HAW HAMBURG

BACHELORTHESIS
The Kiet Dang

Prozedurale Generierung von Gebäuden im Cartoon-Look

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

The Kiet Dang

Prozedurale Generierung von Gebäuden im Cartoon-Look

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. Andreas Meisel

Eingereicht am: 25. Februar 2022

The Kiet Dang

Thema der Arbeit

Prozedurale Generierung von Gebäuden im Cartoon-Look

Stichworte

Prozedurale Generierung, Cartoon-Look, Deformation

Kurzzusammenfassung

Verwenden der Deformation, um einem Gebäude, das durch prozedurale Generierung erzeugt wurde, ein Cartoon-Aussehen zu verleihen …

The Kiet Dang

Title of Thesis

Procedural generation of buildings in cartoon look

Keywords

Procedural generation, Cartoon-Look, Deformation

Abstract

Using Deformation to achieve a cartoon look for a building, which created by procedural generation…

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Gebäuden mit Cartoon Stil sind leicht an ihrer Form zu erkennen. Sie sind oft stark von ihrer normalen Form verformt, um die Unterhaltung oder Fantasie zu bieten. Die Abbildungen 1.1 und 1.2 sind Beispiele für Gebäuden im Cartoon-Look, also sind ihre Wände im Vergleich zu normalen Gebäuden stark verformt. Im Bereich der Computergrafik kann die Form von einem normalen Gebäudemodelle verformt werden, indem die Deformation angewendet werden.Es gibt viele Arten von Verformungen wie Biegen, Verdrehen oder Dehnen, aber in dieser Arbeit werden nur Deformationen zu biegen konzentriert, die in der Lage sind, Oberflächen sowie Kanten von Polygonen in 2D-Ebenen zu biegen.

Darüber hinaus werden Gebäude in der Computergrafik oft durch Formgrammatik erstellt, oder genauer gesagt durch Computer Generated Architecture (CGA)-Formgrammatik. Dies ist ein System, das eine Menge von Transformationen sowie eine Grafik-Engine enthält, mit der Fähigkeit, Parameter vom Benutzer zu übernehmen und die neuen Formen aus einer Originalform zu generieren. Die Transformationen dieses Systems werden Regeln genannt, und Deformationen werden in diesem System als Regel entwickelt.

Abbildung 1.1: Cartoon Haus ist nach innen gebogen [24]

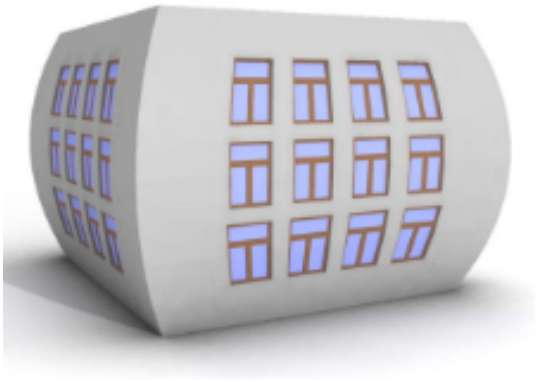


Abbildung 1.2: Cartoon Haus ist nach außen gebogen [24]

Da die Hauptaufgabe der Verformung darin besteht, Polygons zu biegen, ist die Verwendung einer parametrischen Kurve zum Orientieren der Verformung der geeignete Ansatz zur Implementierung der Deformationen.In dieser Arbeit wird eine Bezier-Kurve zweiter Ordnung ausgewählt, um die Verformung zu steuern. Dies ist eine parametrische Kurve, die durch ihre Kontrollpunkte geformt wird. Bezier-Kurve 2. Ordnung bedeutet, dass sie aus 3 Kontrollpunkten gebildet wird, und die Position des Kontrollpunkts in der Mitte ist der Parameter für die Verformung.

Diese Bachelorarbeit besteht aus 5 Teile. Nach der Einleitung ist die Grundlage, wo werden die relevante Kenntnis vorstellt und erklärt. Dann wird das Konzept und die Ziele von dieser Arbeit in der Anforderung und Konzept Teil demonstriert. Die viertel Teil detaillieren die Umsetzung von dem Konzept der dritte Abschnitt, während der Ergebnisse dieser Arbeit in

den Evaluation-Abschnitt analysieren und bewerten. Der letzte Abschnitt ist das Fazit dieser Arbeit.

2 Grundlagen

2.1 CGA Shape

Eine der typischsten Methoden zum Definieren von Prozedurale Modellierung Regeln ist die Verwendung der Formgrammatiken von [20]. Jede Form-Grammatik besteht aus 2 Teilen (Formregeln und Formgenerator). Die Regel wird verwendet, um zu beschreiben, wie wird eine alte Form neuen Details hinzugefügt, um neuen Formen zu generieren.

Unter Verwendung der Shape-Grammatik als Prämisse entwickelten Peter Wonka und Kollegen eine Art von Grammatik, die sich für die Modellierung von Gebäuden eignet und als Split-Grammatik [16] bezeichnet wird. Diese Grammatik konzentriert sich auf die Unterteilung von Formen in mehrere Unterformen, um das Hinzufügen von Details zu erleichtern. In der Abbildung ist ein Beispiel für Split-Grammatik, die Formen ohne Farbe stellen die nicht terminierte Form dar. Die kleinen Quadrate (F) werden aus dem ursprünglichen großen Rechteck (Start) unterteilt. Sie werden dann weiter unterteilt und mit Details und Farben versehen.

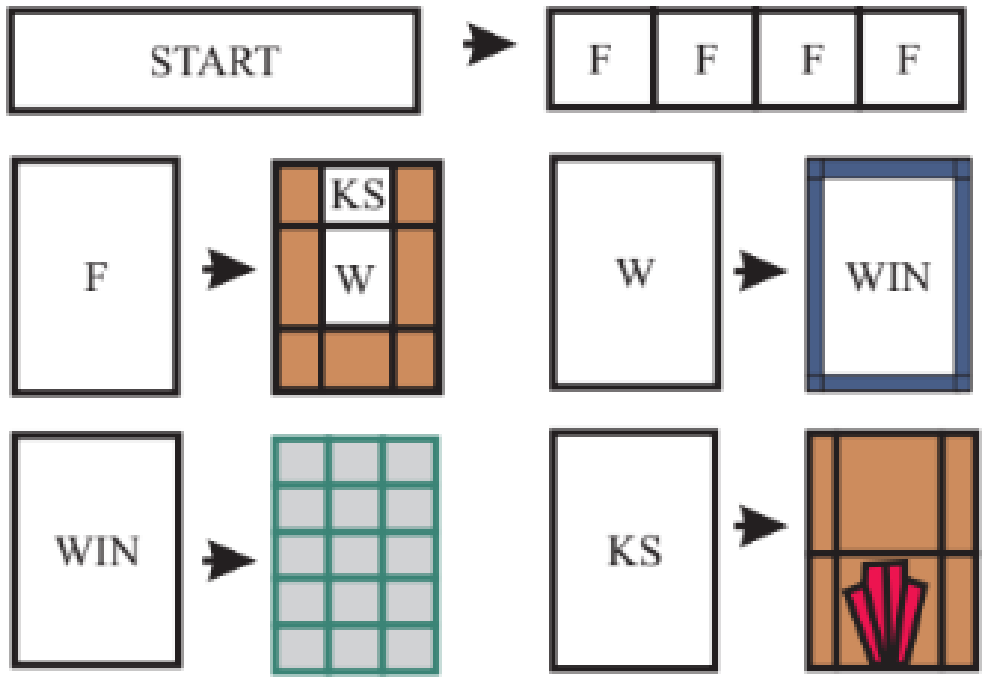


Abbildung 2.1: Beispiel für Split-Grammar

CGA Shape [22] ist eine Erweiterung von Split-Grammar, die 2006 von Wonka und Kollegen weiterentwickelt wurde.Da die Split-Regel der Split-Grammatik sich nur für einfache Modelle geeignet ist, wurde sie weiterentwickelt und zwei neue Split-Regeln erstellt, ”componenten-split” und ”repeat-split”. Während das ”componenten-split” die Fähigkeit hat, eine 3D-Form in kleine Teile in 2D unterteilt, hilft die ”repeat-split” dabei, sie in gleiche Teile zu teilen, was zum Erstellen sich wiederholendes Baustein wie Fenster geeignet ist.

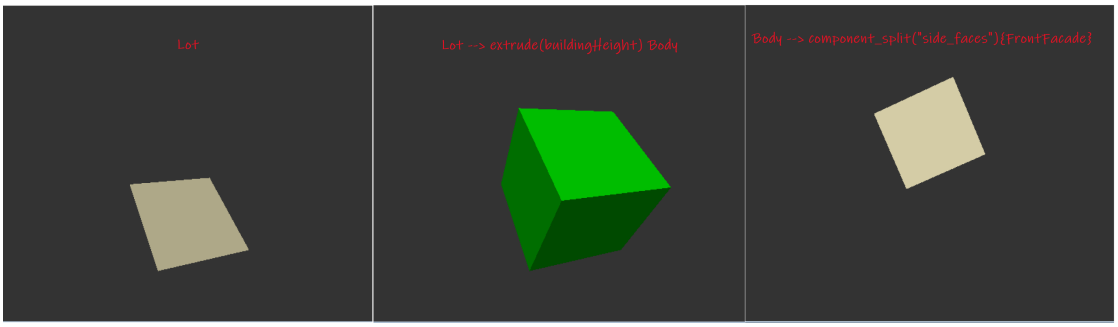


Abbildung 2.2: Beispiel von Form-Grammatik

Die obige Abbildung zeigt ein Beispiel von “component_split“. “Lot“-Form ist hier die Anfangsform von Grammatik. Durch die Extrudiert-Operation ist “Lot“ zu einer kubischen Form geworden, und diese Form wird “Body“ genannt. Um die“Body“-Form einfacher weiterzuentwickeln und Details hinzuzufügen, wird die “component_split“-Operation verwendet, um die Flächen dieser Form zu trennen. Die Deformation wird ähnlich so wie diesen Regeln entwickelt und wird Teil von Grammatik.

2.2 Bézierkurve

Die Beźier-Kurve [9] ist eine der grundlegendsten Kurven, die häufig in der Computergrafik und Bildverarbeitung verwendet wird. Beźier-Kurven können zum Erstellen von Straßen, Flüssen oder anderen Kurventypen verwendet werden. Die Bézierkurve wurde erstmals 1962 von dem französischen Ingenieur Pierre Beźier veröffentlicht und wird daher als Beźier-Kurve bezeichnet.

Eine Beźier-Kurve hat immer mindestens zwei Kontrollpunkte, den Anfang und das Ende der Kurve. Kurven mit höherer Ordnung werden mit mehr Kontrollpunkten definiert (eine Bézierkurve n-ten Grades wird durch n+1 Punkte beschreiben). Zum Beispiel hat eine lineare Bézierkurve (1.Ordnung) der geringsten Anzahl von Kontrollpunkten (2 Kontrollpunkte), dann hat eine Kurve zweiter Ordnung 3 Kontrollpunkten. Die allgemeine Formel für eine Kurve mit dem beliebigen Grad ist:

In den meisten Anwendungen werden normalerweise Beźier-Kurven zweiter oder dritter Ordnung verwendet. Die Verwendung von Beźier-Kurven höherer Ordnung ist oft sehr kompliziert und erhöht den Berechnungsprozess, daher werden quadratische oder kubische

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i = \sum_{i=0}^{n} B_{i,n}(t) P_i \tag{2.1}$$

2.1: Die allgemeine Formel für die Bézierkurve mit n-ten Grades

Kurven mehrmals verwendet, anstatt eine Beźier-Kurve höherer Ordnung zu verwenden. Die Deformation wird mithilfe der lineare Beźier-Kurven und Beźier-Kurven 2.Ordnung ausgeführt, um die Punkte der Formen nach dem Biegen entlang der Kurve 2.Ordnung zu finden.

2.2.1 Lineare Bézierkurve

Die Bézierkurve, die nur 2 Kontrollpunkte enthält, heißt lineare Bézierkurve. Sie ist eine Gerade, kann aber dennoch durch eine Kurvengleichung dargestellt werden:

$$B(t) = \sum_{i=0}^{1} t^i (1-t)^{1-i} P_i = (1-t)P0 + tP1, t \in [0,1] \tag{2.2}$$

2.2: Die Formel für die lineare Bézierkurven (n=1)

Das t in der Funktion für eine lineare Beźier-Kurve beschreibt, wie weit B(t) von P0 zu P1 entfernt ist. Wenn beispielsweise t = 0.75 ist, ist B(t) drei Viertel des Weges von Punkt P0 zu P1. Da t von 0 bis 1 variiert, beschreibt B(t) eine gerade Linie von P0 bis P1.

Abbildung 2.3: Lineare Bézierkurve-Kurve

2.2.2 Bézierkurve-Kurve 2.Ordnung

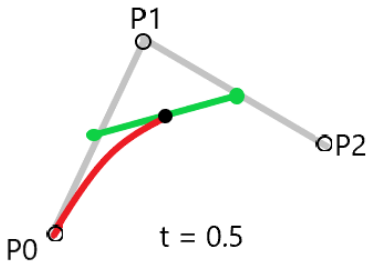


Abbildung 2.4: Bézierkurve-Kurve 2.Ordnung

eine quadratische Bézierkurve ist durch 3 Punkten P0, P1 und P2 bestimmt.Die Kurve beginnt bei P0, und geht in Richtung P1, aber die Kurve geht nicht durch P1, dies Punkt dient nur der Richtung, endet schließlich am Punkt P2. Die Formel zur Berechnung der quadratischen Bezierkurve lautet wie folgt:

$$B(t) = \sum_{i=0}^{2} t^i (1-t)^{2-i} P_i = (1-t)^2 P0 + 2t(1-t)P1 + t^2 P2, t \in [0,1] \tag{2.3}$$

2.3: Die Formel für die quadratische Bézierkurven (n=2)

2.3 Deformation

Deformation ist ein Prozess, um die Form eines Objekts in einer 2D- oder 3D-Grafik zu verändern. Durch dieses Verfahren kann ein Objekt gebogen, verdreht oder spitz werden. Eine der am beliebtesten Ansatz der Deformation ist die Freiform-Deformation (kurz: FFD, engl.: Free Form Deformation) von [1]. Bei der FFD wird das Objekt in einem Gitter von Kontrollpunkten gelegt. Die Kontrollpunkte können als Koeffizienten von dem Bernsteinpolynom betrachtet werden. Durch Ändern der Position der Kontrollpunkte werden die Punkte des Objekts mithilfe des Bernsteinpolynoms neu berechnet, dadurch das Objekt deformiert wird. Die Deformation dieser Arbeit wird von Freiform-Deformation inspiriert.

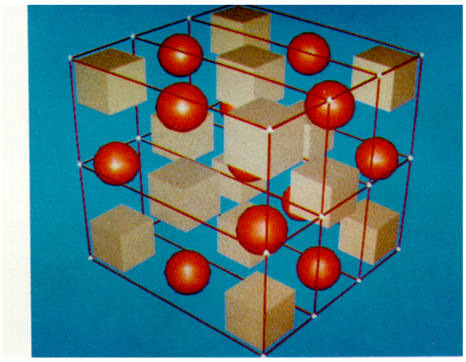


Abbildung 2.5: Unversetzte Kontrollpunkte
Quelle:[1]

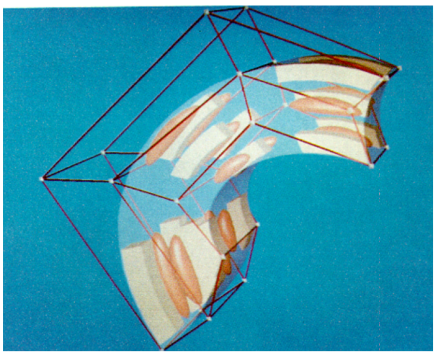


Abbildung 2.6: Deformierte Kontrollpunkte
Quelle:[1]

2.4 Tesselation

Mit der Verwendung des CGA-Shape-Frameworks werden 3D-Formen oft in mehrere Polygone in der 2D-Ebene aufgeteilt. Zur Vereinfachung der Modifizierung dieser Polygone wird Tesselation angewendet.

Tesselation ist eine Technik, die sich mit der Zerlegung von Polygonen beschäftigt. Es gibt viele Strategien, um diese Technik zu implementieren, aber das Ziel dieses Artikels ist, dass durch diese Technik zusätzliche Stützpunkte nicht nur an den Rändern des Polygons, sondern auch an den Rändern des Polygons erstellt werden. Diese Stützpunkte sind die Voraussetzung für die Verformung.

2.5 Delaunay Triangulierung

Die Delaunay-Triangulation [5] ist eine Menge von Dreiecken, die aus einer diskreten Menge von Scheitelpunkten besteht, sodass kein Scheitelpunkt innerhalb des Umkreises eines Dreiecks in der Menge liegt (Der Umkreis eines Dreiecks ist ein Kreis, der durch alle seine Eckpunkte geht). Mit dieser Eigenschaft von Delaunay-Dreiecken können die Dreiecksnetze aus beliebigen Punkten erzeugt werden. Die Triangulation ist nach Boris Delaunay für seine Arbeiten zu diesem Thema aus dem Jahr 1934 benannt.

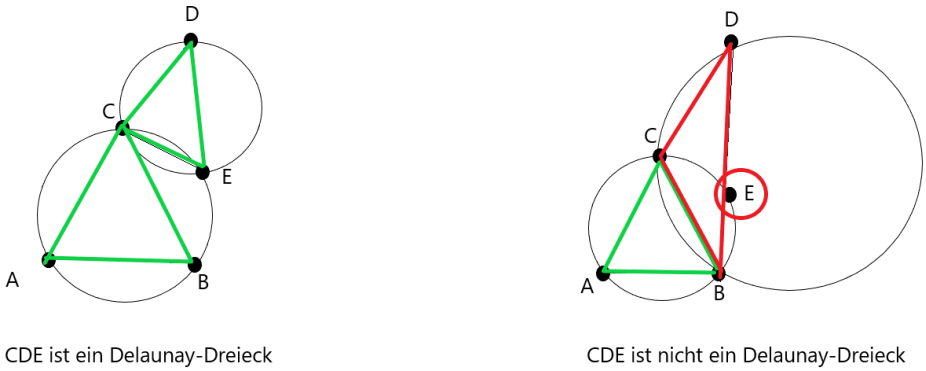


Abbildung 2.7: Beispiel für das Delaunay-Dreieck

.

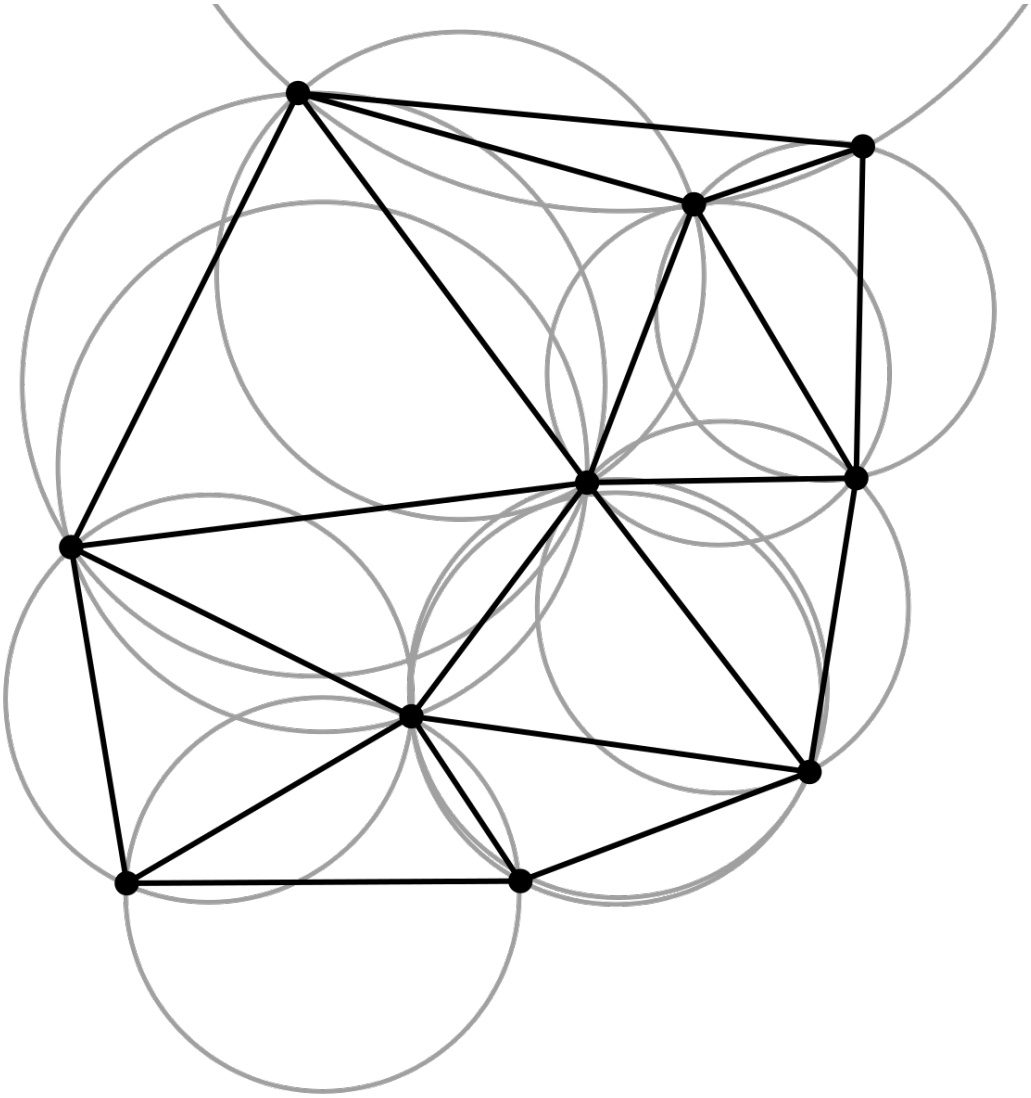


Abbildung 2.8: Beispiel für das Delaunay-Triangulation [1]

.

2.5.1 Bowyer–Watson-Algorithmus

Eine Möglichkeit, die Triangulation zu berechnen, ist der Bowyer-Watson-Algorithmus [4]. Der Algorithmus ist ein inkrementeller Algorithmus und nach den beiden Wissenschaft-

[1] Bild von en.wikipedia.org/wiki/Delaunay_triangulation#/media/File:Delaunay_circumcircles_vectorial.svg

lern Adrian Bowyer und David Watson benannt. Beide haben den Ansatz verwendet, Punkte nacheinander zu dem Satz von verarbeiteten Punkten hinzuzufügen. Bei jedem Hinzufügen eines neuen Punktes werden die Dreiecke im Netz überprüft, ob die Bedingung ein Delaunay-Dreieck noch erfüllt ist oder nicht.In dem Fall liegt der neue Punkt in einem Dreieck von der aktuelle Triangulation, wird das Dreieck aus der Triangulation entfernt.Indem die Konnektivität der Triangulation verwendet wird, um Dreiecke, die entfernt werden sollen, effizient zu lokalisieren, kann der Algorithmus O(N log N)-Operationen verwenden, um N Punkte zu triangulieren, obwohl es spezielle entartete Fälle gibt, in denen dies bis zu $O(N^2)$geht.

2.6 Globale und lokale Koordination

Eine Form, die zu einer Form-Grammatik gehört, kann Unterformen enthalten. Um also eine Form zu verformen, die untergeordnete Formen enthält, müssen auch die untergeordneten Formen verformt werden.Die Beziehung zwischen dem Koordinatensystem der Hauptform und den Unterformen hilft dabei, die Verformung korrekt durchzuführen.

Ein Koordinatensystem wird verwendet, um die Position von Objekten in einem bestimmten Raum zu bestimmen. Das Koordinatensystem, das alle Objekte im betrachteten Raum enthält, wird als globales Koordinatensystem bezeichnet.Innerhalb eines globalen Koordinatensystems können Objekte ihr eigenes Koordinatensystem haben, das als lokales Koordinatensystem bezeichnet wird. Dieses lokale Koordinatensystem hilft, die Position anderer Objekte relativ zu diesem Objekt zu bestimmen. Ein Punkt im globalen Koordinatensystem erfordert eine Transformationsmatrix, um die Koordinaten dieses Punktes im lokalen Koordinatensystem zu finden.

$$P_{global} \cdot M^T = P_{local}$$

3 Anforderungen und Konzept

Das nächste Kapitel beschäftigt sich mit den Herausforderungen dieser Arbeit. Das Konzept zur Lösung dieser Probleme werden auch in diesem Abschnitt Schritt für Schritt demonstriert, bis das Endziel erreicht ist.

Wie bereits in der Anleitung erwähnt, ist es das Ziel dieser Arbeit, eine Deformation zu entwickeln, die auf ein aus der Form-Grammatik generiertes Gebäudemodell angewendet werden kann. Im Abschnitt über die Deformation wurden mehrere Arten der Deformation aufgelistet, aber für diese Arbeit wird nur die Biegung, die einzige Deformation, betrachtet. Diese Deformation wird nicht auf das gesamte Gebäudemodell angewendet, sondern individuell auf jeden Gebäudeteil, die in eine flache Oberfläche liegen. Der Zweck besteht darin, die Formen des Gebäudemodells stark deformiert werden, um die dem Haus ein Cartoon-Look zu verleihen. Die Ansätze zur Entwicklung dieser Deformation sind das Deformieren der Kanten und Flächen der KinderForm des Gebäudemodells, sodass die Ergebnisse wie in der Abbildung 3.2 erhalten werden.

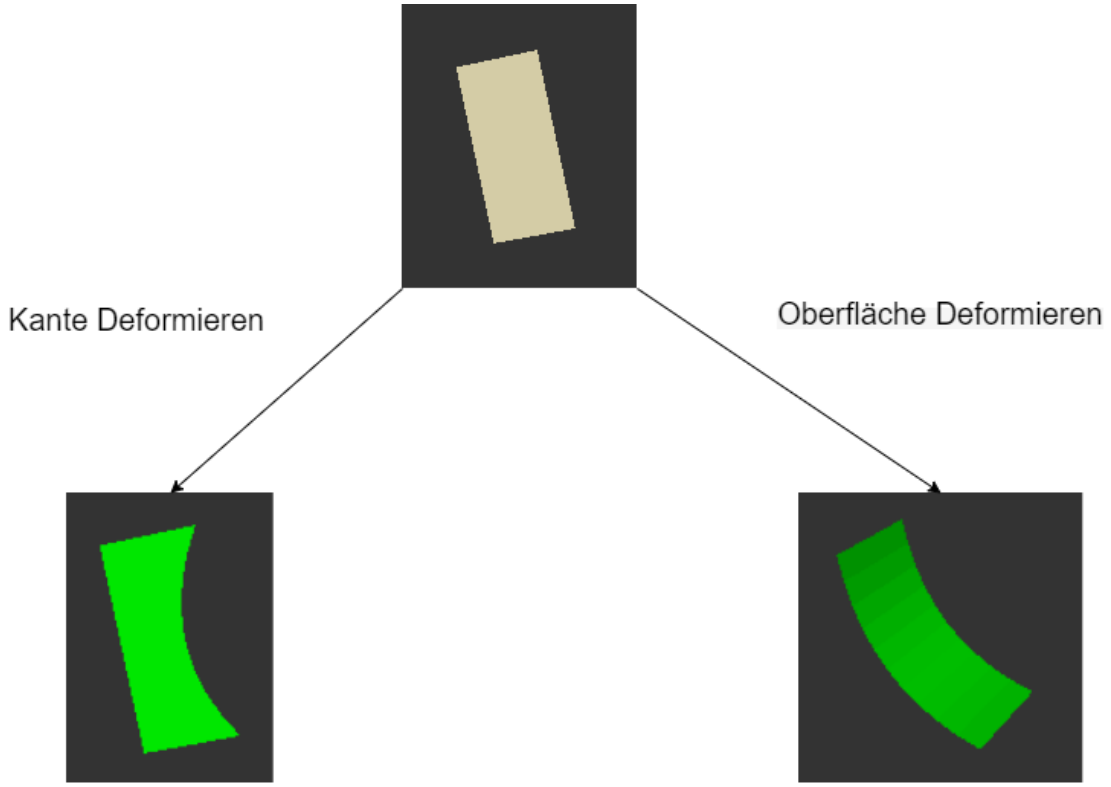


Abbildung 3.1: Die Deformationen

.

Zudem lassen sich diese Deformationen nicht nur für einzelne Polygone verwenden, sondern auch in Form-Grammatiken integrieren.Insgesamt werden drei Herausforderungen in diese Arbeit gestellt:

- Entwickeln eine Operation, um die Kanten eines Rechtecks zu verformen.
- Entwickeln eine Operation, um die Oberfläche des Polygons zu verformen.
- Integrieren die Deformation in Form-Grammatiken.

3.1 Deformieren der Oberfläche der Form

Das Deformation der Oberfläche von Polygonen besteht aus drei Schritten: Rastern, Deformieren und Triangulieren.

Zunächst wie beim FFD werden Kontrollpunkte innerhalb des Polygons erstellt. In der 2D-Ebene kann die Erstellung von Kontrollpunkte durch Rasterprozesse erfolgen. Dann wird eine geeignete Bézierkurve verwendet, um die Position der Kontrollpunkte zu manipulieren. Schließlich wird die Delaunay Triangulierung verwendet, um die Kontrollpunkte miteinander zu verbinden und die Form nach der Deformation zu erstellen.

3.1.1 Verfeinern

Der Grund, warum Kontrollpunkte wie in FFD erstellt werden müssen, liegt noch darin, dass, es nicht möglich ist, die Verformung des Polygons anzuzeigen, wenn nur die originalen Eckpunkte des Polygons verwendet werden, insbesondere wenn diese Deformation innerhalb des Polygons stattfindet. Es gibt viele Möglichkeiten, ein Polygon zu rastern, und eine davon ist der Scanline-Algorithmus.Dieser Algorithmus funktioniert, indem er die Scan-Linie mit Polygonkanten schneidet und das Polygon zwischen Paaren von Schnittpunkten füllt, dadurch kann er beliebige Polygons rastern.

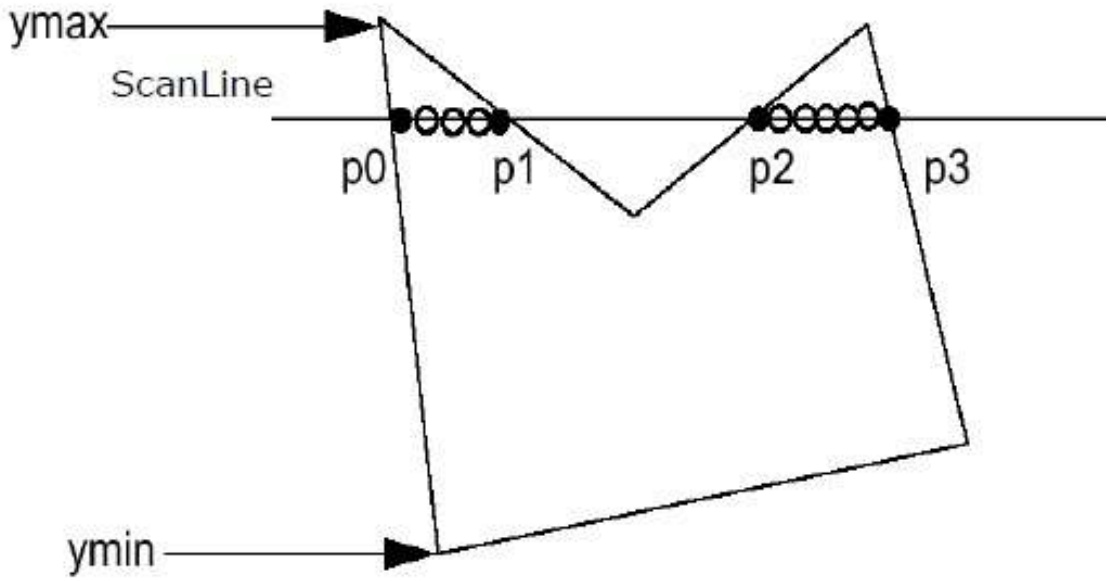


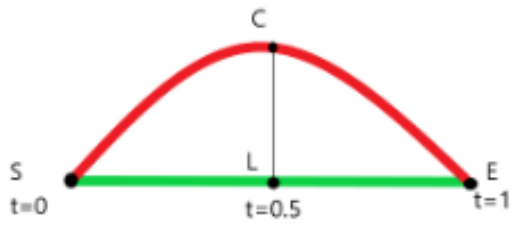
Abbildung 3.2: Rastern ein Polygons mit Scanline-Algorithmus[1]

[1] Bild von www.tutorialspoint.com/computer_graphics/images/scan_ line_algorithm.jpg

In diesem Rastern-Schritt sollen das Thema Qualität und Aufwand auch beachtet werden. Je mehr Kontrollpunkte erzeugt werden, desto glatter wird das Bild des Polygons, aber auch mehr Zeit und Ressourcen werden benötigt, um diesen Prozess durchzuführen. Daher ist es essenziell, ein Verhältnis von Abständen zwischen Punkten und Rastern zu wählen. Ferner zielt die Verwendung des Scanline-Algorithmus darauf ab, sich an alle Polygone in der 2D-Ebene rastern können. Es lohnt sich auf jeden Fall, sich auf das Rasterverfahren für jedes Polygon zu spezialisieren, um die Laufzeit zu verkürzen oder die Qualität zu erhöhen.

3.1.2 Deformieren

Nach der Erstellung der Kontrollpunkte werden die Position dieser Punkte in der deformierten Oberfächer des Polygons ermittelt.



B(t): Beźier-Kurvesgleichung
S, E: Start- und Endpunkt
L: Punkt auf der Linie
C: Punkt L nach der Deformation
C = B(SL\SE) : Deformationsformel

Deformation mittels der Beźier-Kurvesgleichung

In der Abbildung 3.1.2 wird die Deformation eines Punkts des Polygons mithilfe der Bézierkurve demonstriert. Wie in Abschnitt 2.2 bekannt, lässt sich eine Gerade auch durch die Gleichung der Bézierkurve darstellen. Die Position eines Punktes auf der Geraden steht in Relation zum Parameter t der Kurvengleichung, deshalb kann die Variable t aus der Position dieses Punktes abgeleitet werden, indem die Länge vom Startpunkt bis zu diesem Punkt durch die Summe der Linienlängen geteilt wird. die Linie und die Kurve dieselben Start- und Endpunkte haben, deswegen hat jeder Punkt auf der Linie also einen entsprechenden Punkt auf der Kurve mit dem gleichen Wert von t, dann kann der Wert eines Punktes im Polygon nach dem Deformieren durch Einsetzen des t-Wertes dieses Punktes in die Kurve ermittelt werden.

3.1.3 Triangulieren

In diesem Schritt werden Kontrollpunkten durch die Anwendung von dem Bowyer-Watson-Algorithmus miteinander verbinden, um ein Dreiecksnetz zu erzeugen. Zuerst wird ein Dreieck, das allen Kontrollpunkten beinhalten können, erstellen. Die Größe als auch die Position den Eckpunkten von dem Dreieck ist nicht wichtig, solange die vorherige Bedingung erfüllt wird. Dann werden Kontrollpunkte nacheinander im Dreieck hinzugefügt, und sich mit den Scheitelpunkten dieses Dreiecks verbinden, um Unterdreiecke zu erstellen. Bei jedem Hinzufügen eines Kontrollpunkts werden die neu erstellten Dreiecke daraufhin überprüft, ob das Dreieck die Bedingung der Delaunay-Triangulation erfüllt. Wenn das Dreieck die Bedingung nicht erfüllt, wird das Dreieck aus der Triangulation entfernt. Wenn alle Kontrollpunkte zum Superdreieck hinzugefügt wurden, enthalten die Dreiecke die Eckpunkte des Superdreiecks. werden eliminiert, auch wenn diese Dreiecke der Delaunay-Dreiecke sind.

3.1.4 Erzeugen Superdreieck

Um Bowyer-Watson-Algorithmus umsetzen zu können, muss zunächst ein Dreieck gefunden werden, das groß genug ist, um alle Punkte der Figur zu erhalten, dies Dreieck wird als Super-Dreieck genannt.

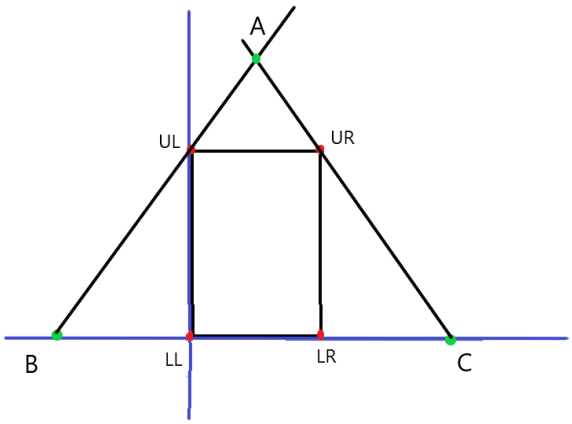


Abbildung 3.3: Super-Dreieck

.

In der Abbildung 3.3 ist das Dreieck, das den gesamten Umfang der Figur enthält. Die Punkte der Figur befinden sich alle in dem Umfang der Figur. Wenn das Dreieck den gesamten Umfang einer Figur enthält, enthält das Dreieck daher den gesamten Punkt dieser Figur. Durch die Größe der Figur können die Punkte Upper Right (UR), Upper Left (UL), Lower Right (LR) und Lower Left (LL) bestimmt werden. 2 beliebige Punkte (B, C) auf der Linie, die durch den Boden, der Umfang der Figur verläuft, werden als Eckpunkte, das Superdreieck ausgewählt, vorausgesetzt, diese 2 Punkte gehören nicht zu dem von UL und UR erstellten Liniensegment. Außerdem muss sich ein Punkt in der linken Seite und ein Punkt in der rechten Seite von der Figur liegen. Der letzte Eckpunkt des Dreiecks ist der Schnittpunkt von zwei Geraden. Der ersten Linie geht durch den linken Eckpunkt des Dreiecks und UL, während die zweite Linie durch den rechten Eckpunkt des Dreiecks und UR erzeugt wird.

3.2 Deformieren der Kante der Form

Die Deformation für die Kante befindet sich nur 2D Ebene, und wandelt eine normale Kante in der gekrümmten Kante. Außerdem wird diese Deformation in diese Arbeit nur für Rechteck spezifizieren.

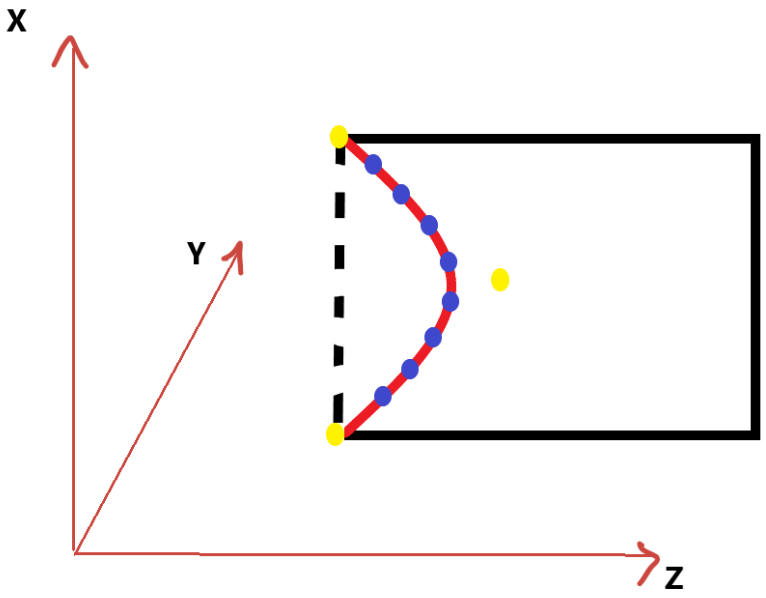


Abbildung 3.4: Deformation in der linken Kante des Rechtecks

.

Um eine Kante eines Rechtecks zu deformieren, wird ein Stützpunkt zwischen dem Anfang und dem Ende dieser Kante hinzugefügt (wie in obige Abbildung gezeigt). Dieser Stützpunkt sollte nicht auf der Linie liegen und dient als 3.Kontrollpunkt, der zusammen mit dem Anfangs- und Endpunkt dieser Kante eine Bezierkurve (rote Kurve in der Abbildung) bildet.

3.2.1 Rastern

Damit Deformation der Kante auch weiterhin von Deformation der Oberfläche verarbeitet werden kann, muss die deformierte Kante Form gerastert werden (Abschnitt 3.1.1). Allerdings ist es in diesem Fall nicht sinnvoll, den Scanline-Algorithmus zu verwenden, da das Auffinden des Schnittpunkts zwischen einer Kurve und einer Geraden viel Rechenaufwand erfordert. Daher wird das Rastern von 'bend_side' auf seine eigene Weise spezialisiert.

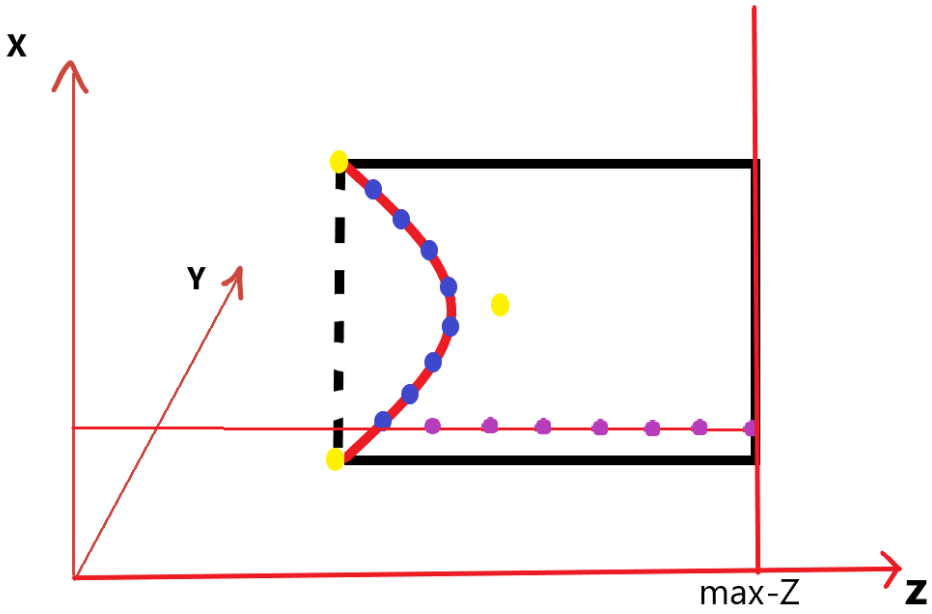


Abbildung 3.5: Rasterung einer Linie in der ”bend_side”-Form

.

Die Abbildung 3.5 zeigt das Rastern der deformierten Form in der Abbildung 3.4. Anstatt den Schnittpunkt zwischen der zum Scannen verwendeten Linie und den Kanten des Polygons zu finden, werden die Punkte auf der Kurve verwendet, um das Raster auszuführen. Aus den Punkten auf der Kurve (blaue Punkte) werden Rasterpunkte (lila Punkte) generiert, sodass der Rasterpunkt denselben x-Wert wie die Kurvenpunkte hat und der z-Wert nicht größer als der maximale z-Wert des Polygons ist.

3.2.2 Triangulieren

Das Triangulieren von der Deformation der Kante ist fast ähnlich sowie die Deformation der Oberfläche.Die Delaunay-Triangulation wird wieder verwendet, aber in diese Deformation müssen die Dreiecke, die aus die Punkten der Kurve erstellt werden, aus der Triangulation entfernen. Der Grund dafür ist, die Dreiecke, die nur aus Kurvenpunkten erzeugt werden, gehören nicht zu der deformierte Form.

3.3 Integrieren in der Form-Grammatik

In der Form-Grammatik werden Formen oft in UnterFormen zerlegt, die sich in der 2D-Ebene befinden, um die Bearbeitung zu erleichtern, daher beschäftigen sich viele Regeln der Form-Grammatik nur mit der Bearbeitung von Formen in 2D-Ebene. Wenn die Oberfläche der Form deformiert wird, wandelt jedoch eine 2D-Form in eine 3D-Form um. Dies führt zu dem Problem, dass, die Regeln, die nur 2D-Formen verarbeiten kann, nach Integrieren der Deformation funktioniert nicht mehr. Um sicherzustellen, dass diese Regeln noch gültig ist, wird das Deformieren der Oberfläche als letzter Schritt in der Grammatik sein.

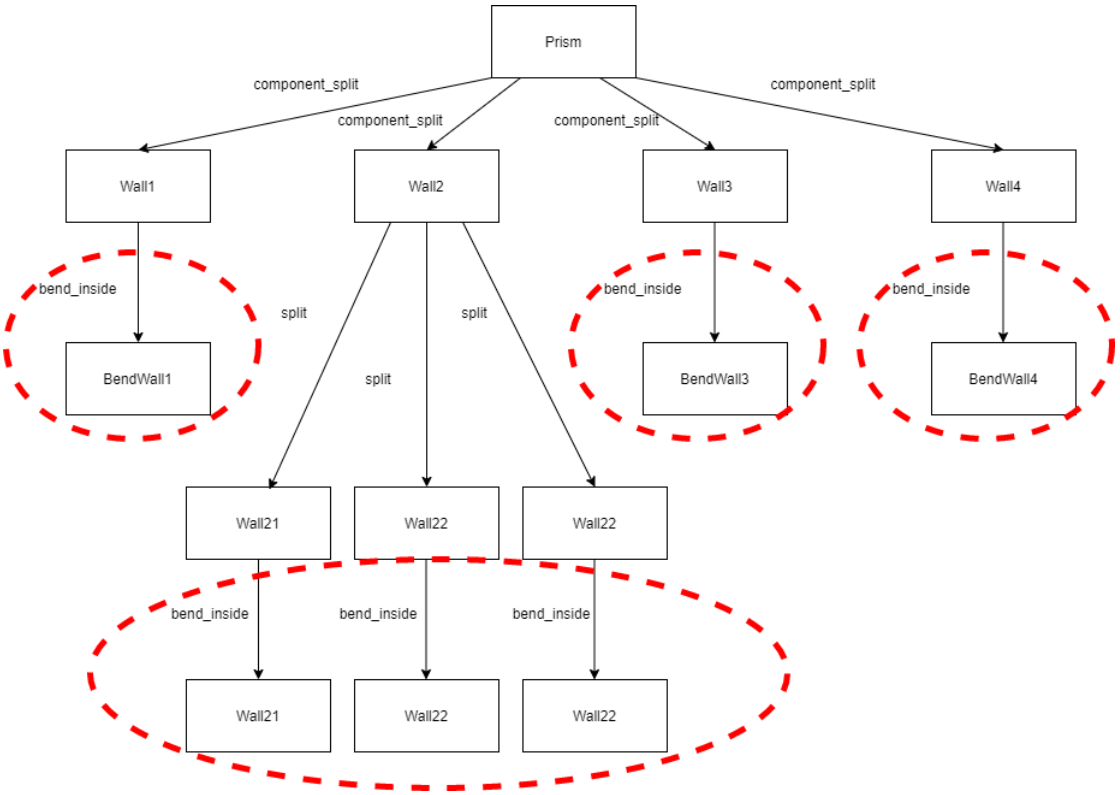


Abbildung 3.6: Beispiel für eine Grammatik mit der ”bend_inside”Operation

.

Die Abbildung 3.6 zeigt die Stelle, an der die Deformation der Oberfläche (in diesem Beispiel wird diese Deformation als ”bend_inside” genannt) angewendet wird. Nachdem die Prism-Form mithilfe der Regel ”component_split” in vier Wall-Formen aufgeteilt wird, dann wird Form ”Wall2” durch Regel ”split” weiter in 3 Teile aufgeteilt. Schließlich werden alle

Formen, die ohne untergeordnete Formen sind, "bend_inside"Regel angewendet. Die Platzierung von ”bend_inside”-Regel am Ende jedes Zweiges von dem Baum der Formen vermeidet auch die Störung der zukünftigen Entwicklung von Regeln, die nur in der 2D-Ebene operieren können.

3.3.1 Deformieren UnterFormen

Bei der Deformation der Oberfläche von der Form werden die UnterFormen, die in gleiche Ebene, durch eine gemeinsame Kurve deformiert. Diese Kurve wird auf Koordinaten der UnterForm transformiert, und die Punkten, die im Bereich dieser UnterForm, deformiert. Das folgende Beispiel zeigt die Deformation einer Form, die aus 4 UnterFormen besteht.

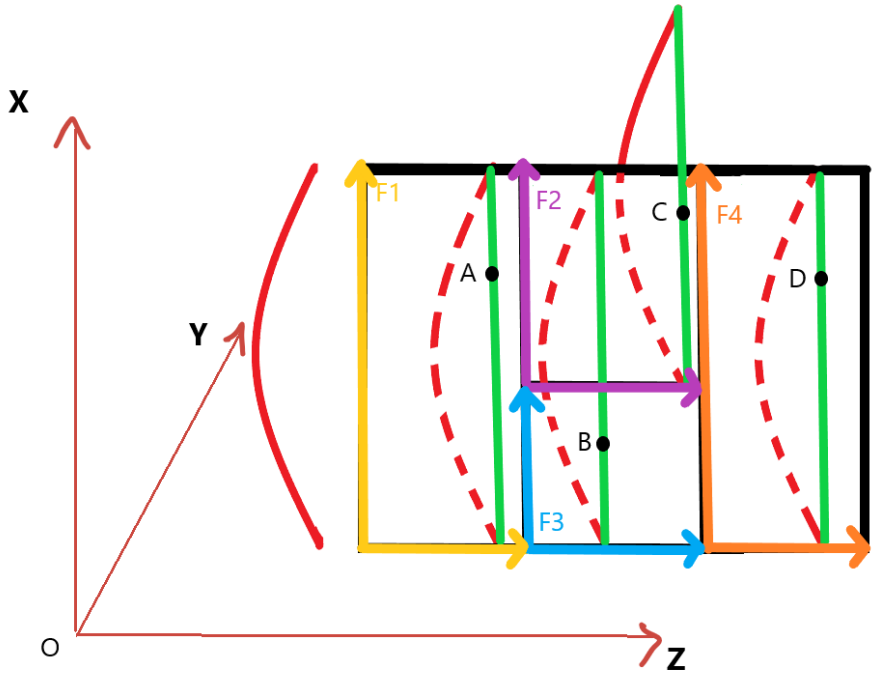


Abbildung 3.7: Die Deformation in unterschiedliche Koordinatensystemen
Das schwarze Rechteck ist in 4 UnterFormen unterteilt, F1, F2, F3 und F4
A, B, C und D sind jeweils die Punkte von Abbildung F1, F2, F3 und F4

.

Die roten Kurven sowie die grüne Gerade in der obigen Abbildung sind die Linien, die schon in der Abbildung 3.1.2 erwähnt waren. Sie werden parallel zur x-Achse platziert und entlang der z-Achse verschoben, um die neue Position der Punkten von der Formen zu ermitteln.

Zum Ermitteln die Position von Punkten der UnterFormen nach der Deformation, wird die Kurve in das lokale Koordinatensystem der UnterForm transformiert. Danach können die neuen Positionen der Kontrollpunkte nach der Deformation durch die Formel in der Abbildung 3.1.2 ermittelt werden.

Auch in der Abbildung entsteht hier ein Problem, das heißt, nach der Umwandlung in das lokale Koordinatensystem der Form F2 wurde die Kurve angehoben, weil das lokale Koordinatensystem von F2 die ursprünglichen Koordinaten höher als das globale Koordinatensystem hat. Befindet sich die Kurve an dieser Position, werden die Punkte der UnterForm F2 falsch deformiert, sodass muss die Kurve durch ein Offset unten verschieben. Um die Offset für das Transformieren der Kurve zu berechnen, müssen der ursprüngliche Punkt des globalen Koordinatensystems im lokalen Koordinatensystem transformiert, dann kann das Offset als die Differenz des z-Wertes zwischen dem Ursprung des lokalen Koordinatensystems und dem Ursprung des globalen Koordinatensystems berechnet werden.

4 Umsetzung

In diesem Kapitel wird die Umsetzung der beiden Deformationen, die schon im Konzept erwähnt sind, vorgestellt. Diese Deformationen werden als Erweiterungen des vorhandenen Projekts der Gruppe Computergrafik HAW Hamburg aufgebaut. In diesem Projekt ist die verwendete Grafik-Engine "JMonkey", eine in Java geschriebene Spiel-Engine. Manche grundlegenden Regeln von Figur-Grammatik (extrude, component_split, split ...) werden durch die Teilnehmer des Projekts implementiert. Um die Einheitlichkeit der Codekonvention zu wahren, werden die beiden Deformation unter den englischen Namen ”bend_surface” und ”bend_edge” implementiert.

4.1 Aufbau der Regel der Figur-Grammatik

Jeder Regel bestehen in diesem Projekt aus drei Komponenten (ShapeOperation, Shape und MeshGenerator).

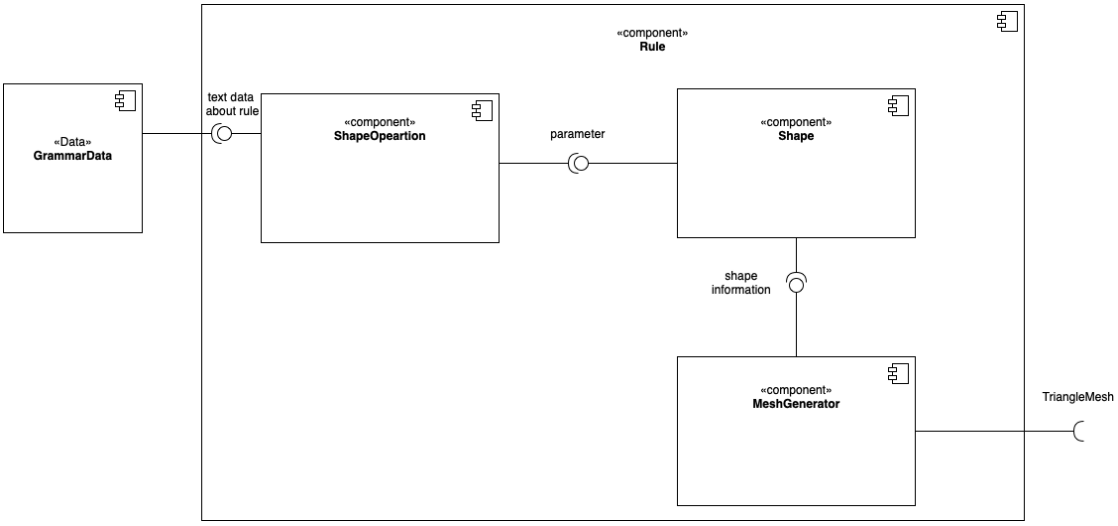


Abbildung 4.1: Komponentendiagramm der Regel der Figur-Grammatik

.

- ShapeOperation verarbeitet Textinformationen aus Grammatikdaten und konvertiert sie in Parameter, um Figur zu formen.
- Shape ist die Datenstruktur, die verwendet wird, um die aus der jeweiligen Grammatik generierten Figuren zu definieren.
- MeshGenerator verwendet die Shape-Informationen, um ein dreieckiges Netz zu erstellen. Das Dreiecksnetz wird dann mithilfe der Jmonkey-Engine in eine Grafik umgewandelt.

Basieren auf der obigen Architektur werden die folgende Java-Klasse für die Deformationen erstellt:

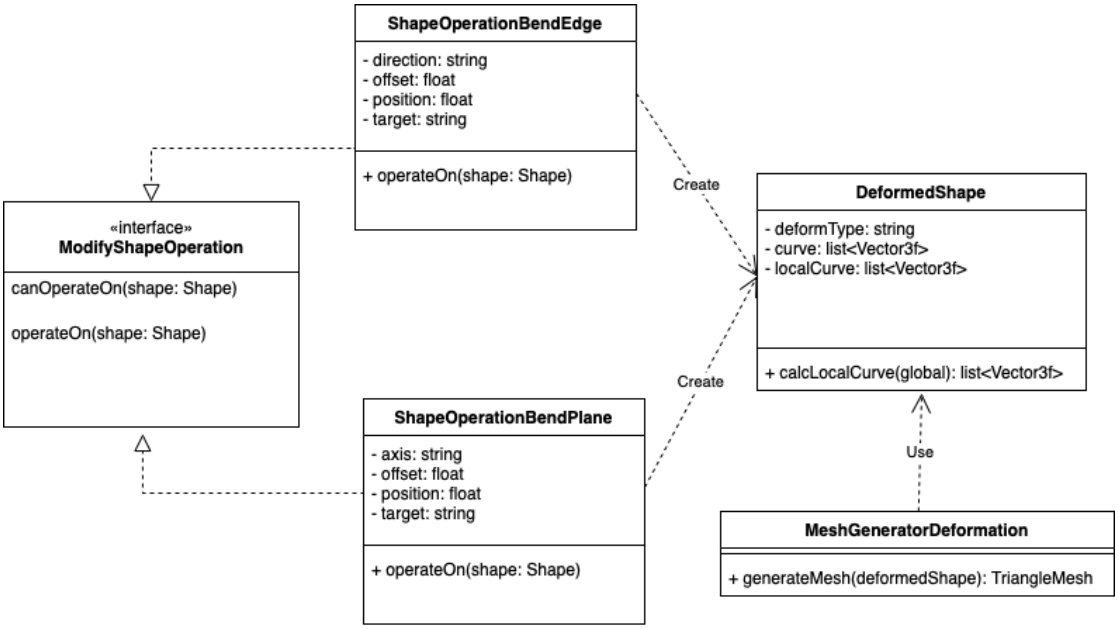


Abbildung 4.2: Klassendiagramm

.

Im Klassendiagramm entsprechen die Klasse ShapeOperationBendPlane und ShapeOperationBendEdge den 2 Deformationen, die schon im Konzept erwähnt (Deformation für Oberfläche und Deformation für Kante). Diese beiden Klassen, die gleichen Felder und Funktionen haben, werden sie dennoch in zwei separate Klassen unterteilt, da die Funktionseingabe "operateOn()" der beiden Klassen unterschiedliche Eingabe annehmen können. Während ShapeOperationBendPlane jedes Polygon in der 2D-Ebene deformieren kann, kann die ShapeOperationBendPlane-Klasse nur rechteckige Polygone verarbeiten. Obwohl sie der unterschiedliche Eingabetyp haben, verwenden sie beide gleich Ausgabetyp, nämlich DeformedShape. Diese Ausgaben werden weiterhin von der MeshGeratorDeformation-Klasse verarbeitet, um dreieckige Netz zu erstellen.

4.2 bend_surface

ShapeOperationBendPlane ist die Implementierung des Deformierens der Oberfläche der Figur. Um diese Operation aufzurufen, wird die folgende Syntax verwendet:

$$A-->bend\_surface(<Offset>,<Position>,<Axis>,<Target>)B$$

Das folgende Bild zeigt die Rolle der Parameter für die Deformation:

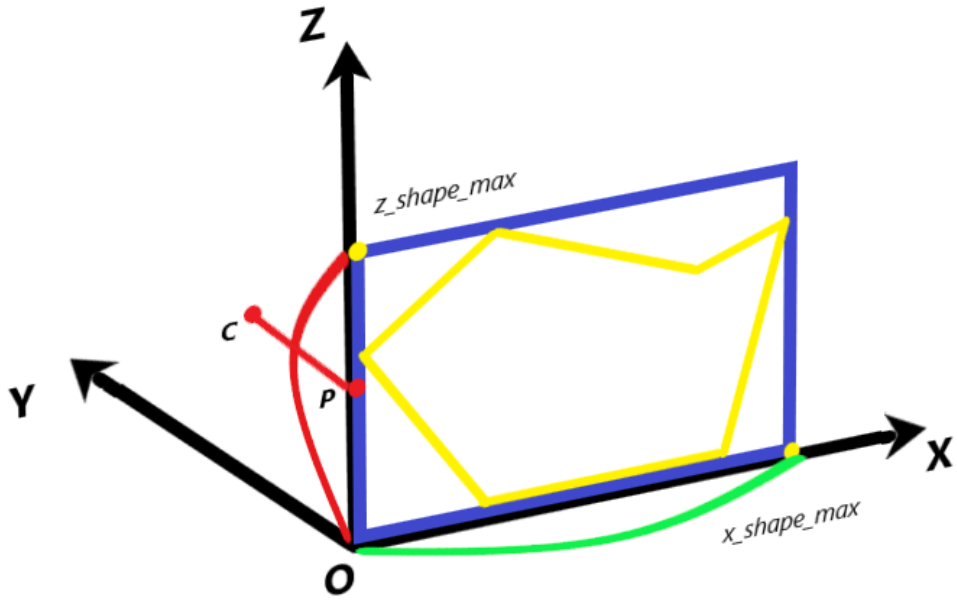


Abbildung 4.3: Deformieren die Oberfläche eines Polygons

.

In diesem Projekt werden die 2D-Figuren im XOZ-Koordinatensystem platziert, daher gibt es zwei Möglichkeiten, die Kurve zu initialisieren. Diese Auswahl wird über den Parameter ”Axis” übermittelt. Wenn diese Auswahl die x-Achse ist, wird die Kurve entlang der x-Achse gesetzt, und wenn das Z-Symbol im Parameter ”Axis” eingetragen ist, wird die Kurve auf die y-Achse entlang gesetzt. Dieser Parameter beeinflusst auch die Bestimmung der Start- und Endpunkte der Kurve. Der Startpunkt der Kurve ist immer der Ursprung der Koordination, während der Wert x und z des Endpunkts vom Parameter ”Axis” der Grammatik abhängt. Wenn Achse als X ausgewählt wird, hat der Endpunkt einen X-Wert gleich dem x-max des Scope der Figur, und die Werte von y und z entsprechen dem Wert des Ursprungs. Wenn die z-Achse ausgewählt ist, ist der Wert von z des Endpunkts ebenfalls z-max des Scope und die übrigen Werte sind 0.

Zusätzlich zu den Start- und Endpunkten muss ein Kontrollpunkt in der Mitte definiert werden, um die Kurve zu erstellen. Dieser Kontrollpunkt wird durch 2 Parameter Position und Offset bestimmt. Parameterposition ist der Abstand vom Anfangspunkt der Kurve zum Bild

des Kontrollpunkts auf der Linie, die von Anfang und Ende der Kurve gebildet wird (Punkt P in Abbildung 4.3), während Offset der Abstand zwischen dem Kontrollpunkt und die XOZ-Ebene. Die von den 3 oben genannten Punkten erzeugte Kurve wird mit der Listendatenstruktur gespeichert, um die Konvertierung in lokale Koordinatensysteme zu erleichtern.

4.3 bend_edge

Die Grammatik von ShapeOperationBendEdge hat eine Syntax, die fast ähnliche wie ShapeOperationBendPlane.

$$A -- > bend\_edge(< Offset >, < Position >, < Direction >, < Target >)B$$

Der einzige Unterschied besteht darin, dass der Parameter ”Axis” (mit zwei Optionen ”X” oder ”Z”) durch den Parameter ”Direction” mit vier Optionen(Westen, Süden, Osten und Norden) ersetzt wird.

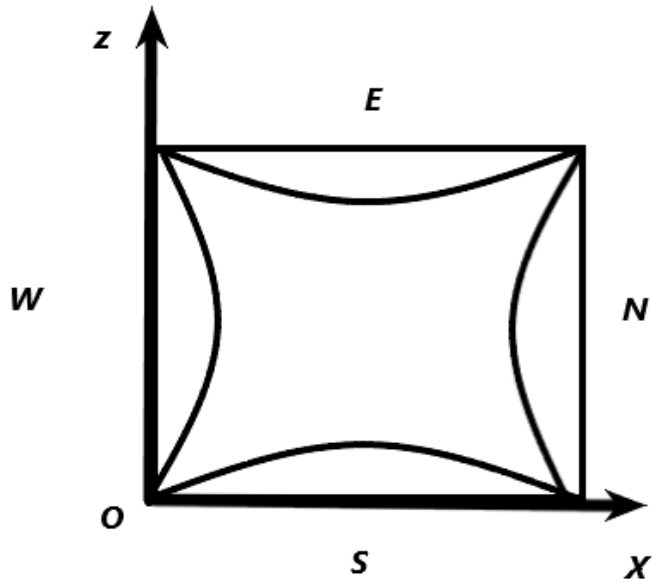


Abbildung 4.4: Deformieren die Kante eines Rechteckes

.

Diese Optionen entsprechen den vier Kanten eines Rechtecks. Durch diesen Parameter wird die zu deformierende Kante bestimmt. Die Kurve, um dieses Deformieren durchzuführen, hat die gleichen Start- und Endpunkte wie die ausgewählte Kante, und der mittlere Kontrollpunkt wird durch Offset und Position wie in der ShapeOperationBendPlane-Klasse definiert. Der Offset ist in diesem Fall jedoch nicht mehr der Abstand vom Kontrollpunkt zur XOZ-Ebene, da dieser Vorgang in der 2D-Ebene stattfindet, sondern der Offset ist der Abstand vom Kontrollpunkt zur Kante, die deformieren muss.

4.4 Die deformierte Figur (DeformedShape)

Die DeformedShape-Klasse erbt die Eigenschaften der Shape-Klasse (Abbildung 4.2), daher enthält eine DeformedShape-Instanz auch Informationen über den Namen, Vorgänge und eine Matrix zur Umrechnung einer Punktkoordinate im lokalen Koordinaten.

Der Parameter "Target"von Regel ”bend_surface” oder Regel ”bend_edge” repräsentiert den Symbolnamen der zu verformenden Figur. Durch diesen Parameter ist es möglich zu wissen, dass diese beiden Regeln auf der zu verformenden Figur oder ihre Kinder angewendet werden.

Falls der Symbolname der Figur gleich dem Parameter ”Target” ist, kann die von den obigen 2 Regeln erzeugte Kurve verwendet werden, um die Figur zu verformen. Im Gegenteil, wenn der Symbolname von Figur nicht mit Parameter ”Target” identisch, d.h, die angewendete Figur eine Unterfigur ist, und eine globale Kurve kann nicht direkt für eine Unterfigur verwendet werden.

Die globale Kurve bzw. ihre Kontrollpunkte müssen durch Matrizen von Unterformen in das lokale Koordinatensystem umgewandelt werden.Um alle für die Konvertierung benötigten Matrizen zu erhalten, wird der folgende Code verwendet:

Algorithm 1
Finden alle lokalen Transformationsmatrizen zwischen globale und lokale Koordinatensystems

```
function GetLocalCoordinations(current_shape, target_name)
    local_coordinates_list := empty_matrix_array
    while current_shape_name ≠ target_name do
        local_coordinates_list add current_shape_local_coordinate
        current_shape ← current_shape_parent
    return local_coordinates_list
```

Nachdem alle lokalen Transformationsmatrizen erhalten wurden, werden die Kontrollpunkte der Hauptkurve in das lokale Koordinatensystem umgewandelt, indem die Kontrollpunkte nacheinander mit den Transformationsmatrizen multipliziert werden.

4.5 Mesh Generierung

Diesem nächsten Abschnitt wird demonstrieren, wie das Konzept der Triangulierung (Abschnitt 3.1.3 und 3.2.2) implementiert wird. Der Schwerpunkt dieses Abschnitts liegt auf der Erstellung von Superdreiecken zur Implementierung des Bowyer-Watson-Algorithmus. Außerdem wird die Entfernung der redundanten Dreiecke von bend_edge Figur in diesem Abschnitt erklärt.

4.6 Umsetzung von Bowyer-Watson-Algorithmus

```
function BowyerWatson (pointList)
   // pointList is a set of coordinates defining the points to be triangulated
   triangulation := empty triangle mesh data structure
   add super-triangle to triangulation // must be large enough to completely contain all the points in pointList
   for each point in pointList do // add all the points one at a time to the triangulation
      badTriangles := empty set
      for each triangle in triangulation do // first find all the triangles that are no longer valid due to the insertion
         if point is inside circumcircle of triangle
            add triangle to badTriangles
      polygon := empty set
      for each triangle in badTriangles do // find the boundary of the polygonal hole
         for each edge in triangle do
            if edge is not shared by any other triangles in badTriangles
               add edge to polygon
      for each triangle in badTriangles do // remove them from the data structure
         remove triangle from triangulation
      for each edge in polygon do // re-triangulate the polygonal hole
         newTri := form a triangle from edge to point
         add newTri to triangulation
   for each triangle in triangulation // done inserting points, now clean up
      if triangle contains a vertex from original super-triangle
         remove triangle from triangulation
   return triangulation
```

Abbildung 4.5: Pseudocode von Bowyer–Watson Algorithmus

.

Im obigen Code werden die Punkte der Figur nacheinander im Superdreieck hinzugefügt. Ein hinzugefügter Punkt wird mit vorhandenen Punkten im Superdreieck verbunden, um neue Dreiecke zu erstellen. Die erstellten Dreiecke werden dann mit dem neu hinzugefügten Punkt überprüft, ob es sich noch um ein Delaunay-Dreieck handelt. Dreiecke, die die Bedingung nicht mehr erfüllen, werden zu einer Liste von Dreiecken hinzugefügt, die eliminiert werden müssen.

Das Speichern der zu entfernenden Dreiecke ist für einige Programmiersprachen erforderlich, die ein Element nicht löschen können, wenn die Liste dieses Element durchläuft wird.In

diesem Projekt kann die Iterator-Klasse von Java verwendet werden, um ein Dreieck direkt in der Schleife zu löschen, dadurch Speicher gespart als auch die Leistung des Mesh-Generierungsprozesses erhöht wird.

4.6.1 Entfernen die falschen Verbindungen

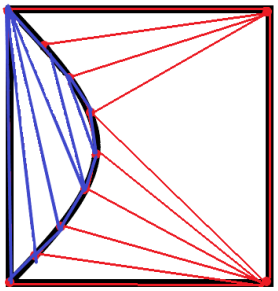


Abbildung 4.6: Dreiecksnetz mit redundanten Dreiecken

.

Bei der Anwendung des Bowyer-Watson-Algorithmus auf die durch die ben_edge-Regel erzeugten Figuren werden einige unerwünschte Dreiecke werden erzeugt (die blauen Dreiecke in Abbildung 4.6). Diese Dreiecke werden aus Punkten auf der Kurve erstellt. Sie sind immer noch im Scope der Figur, deswegen werden sie auch durch den Algorithmus verbunden. Diese Dreiecke sollten eliminiert werden, indem überprüft wird, ob alle drei Eckpunkte des Dreiecks gekrümmt sind. Wenn diese Dreiecke nur 2 oder weniger Scheitelpunkte auf der Kurve haben, sind die Dreiecke (rote Dreiecke) gültig.

5 Evaluation

In diesem Abschnitt werden die beiden Operationen bend_plane und bend_edge basierend auf der Laufzeit sowie der Anzahl der Scheitelpunkte und Dreiecke, die diese beiden Operationen erzeugen, analysiert. Der Zweck dieser Messung besteht darin, Daten zu sammeln, um sie in Zukunft mit anderen Deformationsmethoden vergleichen zu können.

In diesem Experiment werden die Metrik während der Verformung eines einfachen Hauses (Abbildung 5.2) gemessen. Der Deformationsaufwand wird durch Messung der Zeitdifferenz sowie der Anzahl der Eckpunkte und Dreiecke vor und nach der Anwendung der Deformationsregeln berechnet.Die Laufzeit wird vom Parsen der Grammatik bis zur Fertigstellung des Dreiecksnetzes von dem Gebäude gemessen.

Wie den Messergebnissen zu entnehmen ist, erhöht sich die Zeit nach dem Aufbringen der Verformung deutlich, ca. 50 Millisekunden für deformiertes Gebäude A und ca. 90 Millisekunden für Gebäude B. Dieser Laufzeitunterschied ist recht gering und beeinflusst den Bau von Gebäuden nicht wesentlich. Die Anzahl der neuen Scheitelpunkte und Dreiecke ist jedoch viel größer als zuvor, dies liegt an der Verwendung des Scanline-Algorithmus-Algorithmus. Das Erhöhen der Anzahl von Scheitelpunkten und Dreiecken erfordert mehr Speicher beim Transformationsprozess des Gebäudes. In Bezug auf die Grafik kann die Deformation der beiden Gebäude deutlich gesehen werden, obwohl die deformierte Ebene nicht so glatt ist, da die Linien, die die Eckpunkte in der Form verbinden, noch sichtbar sind.

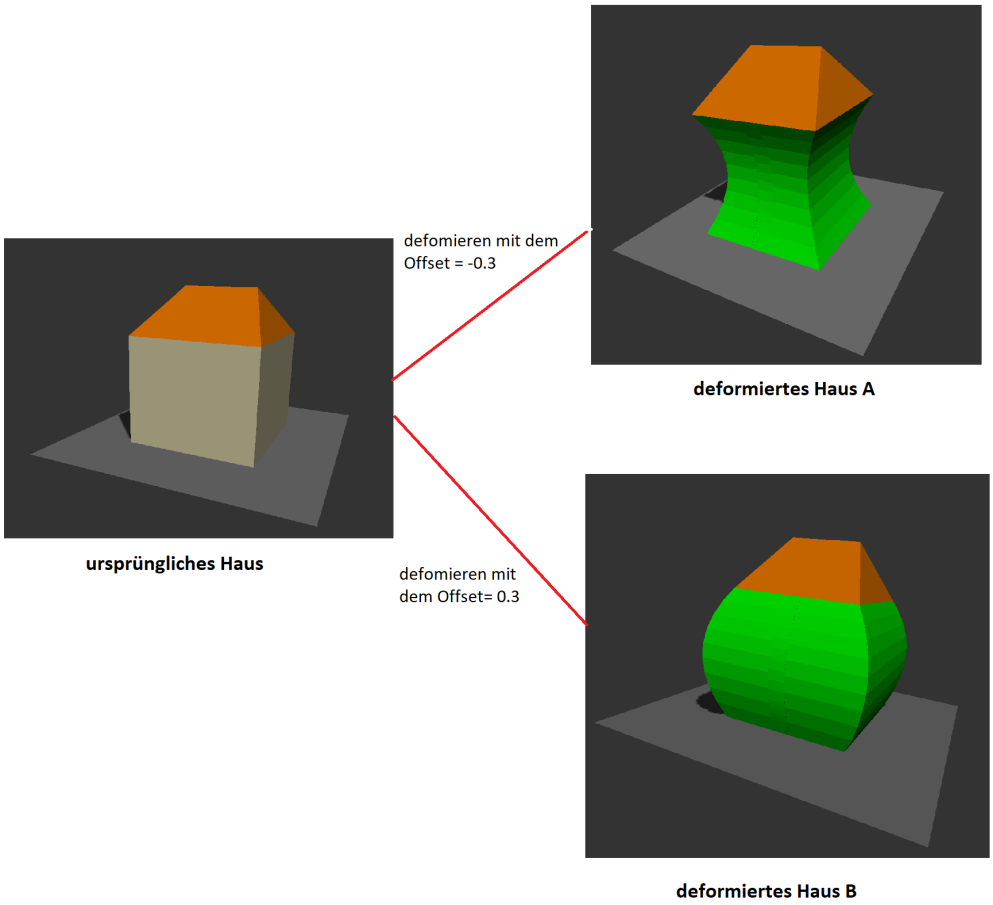


Abbildung 5.1: Deformieren eines Gebäudes

.

	ursprüngliches Haus	deformiertes Haus A	deformiertes Haus B
Laufzeit (ms)	6	58	93
Scheitelpunkte	46	632	784
Dreiecke	28	944	1248

Tabelle 5.1: durchschnittliche Messdaten nach 10 Versuchen

Verformung als Regel der Grammatik umsetzen bringt noch einen weiteren Minuspunkt, nämlich dass die Formen nach der Verformung nicht zusammenpassen.

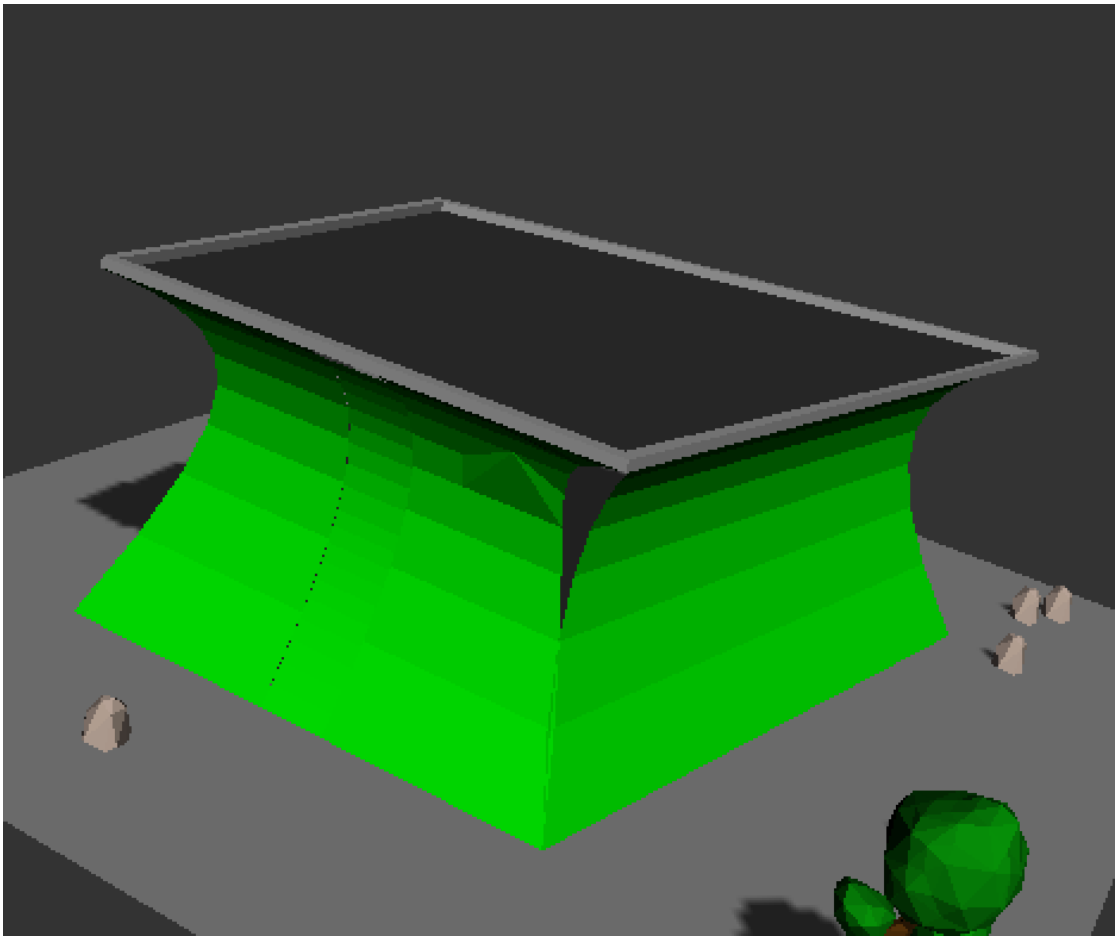


Abbildung 5.2: Das Gebäude mit dem Boden,der nicht quadratisch nicht

.

Wie in der Abbildung zu sehen ist, stoßen die Wände des Gebäudes nach der Verformung aneinander, wenn der Boden des Gebäudes kein Rechteck ist. Der Grund dafür ist, dass, wenn eine Wand nicht rechtwinklig zur Nachbarwand ist, die Kurve der „bend_edge“-Regel (4.3) dieser Wand auch nicht mehr mit der Kurve der „bend_surface“-Regel (4.3) der Nachbarwand übereinstimmt ist. Dies schränkt die Fähigkeiten von Deformation ein, die nur auf Gebäude angewendet werden können, deren Grundfläche ein Rechteck ist.

6 Fazit

Das Ziel dieser Bachelorarbeit war es, eine Deformation zu entwickeln, die auf Gebäude anwendbar ist, die durch CGA-Shape generiert werden. Anhand der Ergebnisse von der Evaluation kann bestätigt werden, dass das Ziel dieser Arbeit erreicht wurde. Die Implementierung von Deformation als Regeln der Grammatik ist ein wichtiger Punkt diese Arbeit. Dadurch wird die Struktur des Projekts gehalten, und die Deformation wirkt sich weder auf die alten Regeln noch auf die Entwicklung neuer Regeln in der Zukunft aus. Die Deformation hat jedoch auch einige Nachteile, da die Anzahl der durch die Deformation erzeugten Eckpunkte und Dreiecke ziemlich groß ist und diese Deformation nur für Formen, die in der 2D-Ebene liegen, gilt. Die Verformung kann in Zukunft verbessert werden, indem ein besser geeigneter Algorithmus als Scan-Line-Algorithmus verwendet wird, um Stützpunkte für die Verformung zu erzeugen.

Literaturverzeichnis

[1] Thomas W. Sederberg and Scott R. Parry: Free-form deformation of solid geometric models. (1986)

[2] Anjyo, Ken ichi ; Hiramitsu, Katsuaki: Stylized Highlights or Cartoon. (2003)

[3] Anjyo, William B. Ken-ichi: Tweakable Light and Shade for Cartoon Animation. (2006)

[4] Bowyer, Adrian: Computing Dirichlet tessellations. (1981)

[5] De Loera, Jesús ; Rambau, Jörg ; Santos, Francisco: *Triangulations, Structures for Algorithms and Applications. Algorithms and Computation in Mathematics. Vol. 25. Springer.* 2010

[6] Gallier, Jean: Curves and Surfaces In Geometric Modeling: Theory And Algorithms. (2018)

[7] Gudala Rajesh, Jitendra P. K.: Deformation Modeling of a Flexible Instrument Using a Bézier Curve. (2021)

[8] Holtmeier, Mathias: Beispiel-getriebene prozedurale Geb aude-Generierung unter Anwendung einer Shape-Grammar. (2020)

[9] James D. Foley, Steven K. Feiner John Hughes Morgan McGuire David F. S. ; Akeley, Kurt: *Computer Graphics: Principles and Practice.* 1995. – ISBN 9780201848403, 0201848406

[10] John F.Hughes, Morgan Mcguire David F.Sklar James D.Foley Steven K.Feiner Kurt A.: *Computer Graphics Principles and Practice (Third Edition).* Reading, Massachusetts : Addison-Wesley, 2014. – ISBN 978-0-321-39952-6

[11] Juan, Christina de ; Bodenheimer, Bobby: Cartoon Textures. (2014)

[12] L. Hilario, N. M. ; Mora, M.C.: Free Form based active contours for image segmentation and free space perception. (2016)

[13] Liu Bi: Computing offsets of curves on triangular mesh. (2015)

[14] Nasr, Nabil ; Higgett, Nick: Traditional cartoon style 3D computer animation. (2002)

[15] Ouiddad Labbani-I, Pauline Merveilleux-O ; Ruatta, Olivier: A tensor optimization algorithm for Bézier Shape Deformation. (2015)

[16] Peter Wonka, Francois S. ; Ribarsky, William: Instant Architecture. (2003)

[17] Riyad Al-Rousan, M. S.: Stylized line drawing and shading effects for cartoon rendering. (2014)

[18] Shene, C. K: Finding a Point on a Bézier Curve: De Casteljau's Algorithm. (2012)

[19] Stefan Lienhard, Cheryl La, Pascal Müller, Peter Wonka and Mark Pauly: Design transformation for rule based procedural modelling. (2017)

[20] Stiny, G. ; Gips, J: Shape grammars and the generative specification of painting and sculpture. (1972)

[21] Thomas Sederberg: Bézier curves. (2005)

[22] Wonka, Peter ; Muller, Pascal ; Haegler, Simon ; Ulmer, Andreas ; Gool, Luc V.: Procedural Modeling of Buildings. (2006)

[23] Z. M. Chen, Y. Chen, F. Liang: A new offset approach for curves on triangle mesh surfaces. (2006)

[24] Zmugg, René ; Thaller, Wolfgang ; Krispel, Ulrich ; Edelsbrunner, Johannes ; Havemann, Sven ; W.Fellner, Dieter: Procedural Architecture Using Deformation-aware Split Grammars. (2015)

A Anhang

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