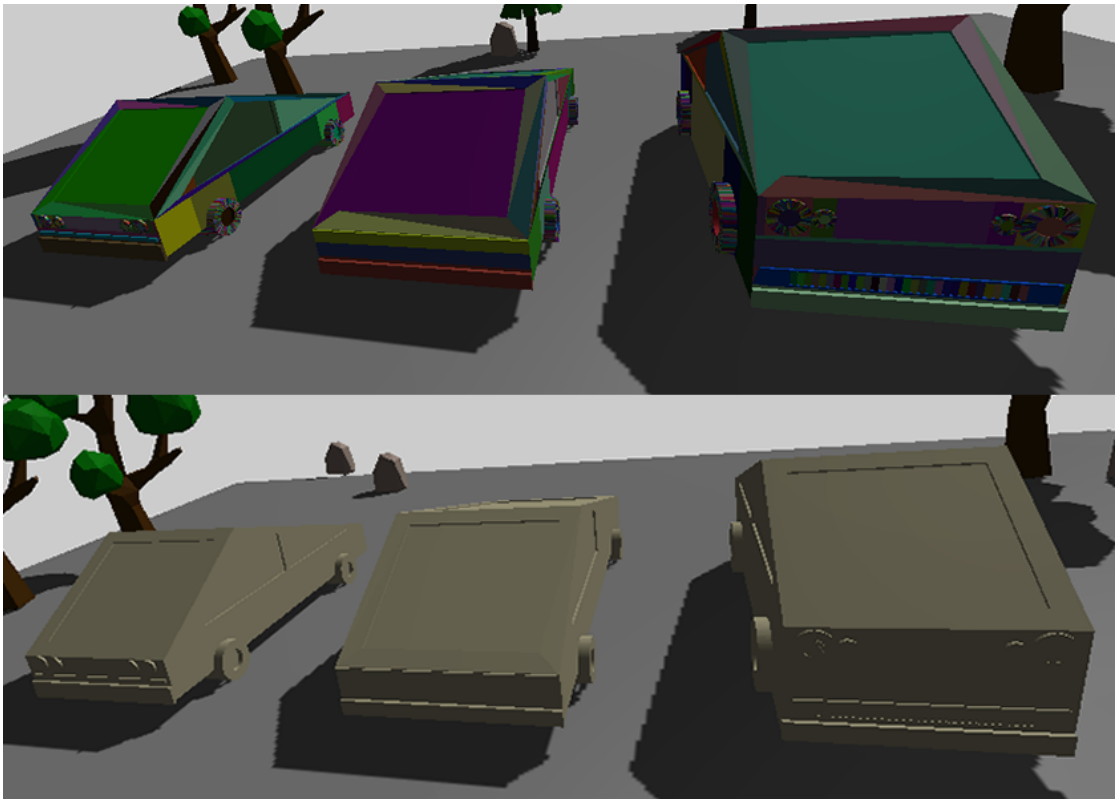


Abbildung 6.3: Drei exemplarische Generierungen der Tesla Cybertruck Inspiration

6.2 Anforderungen

Variationen: Die Implementierung der Fahrzeug Generierung unterstützt verschiedene Variationen zwischen den Fahrzeugmodellen. So wurde sich von den charakteristischen Merkmalen der realen Fahrzeuge inspiriert und diese für sämtliche Generierungen verfügbar gemacht. Bei der folgenden Abbildung 6.4 handelt es sich um eine Erweiterung, der aus dem Kapitel Konzept bekannten Abbildung 4.2. Ergänzt sind hier die verschiedenen Variationen, die im Prozess der Generierung möglich sind. Dies resultiert in insgesamt 52488 verschiedenen Fahrzeug-Kombinationen.

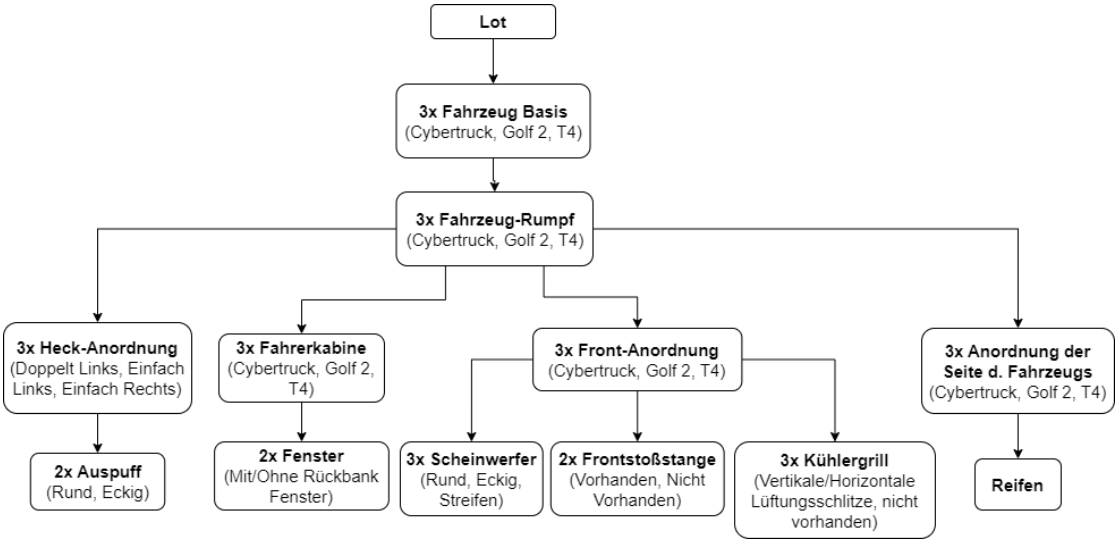


Abbildung 6.4: Hierarchischer Aufbau der Komponenten zuzüglich der Anzahl der implementierten Variationen

Realismus: Der Grad an Realismus ist zwar für diesen Prototyp ausreichend, allerdings ist ein deutlicher Unterschied zu den, als Inspiration verwendeten Fahrzeugmodellen wahrzunehmen. Eine grundlegende Ähnlichkeit ist jedoch zu erahnen. Mangeln tut es hierbei an Feinheiten, wie etwa Aussparungen von Komponenten (z.B. Rad-Kästen) oder auch Deformationen von Oberflächen, um ein realistischeres Erscheinungsbild generieren zu können. Dennoch verwendet die Grammatik quasi alle Essenziellen und charakteristischen Bestandteile eines Fahrzeugs, die dem Ziel dieses Prototyps dienen.

Skalierbarkeit: Mittels des „vehicle_base“-Operators werden sämtliche Regeln, die statisch zu evaluieren sind, zurückgesetzt. Dies ermöglicht eine stetig unabhängige Generierung. Selbst mehreren Generierungen von Fahrzeugmodellen, innerhalb einer Grammatik-Evaluation, sind so bedenkenlos möglich. Dieses Zurücksetzen, als auch der modulare Aufbau der Grammatik ermöglicht eine stetig zuverlässige Skalierbarkeit.

Konfigurierbarkeit: Sämtliche grundlegenden Attribute der Fahrzeug Generierung werden innerhalb der Grammatik mittels Variablen verwendet, die den Zweck der Konfigurierbarkeit dienen. Lässt man relative Werte unbeachtet, welche für universelle Dimensionen gedacht sind, so werden alle verwendeten Werte innerhalb der Grammatik mittels Variablen genutzt.

6.3 Metriken

Die gesamte Grammatik umfasst unter der Berücksichtigung von Laufvariablen und den Setup-Variablen insgesamt 33 Variablen über eine Menge von 76 Regeln. Diese Regeln ermöglichen insgesamt 52488 voneinander verschiedener Fahrzeuge.

Die folgende Tabelle beleuchtet die Laufzeit der neu implementierten und modifizierten Operatoren als einzelnes und die Generierung von zufälligen Fahrzeugmodellen als ganzes. Für die Operatoren wird jeweils der günstigste Ausgangspunkt verwendet, so operiert der ellipse-Operator direkt auf dem Lot. Gemessen wird bei den Operatoren lediglich die Ausführung der Operation, hingegen wird bei der Generierung eines Fahrzeuges ebenso die gesamte Ableitung des ShapeTrees gemessen. Dies erzeugt einen vergleichsweise geringen Overhead.

Subjekt	min [ms]	max [ms]	Durchschn. Zeit [ms]
duplicate-Operator (1 Child)	5.000	5.600	5.300
duplicate-Operator (10 Childs)	15.100	27.600	18.580
duplicate-Operator (100 Childs)	78.700	114.800	106.360
ellipse-Operator (50 Vertices)	11.400	13.400	12.140
ellipse-Operator (500 Vertices)	59.400	63.200	61.000
repeated_split-Operator (REPEATING)	21.900	24.600	23.060
square-Operator	11.000	128.000	11.680
padding-Operator (Rechteck)	11.900	15.000	13.020
padding-Operator (Ellipse [50 Vertices])	68.300	84.200	72.200
padding-Operator (Ellipse [500 Vertices])	528.300	554.600	538.820
vehicleCabin-Operator (Tesla Cybertruck)	3.900	5.000	4.340
vehicleCabin-Operator (VW Golf 2)	4.300	6.100	4.780
vehicleCabin-Operator (VW T4)	4.000	7.700	5.060
vehicleBase-Operator	1.600	2.700	1.820
Fahrzeug Generierung (+ Overhead)	1.683.000	3.610.300	2.590.380

Tabelle 6.1: Zeitmessungen verschiedener Operationen und Generierungen. Werte ergeben sich jeweils aus fünf Ausführungen

7 Fazit und Ausblick

Im Rahmen dieser Arbeit sollte ein Konzept ausgearbeitet werden, mittels dessen Fahrzeuge prozedural generiert werden sollen. Angewandt wurde ein grammatikalischer Ansatz der prozeduralen Contentgenerierung. Als Ziel galt es insbesondere, existierende Fahrzeugmodelle, konkret einen Tesla Cybertruck als Klasse eines SUVs, VW Golf 2 als Klasse eines Kleinwagens und einen VW T4 als Klasse eines Transporters zu imitieren. Um dieses Vorhaben zu realisieren, wurden bestehende Komponenten des bestehenden Softwareprojekts modifiziert und eine Grammatik ausgearbeitet, die dieses ermöglicht.

Zum einen wurde der bestehende Grammatik-Parser insofern angepasst, dass dieser die Möglichkeit bietet, Regeln mit bestehenden Alternativen statisch zu evaluieren, sofern diese bereits zuvor evaluiert wurden. Insbesondere dann, wenn man symmetrische Ableitungen bilden möchte, ist ein solches Vorgehen nötig. Darüber hinaus wurde der Grammatik-Parser weiterführend angepasst, um eine Skalierung sämtlicher Variablen vorzunehmen. Dieses Prinzip umschreibt die implementierten „Setup-Variables“. Verwendet werden kann dies für eine Anpassung der Größeneinheit des bestehenden Systems.
Um schließlich die Grammatik zu bilden, war es nötig verschiedene Operatoren und Shapes zu implementieren. Insbesondere wurde das bestehende Softwareprojekt um allgemein gültige Operatoren erweitert, welche in zukünftigen Vorhaben Anwendung finden können, Aber auch Operatoren die sich spezifisch auf das Generieren von Fahrzeugen beziehen, wurden implementiert.
So ist das Softwareprojekt nun in der Lage aus beliebigen Polygonen parametrisierbare Paddings zu erzeugen, aus Rechtecken Ellipsen und Kreise zu generieren, Rechtecke auf Quader abzubilden, Shapes zu duplizieren, Polygone wiederholend zu partitionieren, wobei die abzubildenden Symbole wiederholend Anwendung finden und fahrzeugspezifische Formen zu generieren, welche insbesondere die verschiedenen zu Imitierenden Fahrzeuge charakterisieren.

Eine Weiterführung der implementierten Elemente ist durchaus denkbar. So kann die

Generierung von Fahrzeugen weiter verfeinert werden, indem grundlegende Deformationen angewandt werden, um die Fahrzeugmodelle realistischer wirken zu lassen. Darüber hinaus, können weitere Details implementiert werden, um so einen höheren Grad an sowohl Varianz, als auch Realismus zu ermöglichen. Außerdem kann die Grammatik um weitere Fahrzeugmodelle erweitert werden oder gar den Begriff der Cross-Over Fahrzeuge weiter ausprägen, in dem die charakteristische Fahrerkabine weiter in ihre Bestandteile aufgeteilt wird und so beispielsweise ein Cross-Over zwischen der Tesla Cybertruck und VW Golf 2 Kabine ermöglicht wird.

Abschließend ist festzuhalten, dass die Anwendung eines grammatikalischen Ansatzes zur Generierung von 3D-Fahrzeugmodellen durchaus sinnvoll ist. Entsprechende 3D-Fahrzeugmodelle lassen sich so unter gegebenen Voraussetzungen, schnell und effizient implementieren und darstellen. Darüber hinaus ermöglicht die äußerst modulare Natur des grammatikalischen Ansatzes ein einfaches Hinzufügen und Austauschen von Elementen des resultierenden 3D-Modells. Insbesondere die relativ simpel umzusetzende Möglichkeit mittels Wahrscheinlichkeiten gewisse Varianzen zu ermöglichen, stellt ein besonderes Leistungsmerkmal dieses Ansatzes dar.

Literaturverzeichnis

[1] Botsch, Mario ; Pauly, Mark ; Kobbelt, Leif ; Alliez, Pierre ; Levy, Bruno: Geometric Modeling Based on Polygonal Meshes. In: Roussou, Maria (Hrsg.) ; Leigh, Jason (Hrsg.): *Eurographics 2008 - Tutorials*, The Eurographics Association, 2008

[2] Chomsky, N.: Three models for the description of language. In: *IRE Transactions on Information Theory* 2 (1956), Nr. 3, S. 113–124

[3] Ernst, Hartmut ; Schmidt, Jochen ; Beneken, Gerd: *Automatentheorie und formale Sprachen*. S. 371–414. In: *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis - Eine umfassende, praxisorientierte Einführung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2016. – URL https://doi.org/10.1007/978-3-658-14634-4_10. – ISBN 978-3-658-14634-4

[4] Hill, F.S.: *Computer Graphics: Using OpenGL*. Prentice Hall, 2001. – URL https://books.google.de/books?id=PhkoAQAAMAAJ. – ISBN 9780023548567

[5] Hughes, John F. ; Dam, Andries van ; McGuire, Morgan ; Sklar, David F. ; Foley, James D. ; Feiner, Steven ; Akeley, Kurt: *Computer Graphics: Principles and Practice*. 3. Upper Saddle River, NJ : Addison-Wesley, 2013. – ISBN 978-0-321-39952-6

[6] Lindenmayer, Aristid: Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. In: *Journal of theoretical biology* 18 3 (1968), S. 280–99

[7] Müller, Pascal ; Wonka, Peter ; Haegler, Simon ; Ulmer, Andreas ; Van Gool, Luc: Procedural Modeling of Buildings. In: *ACM Trans. Graph.* 25 (2006), 07, S. 614–623

[8] Nischwitz, Alfred ; Fischer, Max ; Haberäcker, Peter ; Socher, Gudrun: *Interaktive 3D-Computergrafik*. S. 25–29. In: *Computergrafik: Band I des Standardwerks Computergrafik und Bildverarbeitung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-25384-4_3. – ISBN 978-3-658-25384-4

[9] Orsborn, Seth ; Cagan, Jonathan ; Pawlicki, Richard ; Smith, Randall: Creating cross-over vehicles: Defining and combining vehicle classes using shape grammars. In: *AI EDAM* 20 (2006), 08, S. 217–246

[10] Prusinkiewicz, Przemyslaw: Graphical applications of L-systems. In: *Proceedings of graphics interface* Bd. 86, 1986, S. 247–253

[11] Rich, E.: *Automata, Computability and Complexity: Theory and Applications*. Pearson Prentice Hall, 2008. – URL https://books.google.de/books?id=lIuu53IcKWoC. – ISBN 9780132288064

[12] Shaker, Noor ; Togelius, Julian ; Nelson, M.: Procedural Content Generation in Games. In: *Computational Synthesis and Creative Systems*, 2016

[13] Sipser, Michael: *Introduction to the Theory of Computation*. Third. Boston, MA : Course Technology, 2013. – ISBN 113318779X

[14] Stiny, George: Introduction to Shape and Shape Grammars. In: *Environment and Planning B: Planning and Design* 7 (1980), S. 343 – 351

[15] Stiny, George ; Gips, James: 'Shape Grammars and the Generative Specification of Painting and Sculpture', 01 1971, S. 1460–1465

[16] Su, Yi-Hsiang ; Kumar, A. S.: Generalized Surface Interpolation for Triangle Meshes with Feature Retention. In: *Computer-aided Design and Applications* 2 (2005), S. 193–202

[17] Togelius, Julian ; Kastbjerg, Emil ; Schedl, David ; Yannakakis, Georgios: What is Procedural Content Generation? Mario on the borderline. (2011), 01

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