Technische Universität München

Ingenieurfakultät Bau Geo Umwelt

Lehrstuhl für Computergestützte Modellierung und Simulation

Entwicklung einer Formgrammatik für die statische optimierte Segmentierung von Brückenbauteilen in Modulbauweise

Bachelorthesis

für den Bachelor of Science Studiengang Ingenieurwissenschaften

Autor:	Stefan Huber
Matrikelnummer:	03717732
1. Betreuer:	Prof. Dr.-Ing. André Borrmann
2. Betreuer:	M. Sc. Lothar Kolbeck

Ausgabedatum:	15. Mai 2021
Abgabedatum:	15. Oktober 2021

Abstract

In this thesis, a shape grammar is created to divide a simplified bridge geometry in small planar modules in two-dimensions. The grammar can achieve a generative segmentation of the geometry. The shape grammar is programmed with the programming interface of the Sortal Grammar Interpreter (SGI). The segmentation suggestions are exported from CAD to numerical calculation program, in which an automated evaluation is implemented. The rating takes three points into account: The angle in a segment length of the edges and the main stress directions of joints. For the first two factors, it is necessary to identify the individual segments in the geometry. Therefore the geometry is transformed into a graph. Individual segments are then recognized by an algorithm that finds the shortest path. A finite element model is used, in which the main stress directions are computed. Finally, advantages as well as problems of the solution are described and evaluated.

Zusammenfassung

In dieser Arbeit wird für vereinfachte zwei dimensionale Brückengeometrie eine Formgrammatik erstellt, die kleine planare Module erzeugt und eine zufällige Segmentierung erreicht. Dafür wird der Ansatz des Generative Designs benutzt. Die Formgrammatik ist mit dem Sortal Grammar Interpreter (SGI) programmiert. Die Segmentierungsvorschläge werden von einem CAD-Programm zu einem Berechnungsprogramm exportiert, worin die Bewertung implementiert ist. Die Bewertung bezieht sich auf folgende drei Punkte Winkel in einem Segment, Länge der Kanten und Hauptspannungsrichtungen der Fugen. Für die ersten beiden Faktoren ist es notwendig die einzelnen Segmente in der Geometrie zu identifizieren. Dazu wird die Geometrie in einen Graphen Repräsentation transformiert. Anschließend werden durch einen modifizierten Dijkstra Algorithmus, die einzelnen Segmente erkannt. Als strukturelles Kriterium wurde ein Finite Element Model erstellt, in den die Hauptspannungsrichtungen berechnet werden. Abschließend werden Vorteile aber auch Probleme der Lösung aufgezeigt und bewertet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

RHS	RightHandSide
LHS	LeftHandSide
FEM	Finite-Elemente-Methode
CAD	Computer-Aided Design

1 Einführung und Motivation

1.1 Einführung

Brücken sind ein wichtiger Bestandteil der Verkehrsinfrastruktur, deshalb muss der Bau von Brücken schnell, effizient, aber auch günstig sein. Der derzeitige Stand der Technik ist es Brücken vor Ort ganz oder in großen Abschnitten herzustellen und dann zu verbinden. Dieses Vorgehen hat zur Folge, dass die Qualität des Bauwerkes von der Witterung abhängt, eine Vielzahl von Facharbeitern notwendig sind und somit eine lange Bauzeit benötigt wird. Ein Ansatz diese Probleme zu umgehen ist es die Brücke in kompakte Fertigteile zu unterteilen, die in Fabriken durch 3D-Druck oder Guss hergestellt werden können. Diese können unkompliziert transportiert und dann montiert werden. Diese Segmentierung ist allerdings nicht trivial, da sich Brücken in Lage und Geometrie unterscheiden. Daher ist es notwendig, adaptive und automatisierte Verfahren zur Segmentierung zu entwickeln. Eine Konzeptzeichnung einer Brücke in Modulbauweise ist in Abbildung 1 dargestellt.

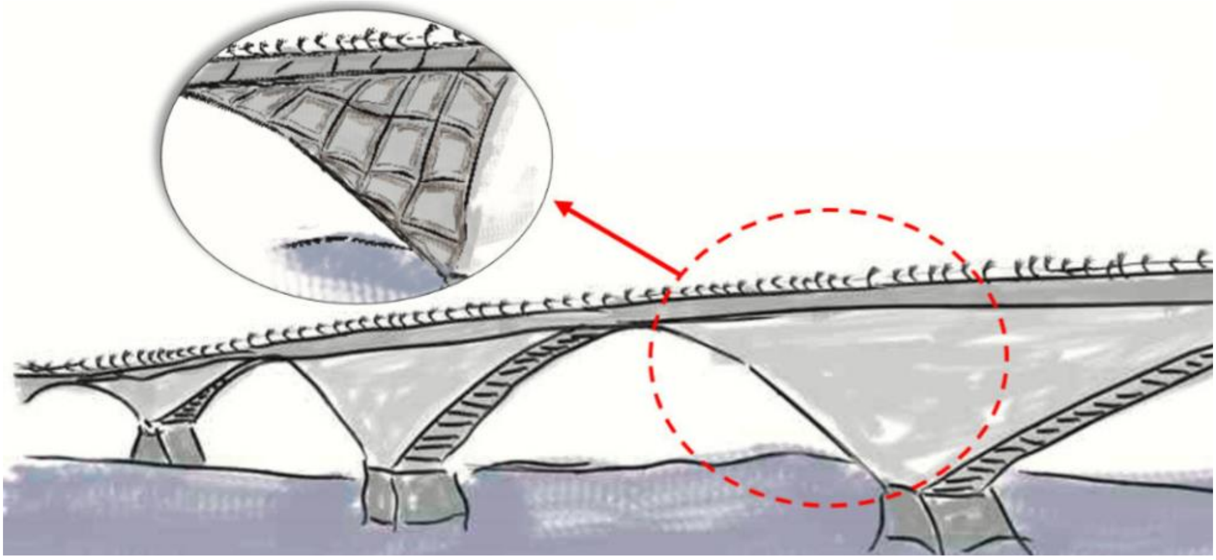


Abbildung 1: Konzeptzeichnung einer Brücke in Modulbauweise

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es eine vereinfachte zwei dimensionale (2D) Brückengeometrie mit Hilfe einer Formgrammatik automatisch in einem CAD Programm zu segmentieren.

Die Entwicklung eines Prototyps für 2D dient der Erprobung des Formalismus, welcher im Falle der Eignung auf den realistischeren 3D-Fall übertragen werden kann.

Zusätzlich werden die segmentierten Geometrien anhand von statischer und produktionsbedingter Gesichtspunkte bewertet. Dabei werden keine zusätzlichen Iterationen durchgeführt, die die bereits bestehenden Segmentierungen nachträglich verändern.

1.3 Aufbau der Arbeit

In Kapitel 2 wird die Brückengeometrie abstrahiert, der Aufbau der Segmente sowie der Ansatz des Generative Designs vorgestellt. Kapitel 3 geht näher auf Formgrammatiken im Allgemeinen ein. Dabei stehen Definitionen und Mechanismen im Vordergrund. Kapitel 4 erklärt das Tool *Sortal*, dessen Implementierung, Einsatzmöglichkeiten, Syntax und Einschränkungen. In Kapitel 5 wird dann der Softwareprototyp diskutiert, der regelbasiert Segmentierungen erzeugt. Die Bewertung jener Segmentierungen anhand festgelegten Faktoren wird in Kapitel 6 beschrieben. Abschließend werden in Kapitel 7 Vorteile und Probleme des verwendeten Tools aufgezeigt.

2 Konstruktiver Kontext

2.1 Aufbau

Die gewählte Geometrie für die Segmentierung wird in Anlehnung an dem Überbau einer Balkenbrücke erstellt (siehe Abbildung 2). Ebenfalls weist die Geometrie eine Ähnlichkeit zu einem Plattenbalken auf.

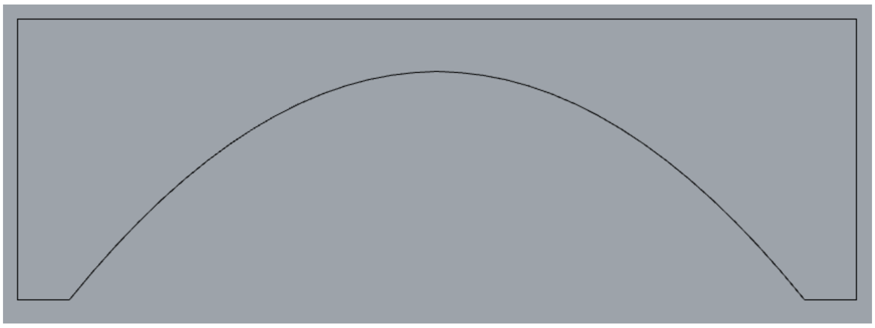


Abbildung 2: Abstrahierte Geometrie für die Segmentierung

Die untere Begrenzung wurde als Parabel symmetrisch in den Aufbau gelegt. Diese vereinfachte Form wurde gewählt, damit sowohl flache als auch hohe Bögen getestet werden können. Diese Geometrie lässt sich auf mehrere Felder erweitern, durch die geraden Linien an den Seiten. Die Geometrie entspricht zudem einer Scheibentragwirkung, für die die angedachte Modulbauweise konzipiert wurde.

Die statischen Faktoren beziehen sich auf die Hauptspannungsrichtungen der Segmente. Diese sollen vorwiegend senkrecht zu den Fugen der Module verlaufen. Die Bewährung und der Beton in den Segmenten soll möglichst axial belastet werden, deshalb sollen die Stäbe in Richtung der Hauptspannungen ausgerichtet werden. Damit kann eine materialeffiziente Abführung der Last sichergestellt werden. Diese Bewährungsstäbe können auch unter Vorspannung gesetzt werden, um einen optimalen Verbund zu erreichen. Dafür ist es notwendig, bereits bei der Produktion Löcher für die spätere Bewährungsführung frei zu halten (siehe Abbildung 3).

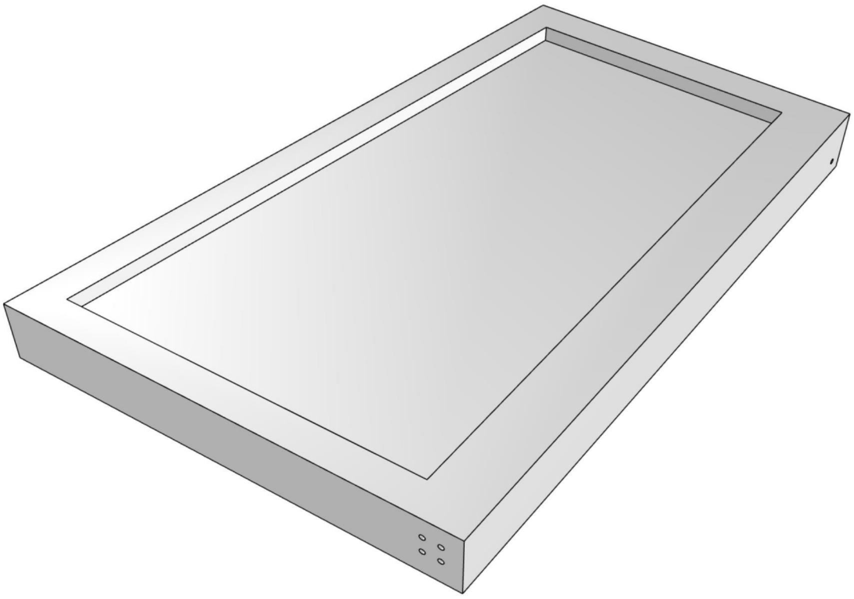


Abbildung 3: Konzept eins Modul mit Aussparung für den Bewährungsstahl

2.2 Generative Design

Generative Design ist ein Lösungsansatz für Entwürfe und Konstruktionsmöglichkeiten. Dabei werden anders als in konventionellen Entwürfen Geometrien nicht händisch erzeugt sondern automatisiert. (vgl. Zwettler 2020). Nach Krish 2011 werden bei diesem Prozess drei Eigenschaften benötigt:

- Ein Designschema,
- Möglichkeit zur Erstellung von Variationen,
- Beurteilung der Entwürfe.

Krish 2011 beschreibt diesen Prozess S.90.

Ausgehend von diesen Voraussetzungen kann ein Ablauf skizziert werden, der diesen Ansatz umsetzt (siehe Abbildung 4).

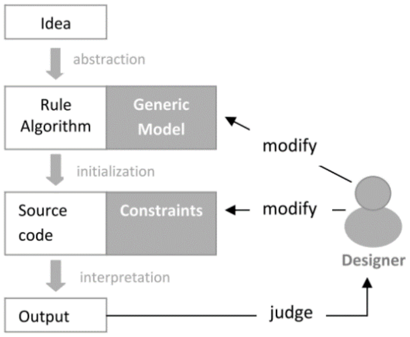


Abbildung 4: Ablauf Generative Design

Quellen: (Krish 2011)

Der Prozess beginnt mit der Idee, welche Konstruktion benötigt wird. Im Kontext dieser Arbeit ist dies der Überbau einer Brücke. Damit werden Rahmenbedingungen für das Model festgelegt. Im nächsten Schritt wird das Model durch einen Algorithmus modifiziert. Dadurch entstehen stochastisch variierbare Lösungen für dieses Problem. Darauf folgend werden die generierten Lösungen bewertet. Hier werden die Eigenschaften beurteilt. Das können Gewicht, Größe, oder Spannungsspitzen sein. Nach der Bewertung dieser Eigenschaften können das Model bzw. die prozedurale Erzeugung Algorithmus angepasst werden, um eine bessere Lösung zu erhalten. Dieser Prozess kann wiederholt werden bis eine nahezu optimale Lösung gefunden wird. Alternativ können auch von dem initialen Model mehrere Lösungen parallel entwickelt und am Ende anhand von Bewertungen eingestuft werden (vgl. Krish 2011 S.90ff). Der am Schluss vorgestellte Ansatz wird in dieser Arbeit verwendet. Durch dieses Vorgehen ist es möglich effizienter Designs zu erstellen, da eine Vielzahl an Entwürfen existiert, die automatisch bewertet werden können. (vgl. Zwettler 2020)

3 Formgrammatik

3.1 Aufbau

Stiny 1980 definiert eine Formgrammatik wie folgt: Eine Formgrammatik besteht aus vier Komponenten:

1. Einer endlichen Menge S von Formen
2. Einer endlichen Menge L von Symbolen
3. Einer endlichen Menge R von Regeln der Form r: α → β
4. Eine beschriftete Form I, die von S und L abhängig ist, und auch als Ursprungsform bezeichnet wird (Stiny 1980, S347)(Übers. des Verfassers).

Eine Formgrammatik wirkt sehr abstrakt, dennoch gibt es sehr viele Parallelen zur gesprochenen und geschriebenen Sprache. Die Formen S und die Symbole L definieren, die unterste Ebene auf die eine Sprache abstrahiert werden kann. Dabei stellen Formen Wörter und Symbole Buchstaben oder auch Satzzeichen dar. Die Regeln R ersetzt Formen oder Symbole (α) durch andere Formen bzw. Symbole (β). Dies ist vergleichbar mit der Groß- und Kleinschreibung oder den Kommaregeln in der deutschen Sprache. Die Ursprungsform ist die Menge der Formen, auf die die Regeln angewendet werden (vgl. Stiny 1980 349ff). Zum Beispiel wird aus dem Satz: „das ist ein testsatz indem ich etwas zeigen möchte.“ durch Anwendung der deutschen Grammatik „Das ist ein Testsatz, indem ich etwas zeigen möchte.“. Es ist ersichtlich, dass durch die Groß- und Kleinschreibung die Symbole „d“ und „t“ durch „D“ und „T“ ersetzt wurden. Dieses Vorgehen kann in drei Schritte unterteilt werden:

1. Deklarativ beschriebene Prozeduren,
2. Mustererkennung,
3. Mustersetzung.

Dieses Prinzip der Chomsky-Grammatiken ist auch bei den Formgrammatiken wiederzufinden. Folgend wird das Verfahren anhand eines Beispiels erläutert. Es wird vorgegeben, dass alle gleichschenkligen Dreiecke durch ein Rechteck ersetzt werden sollen. Dabei wird nur die Art der Geometrie / Form verändert. Die ursprüngliche Größe der Geometrie ist identisch. Dies ist gut an den Hilfslinien im Kreis zu sehen (vgl. Abbildung 5).

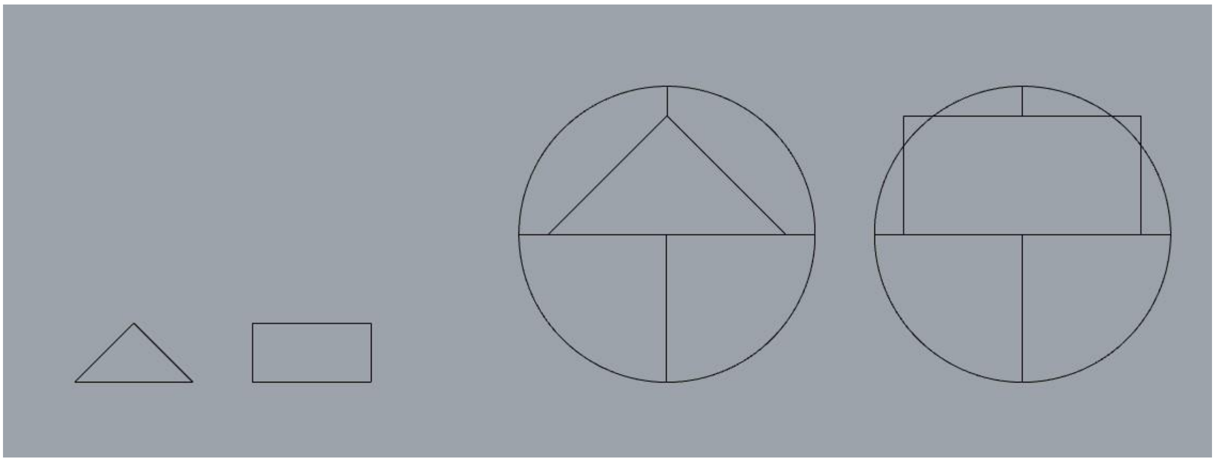


Abbildung 5: Anwendung Formgrammatik. Links wird die Regel angezeigt. Rechts die Anwendung auf die Ursprungsform. Zugunsten der Übersichtlichkeit wurde die Geometrien seitlich verschoben.

3.2 Funktionsweise

Um Formgrammatiken bzw. die Regeln dafür anzuwenden, ist es notwendig die Ursprungsgeometrie zu abstrahieren d.h. in Subformen zu unterteilen. So können auf diese Subformen einzeln die Regeln angewendet werden. Grasl und Economou 2013 geht davon aus, dass für normale Regeln eine Geometrieerkennung mittels einer Bilderkennung oder auch die Ermittlung der durch Schnittpunkte von maximalen Linien möglich ist. Der letztere Ansatz birgt ein Problem, das in Abbildung 5 deutlich wird. Die rechts dargestellten Formen sind damit nicht erkennbar, da die unterste Linie des Dreiecks bzw. Vierecks nicht komplett mit der durchgehenden Linie übereinstimmt. Um dieses Problem zu lösen, wandelt man die Formen in einen Graphen um. Aus Implementierungssicht sind Graphen als Datenstruktur effizienter umzusetzen. Damit können optimierte Mustererkennungsalgorithmen und topologisch-assoziative Mustererkennung für die Implementierung verwendet werden. Auf diese Weise lassen sich Matches für die Anwendung der Regeln finden. Matches sind dabei passende Formen S bzw. Symbol D in der Menge I. Die Mustererkennung ist auch durch andere Strukturen möglich z.B. durch Sortal Structures.

Diese Regeln sind dadurch Charakterisiert, dass ausschließlich durch Rotation und Skalierung die Form in einer Geometrie gefunden wird. Somit muss die exakte Form in der Ursprungsgeometrie gefunden werden.

Um Formen auf einer topologischen Ebene zu erkennen und zu modifizieren, wird eine parametrische Regel benötigt. „Parametrische Formgrammatiken sind eine Erweiterung von Formgrammatiken, in diesen Regeln werden durch das Ausfüllen von offenen

Termen, ein allgemeines Schema definiert.“ (Stiny 1980 S.349) (Übers. des Verfassers). Um parametrische Regeln nutzen zu können, muss der graphenbasierte Ansatz verwendet werden, da es Computerprogramme nicht möglich ist Formen anhand deren Topologie zu erkennen. Hier ist der Aufbau des Graphen bzw. auf die Fromselektierung zu beachten. Durch das Erzeugen eines Graphen werden alle Schnittpunkte von Linien mit Einträgen in einer Adjazenzmatrix modelliert. Ebenso werde die Verbindungskanten in diesem Model als Einträge abgebildet. Damit wird ein Viereck topologisch zu einem Polygon mit sechs Eckpunkten (vgl. Abbildung 6). Somit kann eine Selektierung von Formen nicht durch diese Adjazenzmatrix realisiert werden, da ein Viereck nicht zwingend durch einen geschlossen Graphen mit vier Knoten dargestellt wird. Deshalb wird der Graph mit Teilbeziehungen (Knoten und Kanten) beschrieben. In dieser Adjazenzmatrix existieren auch Kanten, die durch Schnittpunkte getrennt werden. Somit kann ein Algorithmus definiert werden, der abwechselnd nach einem Knoten und einer anschließenden Kante sucht und maximal vier Knoten hat. Infolgedessen kann in dem Graphen jedes Viereck, ausgehend von einer einzelnen Vorschrift, gefunden werden. Dieses Vorgehen funktioniert analog mit anderen Polygonen.

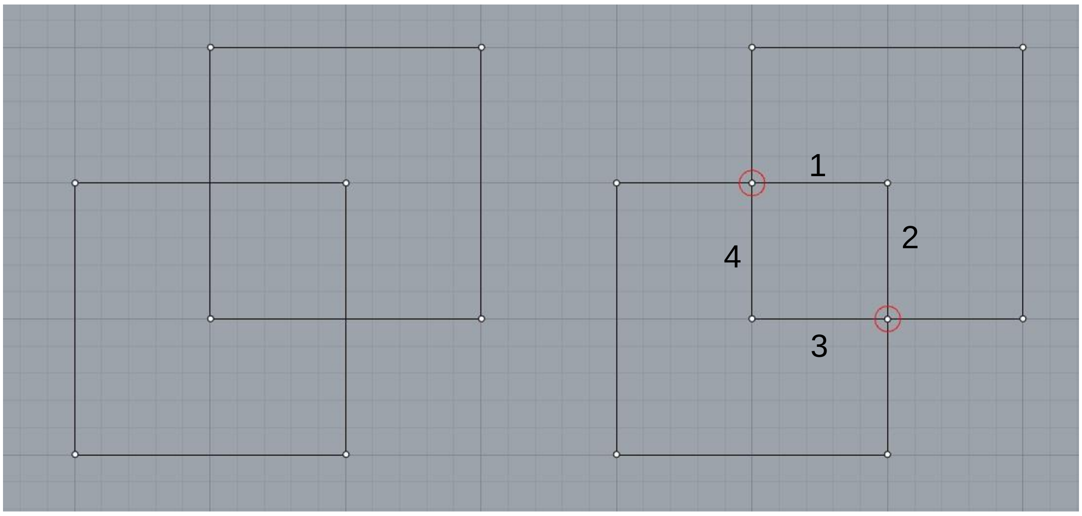


Abbildung 6: Beispiel Form, Links als Form, Rechts als Graph mit Kennzeichnung der zusätzlichen Knoten

Es kann dadurch von α in der Zielgeometrie aus nach Matches gesucht werden. Die Mehrheit der potentiellen Anwendungen ist hierbei redundant. Denn durch die Suche werden alle möglichen Permutationen der Kanten gefunden. Zum Beispiel bilden die Kanten 1234 das identische Viereck wie die Kanten 2341 (vgl. Abbildung 6). Insgesamt existieren 8 Möglichkeiten dieses Viereck zu beschreiben. Daher ist bei der Regelerstellung darauf zu achten geeignete Restriktionen zu wählen, da sonst unerwünschte

Matches gefunden werden. Dieses Verhalten wird besonders dann sichtbar, wenn Linien hinzugefügt werden, die nicht in einer Symmetrieebene des Polygons liegen. Mögliche Restriktionen sind unter anderem Parallelität und Orthogonalität der Kanten, Labels, Farben oder auch Gewichtung. (vgl. Grasl und Economou 2013).

Die bereits erwähnten Restriktionen werden auch *Predicates* genannt. *Predicates* sind Einschränkungen für α, die die Auswahl, der zu verändernden Formen selektieren. Dies ist notwendig, da es durch die Permutation der Kanten eine Vielzahl an Matches gibt. Diese Restriktionen können zusätzlich zu α angegeben werden, aber diese können auch automatisch erzeugt werden. Ein Beispiel dafür ist, dass bei einem Quadrat alle Seiten die identische Länge, sowie einen 90° Winkel an den Eckpunkten haben müssen. (vgl. Stouffs 2019)

Im Gegensatz zu den *Predicates* beinhalten *Directives* zusätzliche Informationen für β. Sie sind nur bei der Anwendung von parametrischen Regeln zwingend notwendig. *Directives* sind nötigt für den Fall, dass eine Struktur topologisch nicht eindeutig definiert ist. Dies ist zum Beispiel für eine Linie der Fall, die an eine bereits existierende Linie von α anknüpfen soll. Dies ist nicht eindeutig definiert, da keine Richtung bzw. Länge in der Graphenrepräsentation beschrieben wird. Um dies eindeutig definieren zu können muss der Winkel und die Länge der neuen Linie angegeben werden. (vgl. Stouffs 2019)

3.3 Related Work

Das Vorgehen mittels einer Formgrammatik Geometrien zu unterteilen ist bereits eine bekannte Strategie. Allerdings beschränkt sich die aktuelle Forschung häufig die Unterteilung und die Zusammenfassung von einfacheren Formen (vgl. Benrós et al. 2012). Beispielsweise werden ausgehend von einem rechteckigen Grundriss mögliche Raumaufteilungen erstellt. Zudem weisen die Grundrissprobleme in der Regel einen höheren Grad an Diskretisierung auf. Bei Benrós et al. existiert eine überschaubare, finite Anzahl an möglichen Kombinationen für das Problem, indem Grundrissaufteilungen auf ein grobes Raster reduziert werden. In dieser Arbeit wird die kontinuierliche Unterteilung mithilfe von Zufallszahlen vorgenommen. Dadurch sind praktisch unendlich viele mögliche Kombinationen denkbar. Schließlich erhöht die Verwendung von parametrischen Kurven als Berandungselement die Komplexität der Arbeit.

4 Anwendung in SortalGI

Der Sortal Grammar Interpreter (SGI) ist ein Werkzeug zur Erstellung und Prozessierung von Formgrammatiken. SGI bietet über verschiedene Schnittstellen Zugang zu seinen Funktionalitäten: Über Python, Rhino sowie Grasshopper. Sortal wurde für diese Arbeit aufgrund der folgenden Eigenschaften verwendet. Zum einen ist die einfache Bedienung durch die visuelle Darstellung in Grasshopper zu nennen. Sowie die Option, Formen mit Hilfe von Rhino, schnell zu erstellen. Außerdem ist eine schnelle Einarbeitung, durch die zahlreiche Tutorials möglich.

4.1 Sorts

In Sortal werden die beschreibenden Elemente *Sorts*, *Individuals* und *Forms* genannt. Sorts sind die Elemente der untersten Ebene und bilden somit die Menge S und L (vgl. Abschnitt 3.1) in Sortal. *Sorts* sind allgemeinen Datentypen aber auch die Beschreibung von Klassenstrukturen in der Geometrie. *Individuales* sind die Objekte die ausgehend von den Mengen erstellt werden. Ein *Form* ist eine Liste/Array von *Sorts*. Weiter können *Sorts* in vier Kategorien eingeordnet werden:

- *Primitive Sort* ist ein primitiver Datentyp, z. B. Punkte oder Linien
- *Attribute Sort* ist eine Verbindung zwischen einem Primitiven Sort und einem beliebigen anderen Sort. Der Primitive Sort ist dabei untergeordnet.
- *Disjunctive Sort* ist eine Menge an Sorts, die keinen visuellen Bezug zueinander haben
- *Compound Sort* ist eine Menge an Sorts, die sich auf denselben Bereich beziehen. Z. B. ein Zusammenschluss aus Punkten und Linien die ein Polygon bilden.

Sorts sind ein ausgereiftes mengentheoretisches Konzept, das die effiziente Komposition und Verarbeitung komplexen Formen ermöglicht. Aufgrund der Bedeutung dieser Datenstrukturen Abbildung, Musterkennung und regelanwendung wurde der Interpreter entsprechend benannt Aufgrund der zentralen Rolle, die *Sorts* einnehmen, wurde diese Bibliothek Sortal genannt (vgl. Sortal.org 2017). Im Folgenden wird auf die Anwendung von Sortal eingegangen, genauere Informationen zu Sortal sind in Stouffs und Krishnamurti (2001)zu finden.

4.2 Syntax

Um eine Formgrammatik in Sortal anwenden zu können, muss mindestens eine Regel erstellt werden. Zur Verdeutlichung wird eine Beispiel Regel erstellt (vgl. Abbildung 7).

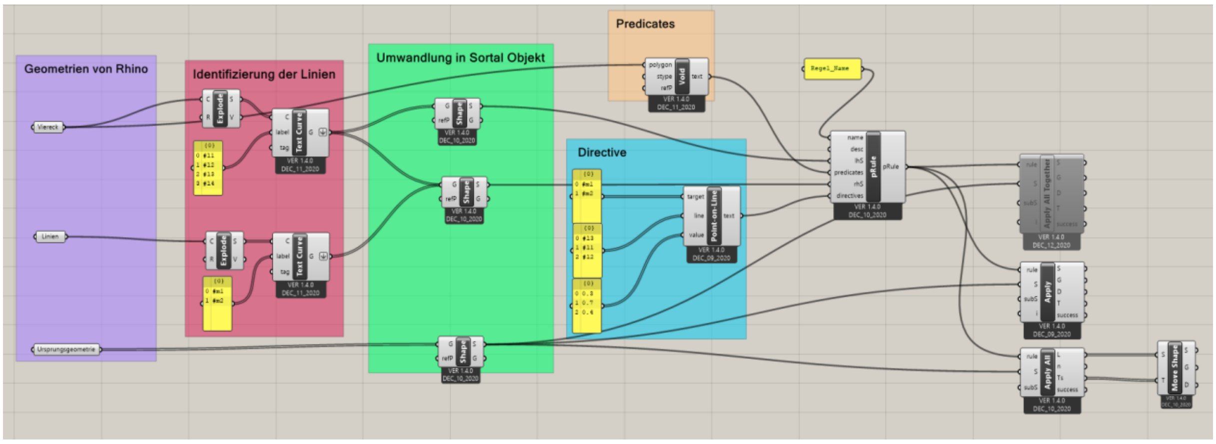


Abbildung 7: Beispielregel in Sortal

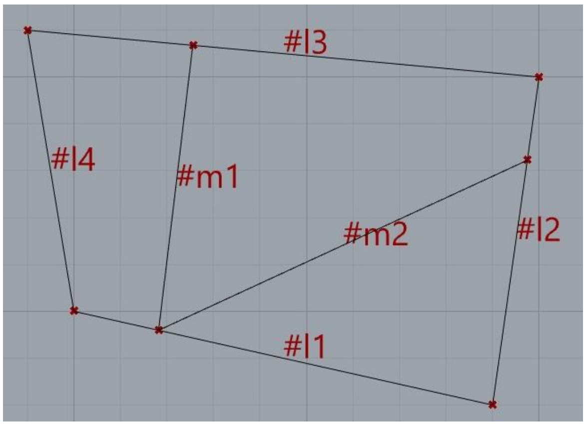


Abbildung 8: LHS und RHS der Beispielregel

Für eine Regel werden eine LHS und eine RHS benötigt. Dabei steht die LHS für α und die RHS für β von Abschnitt 3.1. In Abbildung 8 sind beide abgebildet. #l1-#l4 bilden dabei die LHS und #m1 und #m2 die RHS. Die LHS ist, die zu modifizierende Geometrie, d.h. sie ist der Startpunkt und markiert die Geometrie. Dabei sind auch offene Geometrien möglich. Es ist nicht nur die Topologie der Geometrie wichtig. Zum Beispiel werden Regeln die anhand eines Quadrates erstellt werden, auch nur auf Quadrate angewendet. Das bedeutet, dass nur Vierecke mit gleichen Seitenlängen und rechten Winkel als Match erkannt werden. Wenn hingegen die Regel mit einem allgemeinen Viereck erstellt wird, gibt es keine Einschränkung bezüglich Seitenlängen und Winkeln. Das bedeutet wenn #l1-#l4 in Abbildung 8 als ein Rechteck definiert wird, werden für Abbildung 9 kein Match gefunden.

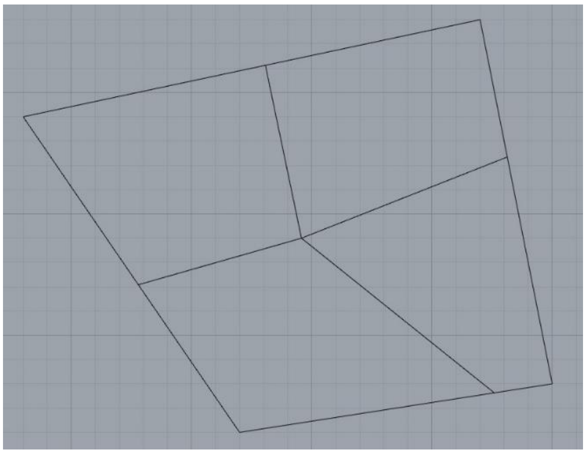


Abbildung 9: Ursprunggeometrie für die Beispielregel

Die RHS legt die Zielgeometrie fest, also die Form, die nach der Regelanwendung entsteht. Hier kann entschieden werden, ob eine Geometrie hinzugefügt, gelöscht oder modifiziert wird. Wichtig ist, dass Sortal die markierte Geometrie durch die RHS ersetzt. Somit muss für das Hinzufügen einer Geometrie auch die Geometrie der LHS, ebenso wie der RHS mit übergeben werden, da sonst nur die hinzugefügte Geometrie erstellt dargestellt wird. Dies ist in Abbildung 7 zu sehen dabei wird die Geometrie des Vierecks sowohl mit der LHS als auch der RHS verbunden.

Weiter ist es für komplexere Anwendungsfälle notwendig *Predicates* und *Directives* zu vergeben, diese sind in Abschnitt 3.2 genauer spezifiziert. Die implementieren *Predicates* bieten die Möglichkeit z.B. nur Polygone zu selektieren, deren Inneres leer sein muss. Ebenso können Ränder definiert werden die als Abgrenzung dienen und diesen Bereich ausnehmen bzw. nur den selektierten Bereich benutzen. In Abbildung 7 ist als Beispiel ein Void Predicate angegeben. Somit kann der Hacken nicht in das gesamte Viereck eingefügt werden.

Die implementieren *Directives* bieten unter anderem die Möglichkeiten die Länge einer Linie, den Winkel zwischen zwei Linien aber auch die Position auf einer Linie festzulegen. Für das Beispiel des Viereckes in Abbildung 9, kann man zwei Endpunkte der Linie (#m) auf eine bestimmte Position der Kanten (#l) festlegen. In Abbildung 7 in der Gruppe *Directive* ist ein Beispiel für die Implementierung. Anhand der Point on Line *Directive* werden #m1 und #m2 als „Target“ definiert. **Line** sind dabei die Linien, auf den die Punkte liege sollen. Mit den **Values** wird die Laufkoordinate über die Linie angegeben. Zudem besteht die Möglichkeit **Values** als zufällige Werte in einem angegebenen Bereich bei jeder Regelanwendung neu zu

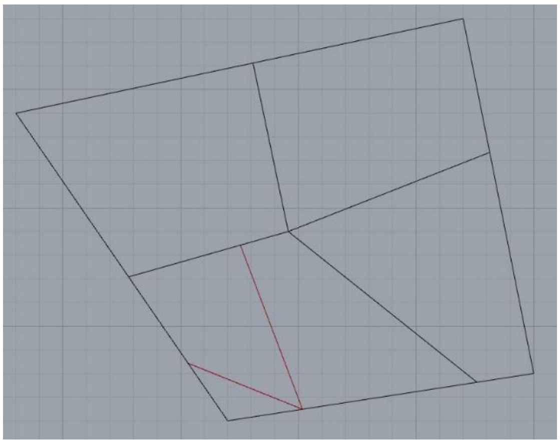


Abbildung 10: Zufällige Regelanwendung auf die Ursprunggeometrie für die Beispielregel

generieren. Im Weiteren wird dies für die Segmentierung der einzelnen Bauteile verwendet.

Mit den bereits genannten Komponenten lassen sich Regeln für die Formgrammatik definieren. Dieses wird in Sortal mit einen Block realisiert, der eine Regel erstellt (Rule oder pRule, falls eine parametrische Regel benötigt wird). Für die Anwendung der Regel auf eine Geometrie muss ein Apply Block benutzt werden. Dieser Block benötigt mindestens eine Regel und eine Ursprungsgeometrie, die verändert werden soll. Es gibt mehrere Varianten des Apply Blocks. Diese sind notwendig, da häufig mehrere Matches existieren, auf die die Regel angewendet werden kann. Deshalb ist es möglich einen Index bei den normalen Apply Block zusätzlich mit anzugeben. Dieser gibt an, welche der vorhandenen Matches verändert werden sollen. Falls dieser Wert nicht gesetzt ist, wird ein zufälliger ausgewählt. Es besteht die Möglichkeit alle selektierten Geometrien mit dem Apply-All Block zu verändern. Dabei ist die rekursive Anwendung der Regel ausgeschlossen, d.h. es werden vor der Transformation die zu ändernden Geometrien markiert und anschließen ersetzt. Zusätzliche gibt es auch ein **Success-Flag** mit dem man überprüfen kann, ob die Regel angewendet wurde oder keine passende LHS in der Ursprungsgeometrie gefunden wurde. Ebenfalls wird bei den Apply-all Block eine „Debug“ Hilfe mitausgegeben. Dabei handelt es sich um einen Vektor, der eine mögliche Verschiebung beinhaltet. Damit ist es möglich alle Anwendungen der Regel simultan zu sehen und den benötigten Index für einen einzelnen Apply-Block herauszufinden (vgl. Abbildung 11). Die Anwendung folgt einer Reihenfolge, die durch die Zeichenreihenfolge festgelegt wird. Damit können sich trotz identischer Ursprungsgeometrien unterschiedliche Reihenfolgen der Anwendungen ergeben. (vgl. Stouffs 2019)

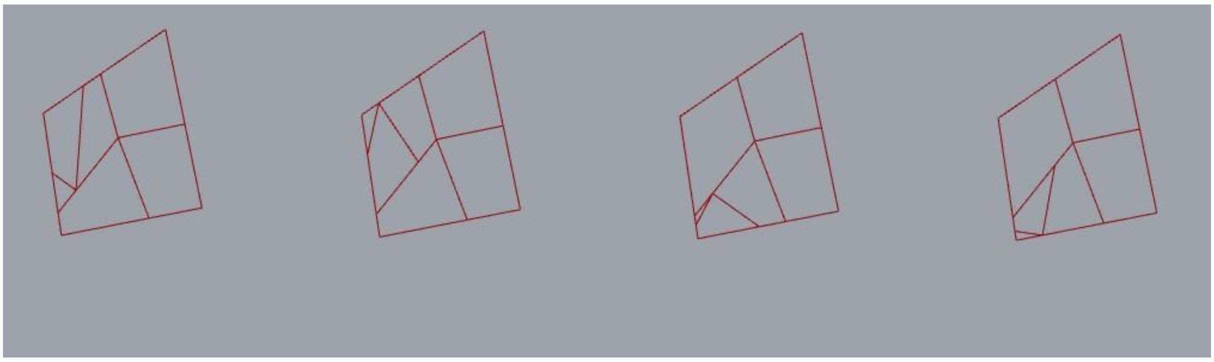


Abbildung 11: Ausschnitt aus den Apply All der Beispielregel

Es besteht ebenfalls die Möglichkeit alle möglichen Matches gleichzeitig anzuwenden, mittels des Apply-All-Together Blocks. Dies kann zu chaotischen Formen führen wie in Abbildung 12 zu sehen ist.

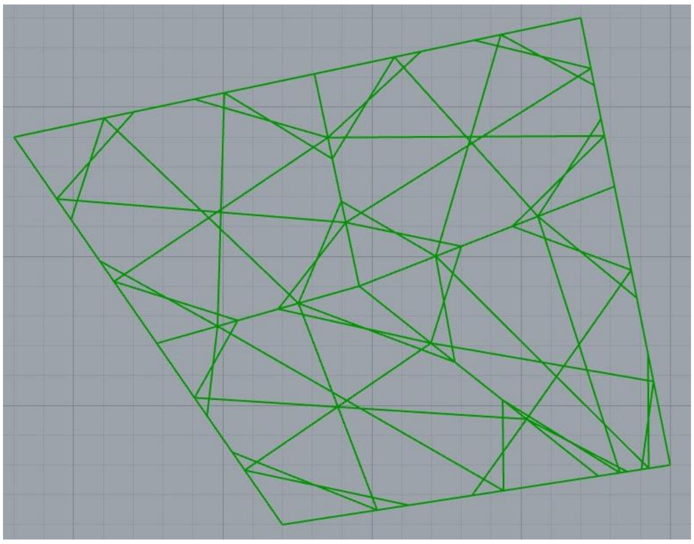


Abbildung 12: Anwendung des Apply All Together Blocks auf Ursprungsgeometrie

4.3 Einschränkungen in Sortal

Sortal besitzt einige Einschränkungen bzw. Eigenschaften, die bei der Implementierungen berücksichtigt werden müssen:

1. Bezeichnung von Linien: Um *Directives* und *Predicates* zu nutzen, muss jede Linie in der LHS eindeutig benannt werden. Dies ist dynamisch nicht möglich.
2. Fehlende *Directives*: Die zurzeit verfügbaren *Directives* beinhalten fast ausschließlich absolute Manipulationen, d.h. Linien können nur mit einer bestimmten Länge und zwischen zwei Linien kann nur ein bestimmter Winkel angegeben werden. Eine Ausnahme dazu ist das „Point on Line“ *Directive*. Dies funktioniert nur mit Linien aus einem einzigen Segment.
3. Fehlende *Predicates*: Die zurzeit verfügbaren *Predicates* können Linien ausschließlich relativ zu anderen Linien in Bezug setzen. Damit ist es möglich beispielsweise die längste bzw. kürzeste Linie zu definieren. Es ist nicht mögliche eine absolute Länge der längsten bzw. der kürzesten Linie anzugeben.

5 Codebeschreibung

Im Folgenden wird die Implementierung, der Formgrammatik in Grasshopper mithilfe von Sortal genauer beschrieben. Da Sortal zurzeit (V 1.3.0) keine *Directive* bietet um mit Bezierkurven zu interagieren, wird die initiale vertikale Segmentierung nicht durch eine Formgrammatik erstellt. Die Implementierung passt die Segmentierung dynamisch an die gegebenen Kurven an. Dies funktioniert nur mit Linien und quadratischen Bezierkurven. Der Code wurde modular aufgebaut, damit es möglich ist die initiale Segmentierung zu ersetzen und die im Folgenden beschriebenen Regeln weiter zu verwenden. Alle Anwendungen der Regeln werden mit „Apply All Together“ realisiert, da Sortel ausgehend von der Konstruktionsreihenfolge die Indexbelegungen des Applyblocks festlegt. Dies soll aber keinen Einfluss auf die Ausführung des Algorithmus haben, weshalb alle Regeln immer komplett angewandt werden. Im Verlauf dieses Kapitels wird dies noch genauer beschrieben.

Ausgehend von der Geometire in Abbildung 13 werden die erste Segmentierung und die verwendeten Regeln zur Diskretisierung und zur Segmentierung verdeutlicht. Zur besseren Übersicht werden nur einzelne Ausschnitte als Screenshot beigefügt.

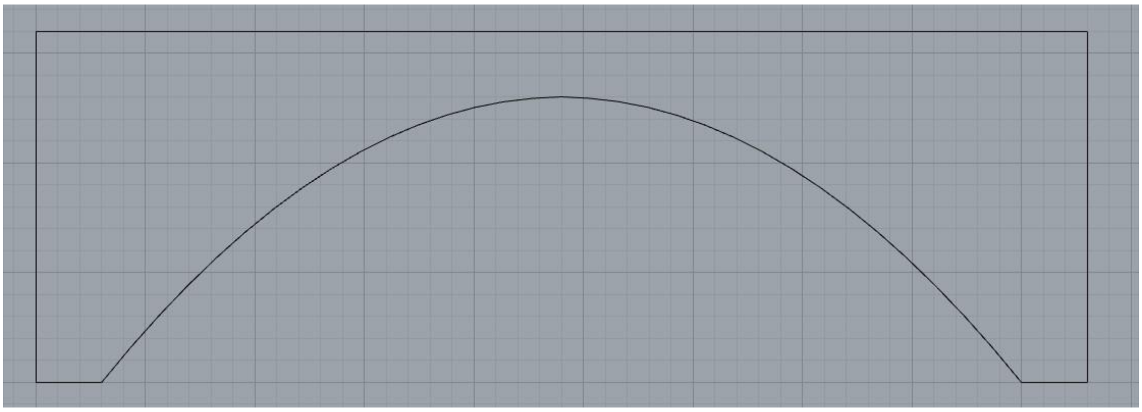


Abbildung 13: Grundform zur Segmentierung

5.1 Erste Segmentierung

Folgend ist eine mögliche Regel (vgl. Abbildung 14) sowie LHS und RHS (vgl. Abbildung 15) aufgeführt zur vertikalen Segmentierung mittels einer Formgrammatik.

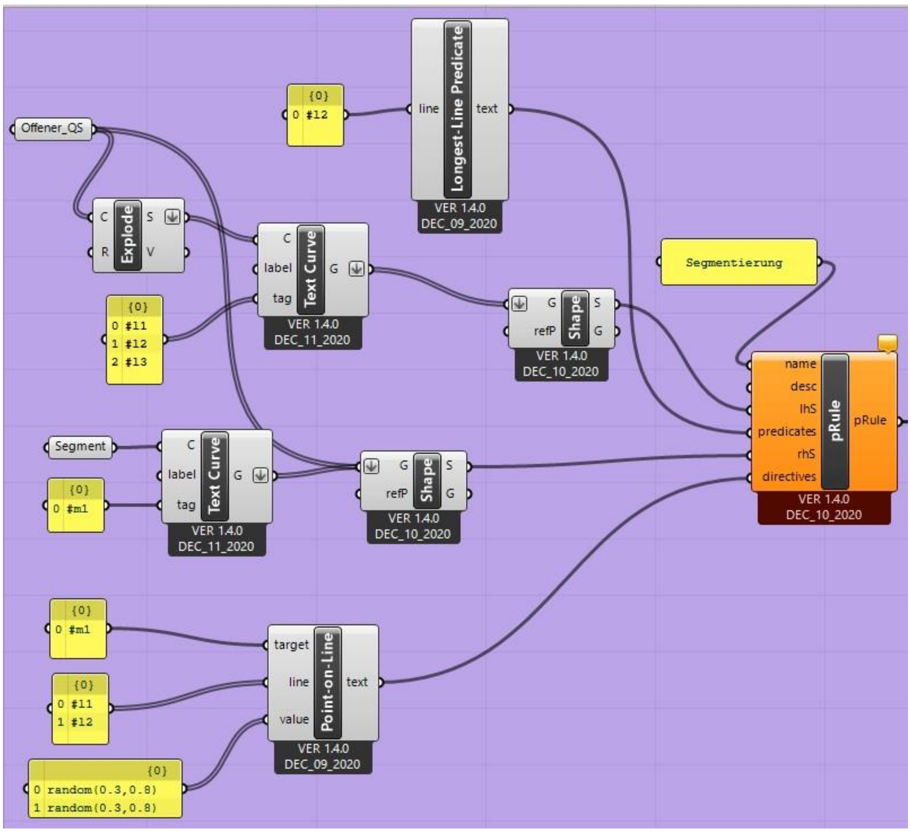


Abbildung 14: Codebausteine für eine erste Segmentierung in Sortal

Die RHS fügt #m1 hinzu. Dabei müssen sich die Endpunkte jeweils auf #l1 und #l2 befinden. Die *Directive* „Point-on-Line“ müssen mit einer passenden z.B. „Point-on-Curve“ ersetzt werden. Dieser Aufruf geht von einer Implementierung analog zu „Point-on-Line“ aus. Diese *Directive* wird in Regel 2 beschrieben. Das *Predicate* „Longest-Line“ wird in Regel 1 beschrieben.

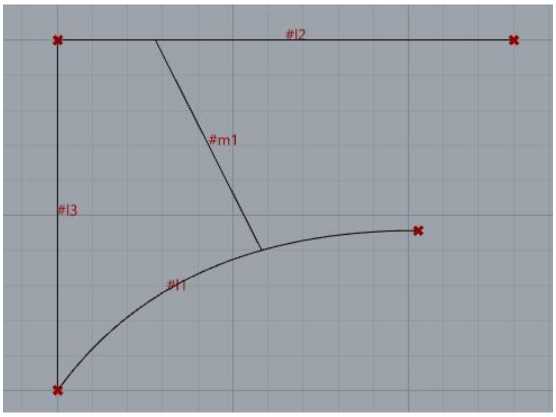


Abbildung 15: LHS und RHS für eine erste Segmentierung

Da, wie bereits beschrieben, Sortal diese native Möglichkeit nicht bietet, wird auf Funktionen von Grasshopper zurückgegriffen. Die Segmentierung erfolgt durch Unterteilung der Längen der oberen und unteren Begrenzung in beliebig viele Abschnitte. Zusätzlich wird die untere Aufteilung modifiziert, da diese mit ihrer Bogenform länger ist als die gerade Linie.

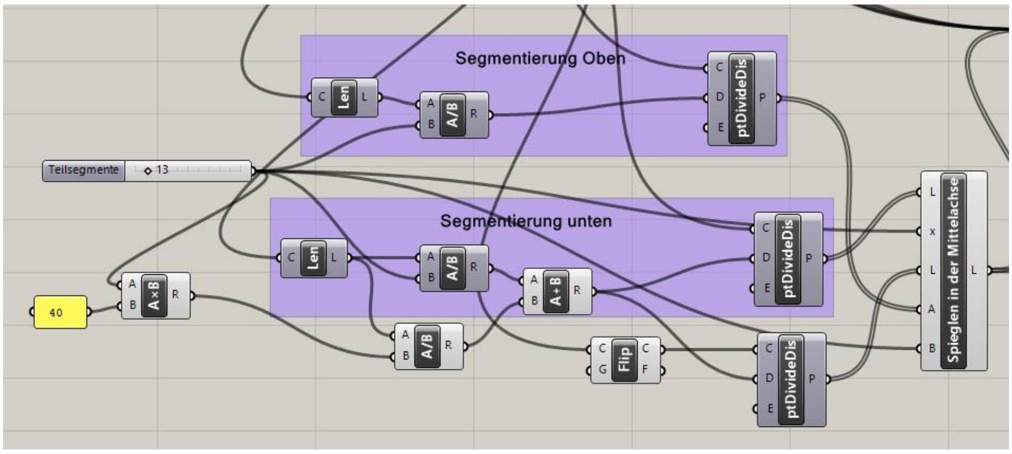


Abbildung 17: Codebausteine für die vertikale Segmentierung

Implementiert ist dies mittels eines Faktors, der abhängig von der Gesamtlänge der Linie und der Anzahl der Unterteilungen ist. Um trotz dieser Modifikation einen Schlussstein zu erhalten, der Keilförmig ist, wird die Segmentierung nur bis zur Mitte angewandt. Die andere Seite wird dafür gespiegelt. Der bereits genannte Faktor errechnet sich, in dem die Gesamtlänge der Kurve in 40*(#Segmente) geteilt wird und anschließend zu der initialen Segmentierung dazu gezählt wird. (vgl. Abbildung 17). Damit wird die Geometrie entlang der Linien von Segment oben zu Segment unten erweitert. (vgl. Abbildung 16)

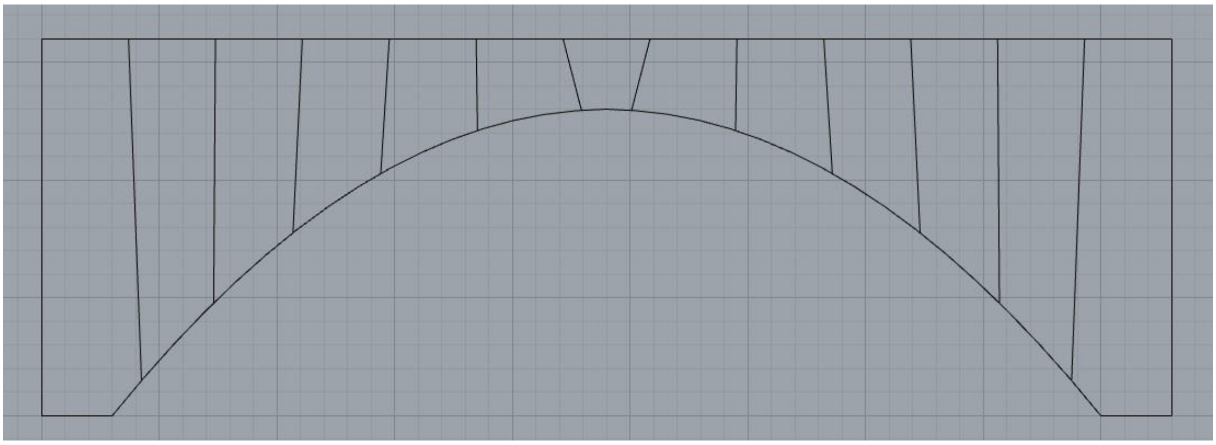


Abbildung 16: Vertikale Segmentierung der Bauteilgeometrie

5.2 Verwendete Regeln

Ausgehend von der in Abbildung 16 wurden drei parametrische Regeln entwickelt. Die erste Regel dient zur Diskretisierung des Bogens. Hierbei wird eine parametrische Regel (pRule) angewendet (siehe Abbildung 19). Die LHS besteht dabei aus den Linien #l1, #l3, #l4 und der Bezierkurve. Die RHS fügt die Linie #l2 hinzu um ein Viereck zu erzeugen (siehe Abbildung 18). Um die Linien zu benennen muss die Geometrie in einzelne Liniensegmente aufgeteilt werden. Dies wird mittels des Blockes Explode zur

Aufteilung in Linien und Text Curve zur Identifikation der Linien realisiert (vgl. Abbildung 19).

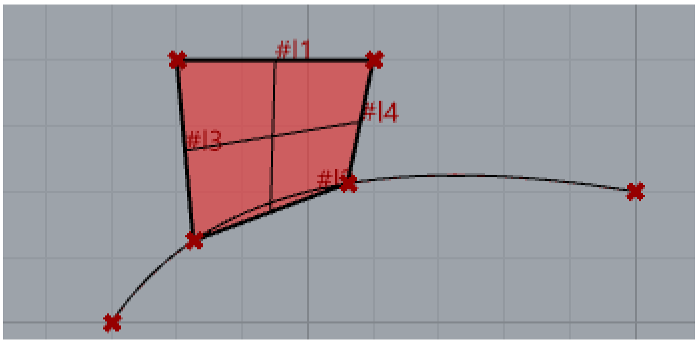


Abbildung 18: LHS und RHS der Regel1

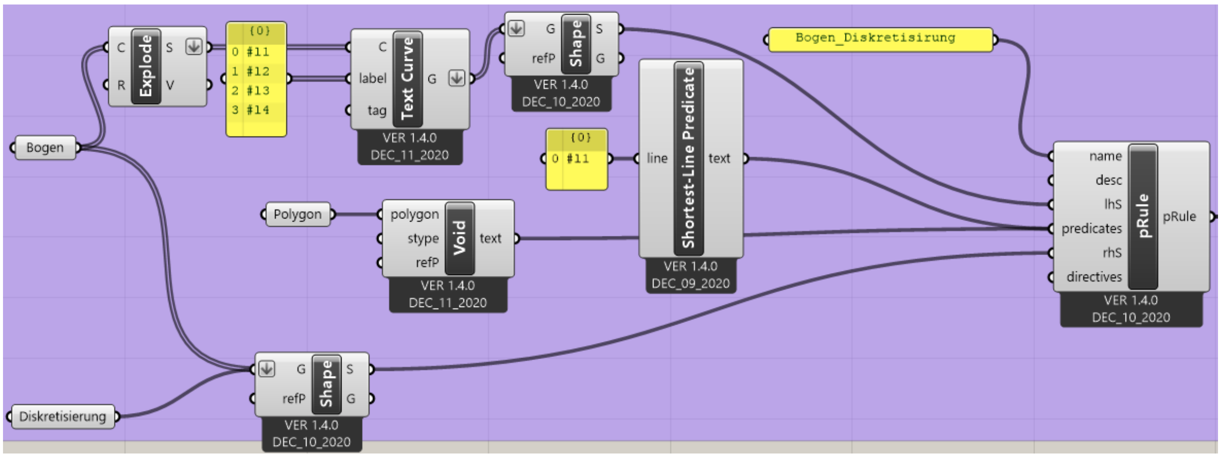


Abbildung 19: Codebausteine von Regel 1

Zusätzlich gibt es noch zwei *Predicates* um die Matches einzuschränken. Eines davon ist das „Shortest-Line Predicate“. Damit wird sichergestellt, dass ein Segment ausreichend groß ist, um bei nachfolgenden Regelanwendungen sinnvoll unterteilt werden zu können. Ebenfalls wird dadurch eine mögliche Anwendung auf die linke und rechte außen Seite verhindert. Das zweite *Predicate* ist das „void-Predicate“, wonach innerhalb des angegebenen Polygons keine Linien oder Kreise dürfen sein. In Abbildung 18 ist dieses Polygon in Rot dargestellt. Damit werden nur noch vertikal benachbarte Linien zu einem Viereck verbunden. In Abbildung 20 ist zu erkennen, dass die Segmente im Zentrum nicht diskretisiert wurden. Dies ist die Auswirkung des „Shortest-Line Predicate“ und wird dadurch nicht in späteren Regelanwendungen berücksichtigt.

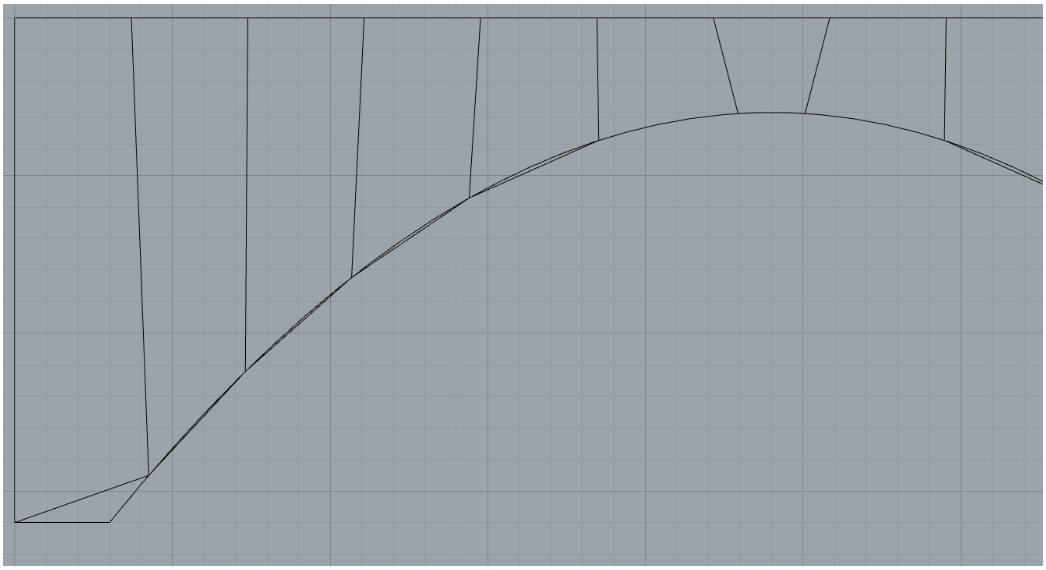


Abbildung 20: Linke Seite der Geometrie nach der Diskretisierung. Die rechte Seite ist symmetrisch zur linken.

Dieses Verhalten ist gewollt, denn bei einer Adaptierung z.B. in einem Überbau ist es von Vorteil den Schlussstein als einen kompletten Block einzubauen. Somit können die Spannungen in diesem kritischen Querschnitt besser auf die benachbarten Blöcke verteilt werden. Es entsteht allerdings auch ein Fehler an den Auflagerflächen, da diese komplett abgeschrägt werden. Der Fehler wird mit einer feineren Diskretisierung vernachlässigbar, worauf im Folgenden, auf Grund des zeitlichen Aufwandes nicht näher eingegangen wird.

Die zweite Regel ist für die weitere Unterteilung der Segmente verantwortlich. Die Grundidee dabei ist alle vorhandenen Vierecke in zwei Vierecke zu unterteilen. Das Viereck wird mit einer Linie unterteilt die zwischen den gegenüberliegenden Kanten eingefügt wird (siehe Abbildung 21). Die Endpunkte auf den Linien sind dabei zufällig jeweils im Bereich von [0,2; 0,5] und [0,5; 0,8]. In Sortal wird dies mit dem Directive Point-on-Line gelöst (vgl. Abbildung 22). Da eine parametrische Regel nur auf die Topologie der LHS und RHS angewendet werden kann, müssen für hinzugefügte Linien deren Endpunkte nicht an schon bekannten Endpunkten sind, eine zusätzliche Information bereitgestellt werden. Dies

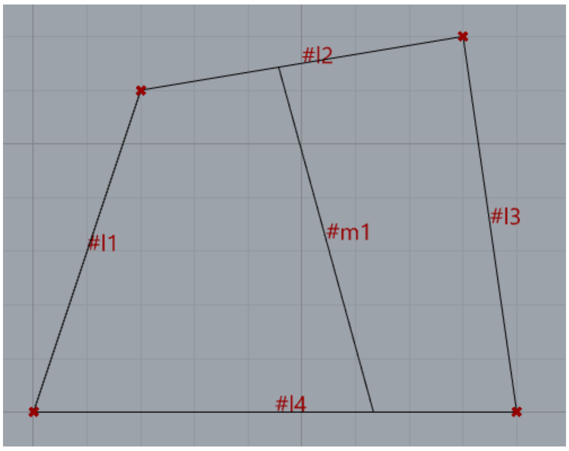


Abbildung 21: LHS der zweiten Regel. Unterteilung der bereits vorhandenen Segmente.

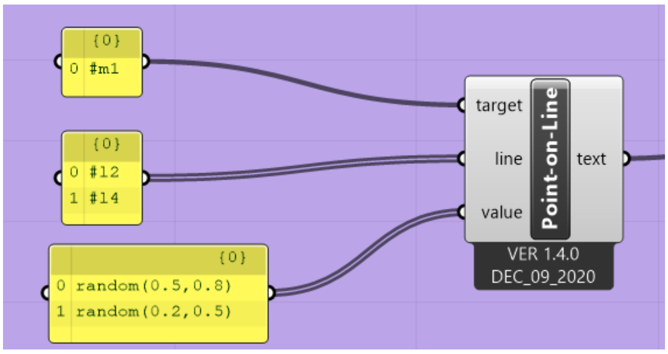


Abbildung 22: Point-on-Line Directive

kann in Form von Winkeln gegenüber den vorhandenen Kanten oder auch in Abhängigkeit der Länge auf einer Kante sein. Möglich ist es auch die absolute Länge der neuen Linie anzugeben. Das verwendete *Predicate* ist wie in der vorhergehenden Regel das Void Predicate. Dies verhindert wie bereits in Regel 1, dass eine Linie über mehrere Segmente erstellt wird. Die zweite Zusatzinformation ist das „Longest-Line“ Predicate. Auf diese Weise wird, analog zu der längsten Linie, die kürzeste Linie in der Form festgelegt. Damit wird sichergestellt, dass die hinzugefügte Linie nur die längste Seite unterteilt. Dadurch sollen die entstehenden Vierecke eine möglichst ähnliche Geometrie erhalten. Durch diese Auswahl an *Predicates* ist die Anwendung nicht eindeutig definiert und durch „Apply all together“ kreuzen sich die hinzugefügten Linien (vgl. Abbildung 23). Der Grund dafür wird in Abschnitt 3.2 beschrieben (Permutation der Kanten). Konkret in dieser Anwendung werden die Vierecke durch zwei zulässige Permutationen erkannt. Dabei sind die Kanten einmal im und einmal gegen den Uhrzeigersinn angeordnet.

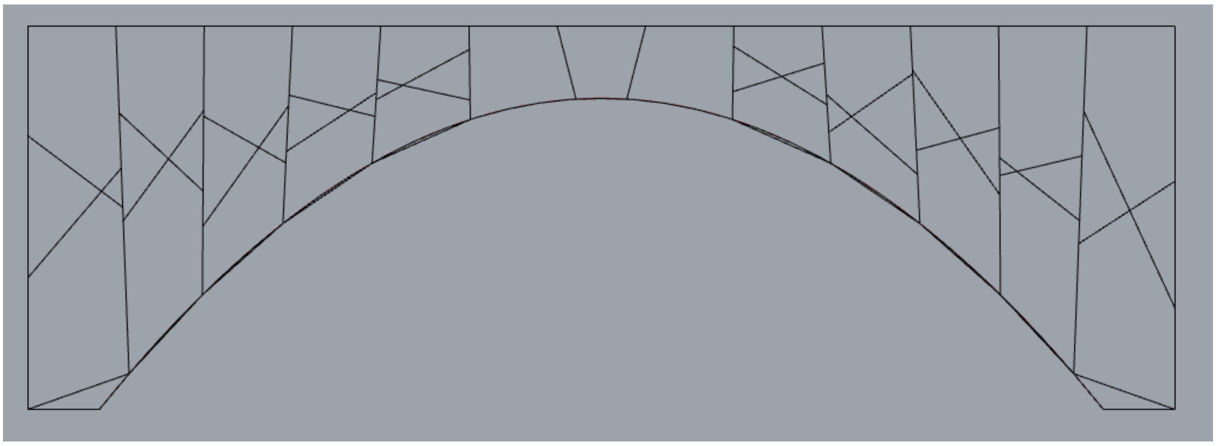


Abbildung 23: Segmentierung nach Anwendung der Regel 2.

Da die Anwendung der Regel abhängig von der Zeichenreihenfolge ist, ist es nicht möglich, dies mittels konsekutiver Anwendung von „Apply“ zu erzeugen. Denn bereits die dritte neu hinzugefügte Linie könnte eine Überschneidung erzeugen.

Aus diesem Grund wird eine dritte Regel (Abbildung 24) benötigt, die diese Überschneidung entfernt. Die LHS sind zwei Linien die sie kreuzen. Die RHS ist eine der beiden Linien und in Abbildung 24 in grün markiert. Bei dieser simplen Regel werden weder Predicates noch Directives benötigt. Damit lässt sich die gewollte Segmentierung erzeugen. (vgl. Abbildung 25)

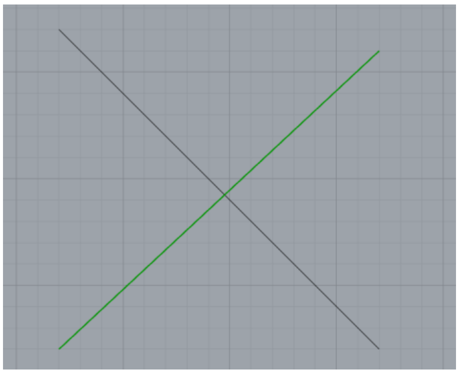


Abbildung 24: LHS und RHS von Regel 3

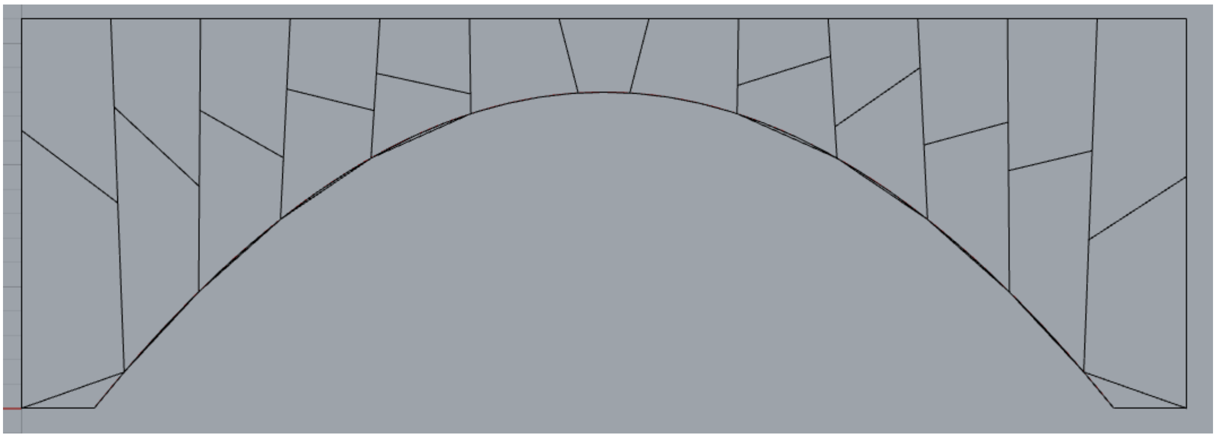


Abbildung 25: Segmentierung nach der ersten Anwendung der Regel 3.

5.3 Workflow

Durch die konsekutive Anwendung der Regel 2 und 3 lässt sich der Grad der Segmentierung einstellen. Eine mögliche Segmentierung durch zwei Iterationen von Regeln 2 und 3 ist in Abbildung 26 zu sehen. Dabei ist es nur mit erhöhten Aufwand mögliche eine minimale Größe der Segmente zu definieren, da in Sortal keine nativen *Predicates* existieren, die absolute Werte z.B. für die Länge einer Linie oder den Winkel zweier Linien als Inputparameter haben.

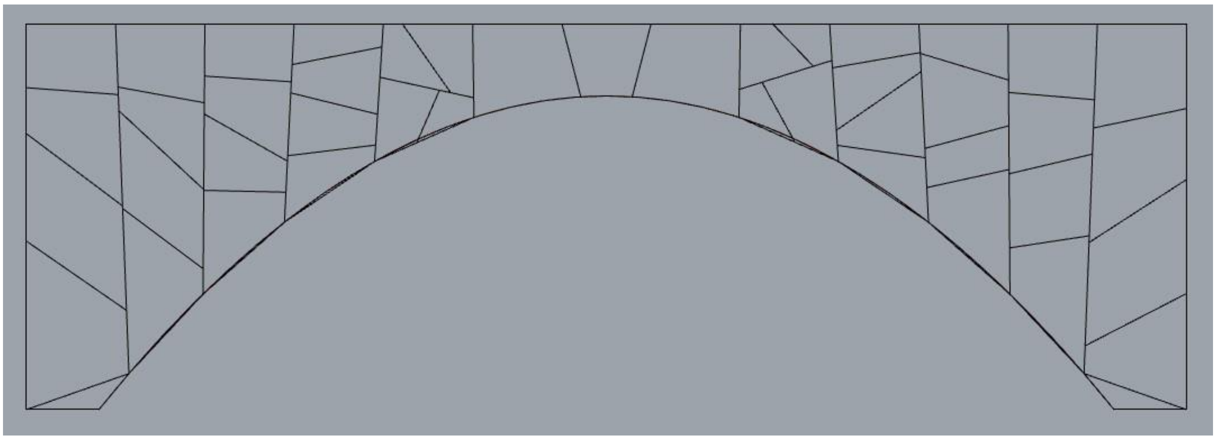


Abbildung 26: Segmentierung nach zweiter Iteration von Regel 2 und 3

Weiter ist in Abbildung 27 der Workflow des gesamten Programms zu sehen. Zur Übersichtlichkeit wurden Einzelne Teile des Programms zu Cluster zusammengefügt. Diese sind in Abschnitt 5 beschrieben. Blöcke bezüglich des Exportes werden in Abschnitt 6 genauer erläutert.

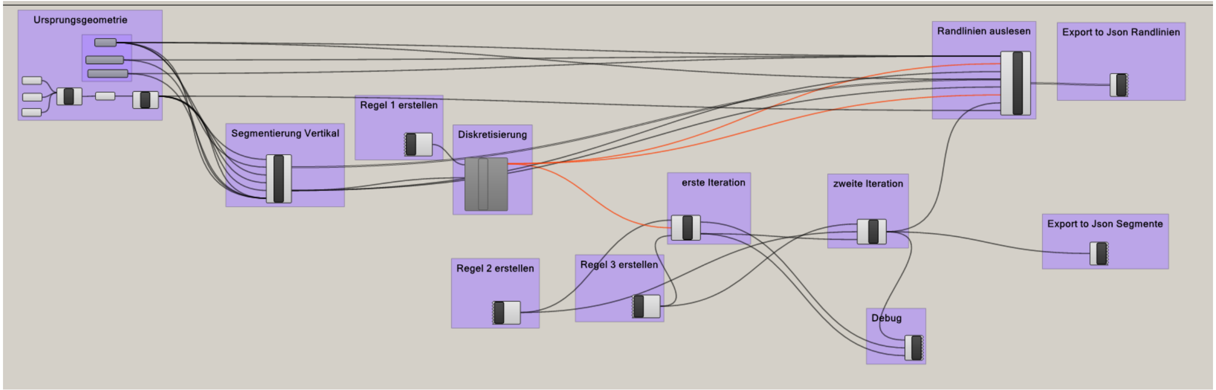


Abbildung 27: Programmablauf in Grasshopper

5.4 Erkundung des Entwurfsraums / Verbesserungen

In diesem Absatz werden alternative Ansätze zur Lösung der Segmentierung beschrieben. Ebenso die Probleme, die die genannten Ansätze beinhalten.

Ein Ansatz, die initiale vertikale Segmentierung variabel zu gestalten, ist mittels einer Regel, die eine Linie von der oberen Begrenzung zur unteren Linie erstellt (vgl. Abbildung 28). Der dabei entstehende Überstand soll mit einer zweiten Regel entfernt werden. Die Linie kann nicht durch die untere Linie begrenzt werden, da Sortal dafür keine Interaktion bietet, siehe dazu auch Abschnitt 4.3. Deshalb wird die Linienlänge auf einen sehr hohen Wert gesetzt, der einen Überstand über die untere Begrenzung sicherstellt. Der Winkel zur oberen Begrenzung wird dabei zufällig generiert.

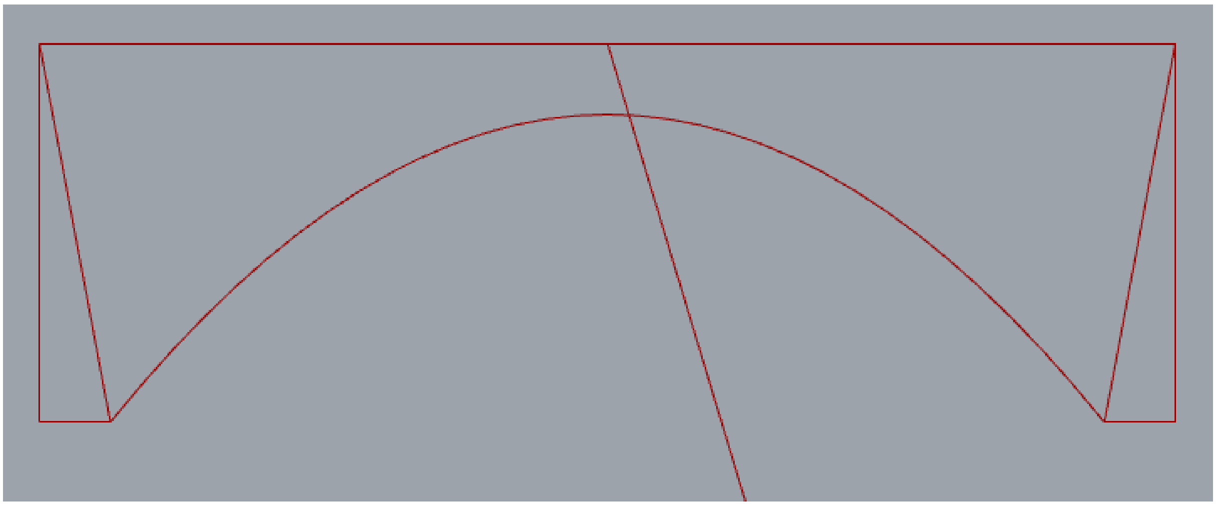


Abbildung 28: Überstand der Vertikalen Segmentierung über Begrenzung. Aufgrund des frühen Stadiums des Testes wurden links und rechts Hilfslinien eingezogen

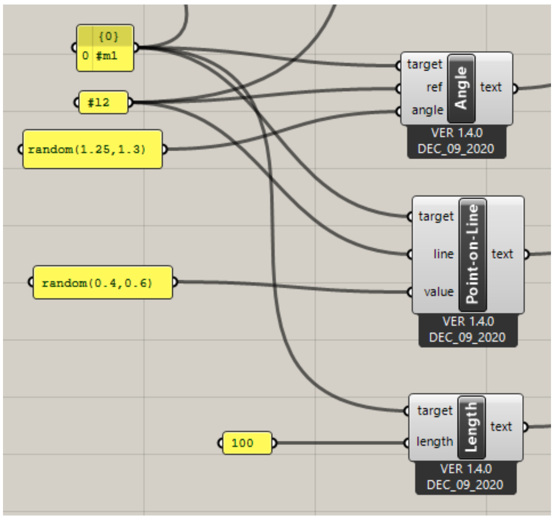


Abbildung 29: Directives für die vertikale Testsegmentierung

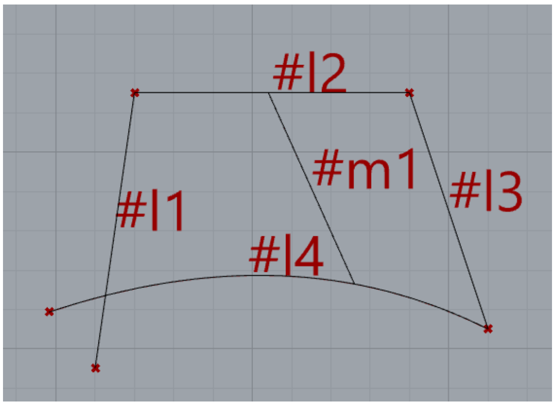


Abbildung 30: LHS und RHS für die Regel der vertikalen Testsegmentierung

Die LHS bzw. RHS sind in Abbildung 30 zusehen. Diese Regel ist ähnlich zu der beschriebenen in Abschnitt 5.2. Wie bereits in der vorhergehenden Beschreibung sind die Linien der LHS mit #l und der RHS mit #m angegeben. In dieser Regel wird die Linie #m1 durch den Winkel zu #l2, den Fußpunkt an #l2 sowie durch die absolute Länge definiert (vgl. Abbildung 29). Wie auch bei der „Point-on-Line“ Directive ist es möglich in dem Angle Directive einen zufälligen Wert für die Winkelangabe zu generieren. Die Länge wurde auf 100 gesetzt. Dies ist ein empirischer Wert, der z.B. durch die Länge der oberen Begrenzung ersetzt werden kann. Für diesen Parameter eine passende Größe zu finden wurde vernachlässigt, da die Regel keine Anwendung in der finalen Segmentierung hat. Eine zweite Regel sollte anschließend dem generierten Überstand entfernen. Dies funktioniert jedoch ausschließlich mit geraden Linien, da die untere Begrenzung noch nicht diskretisiert ist, ist dies für die gegebene Geometrie nicht möglich. Eine mögliche Lösung für das Entfernen des Überstand wäre mit der nativen Sortal Bibliothek denkbar. Dafür müsste der Schnittpunkt der hinzugefügten Linie und der Bezierkurve bestimmt werden, um den Laufparameter für die hinzugefügte Linie zu bestimmen. Mit dieser Information kann der Überstand entfernt werden. Damit wäre eine größere Streuung der vertikalen Segmentierung möglich.

Ein anderer Versuch ist mit einer nachträglichen Verschiebung der Linien eine höhere Variation der vertikalen Segmentierung zu erzeugen. Diese Regel ist komplexer als die vorhergehenden, da die LHS und RHS mehr Linien beinhalten und keine geschlossene geometrische Form bilden (vgl. Abbildung 32). Dabei ist die Linie #w13 Teil der LHS und die Linien #q1-#q3 sind die RHS. Die Linien #q1 und #q2 liegen dabei direkt auf #w13 und damit erfährt nur Teil #q3 eine Verschiebung. Der Aufbau der Regel beinhaltet zusätzlich nur eine *Directive,* die den Winkel von #q3 zu #w13 angibt. Dieser Winkel wird zufällig in dem Bereich von [-0,3; 0,3] generiert. Dabei ist es notwendig, dass die bestehenden Linien konsistent miteinander verbunden bleiben, sowie ein lückenloser Anschluss von neuen Linien.

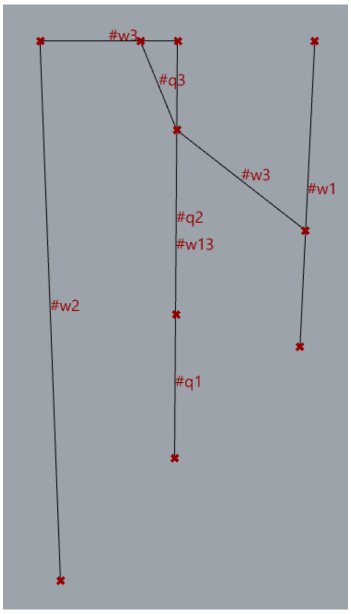


Abbildung 32: LHS und RHS der Verschiebungsregel

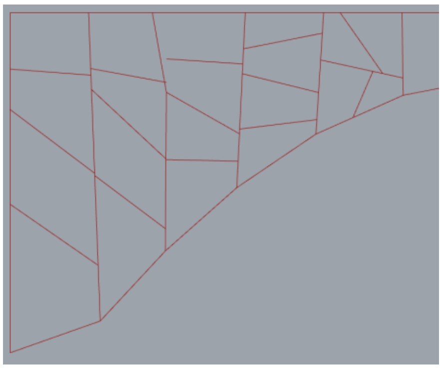


Abbildung 31: Geometrie nach Anwendung der Verschiebungsregel

In Abbildung 31 an der oberen, zweiten vertikalen Linie ist deutlich ein Spalt zu erkennen. Es gibt zwei Probleme mit dieser Regel. Zum einen können Linien nur entfernt bzw. hinzugefügt werden, aber nicht modifiziert wodurch der Anschluss der Linien zueinander verloren geht. Zum anderen ist durch die Vielzahl an vorhandenen Linien die Erstellung einer allgemein gültigen LHS nicht denkbar. Das hätte zur Bedeutung, dass für jede Konfiguration der Anschlüsse eine Regeln mit einer ähnlichen LHS wie in Abbildung 32 erstellen werden müsste. Weiter ist damit dennoch nicht ein Match für eine andere Konfiguration ausgeschlossen, was zu inkonsistenten Anschlüssen wie in Abbildung 31 führt.

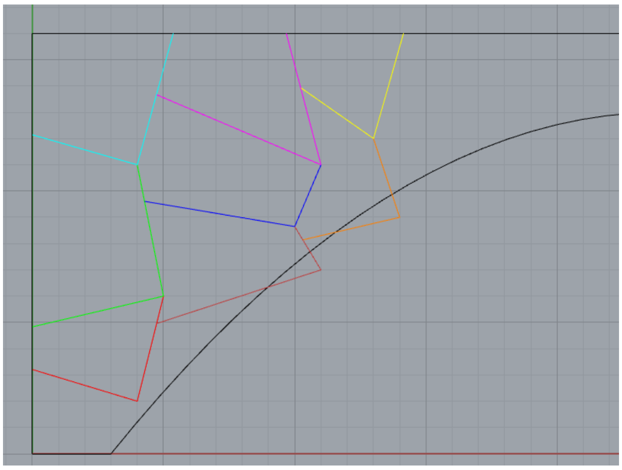


Abbildung 33: Handgezeichnete Segmentierung zur Verdeutlichung der theoretischen Segmentierung

Zusätzlich wird im Folgenden eine alternative Lösungsmöglichkeit beschrieben. Dieser Ansatz ist rein theoretisch und aufgrund der begrenzten Zeit und der schnell wachsenden Komplexität nicht getestet. Die Grundidee dabei ist, die gegebene Geometrie mit Rechtecken „zu füllen“. Eine Möglichkeit dafür ist, durch zwei zusammenhängende Linien die jeweils auf zwei verschiedene Linien anknüpfen. In Abbildung 33 ist eine mögliche Unterteilung dargestellt. Vorteil an dieser Methode ist, dass die Kantenlängen der Segmente regelmäßiger sind. Probleme, die sich bereits bei der theoretischen Überlegung ergeben, sind zum einen der Anschluss an die unteren Begrenzung bzw. der Anschluss an den Schlussstein, wenn von außen nach innen aufgefüllt wird. Analog ist der Anschluss an die vertikale Begrenzung bei einem Vorgehen von links nach rechts. Weiter müssen eine Vielzahl an Regeln erstellt werden, da die entstehenden Segmente aus Polygone mit unterschiedlicher Eckenanzahl bestehen. Weiter ist es bei diesem Vorgehen nötig die Ursprungsgeometrie kontinuierlich anzupassen, damit muss jedes neues Segment mittels eines neuen Apply-Block hinzugefügt werden. Aufgrund dieser beschriebenen Probleme wurde der Ansatz nicht implementiert.

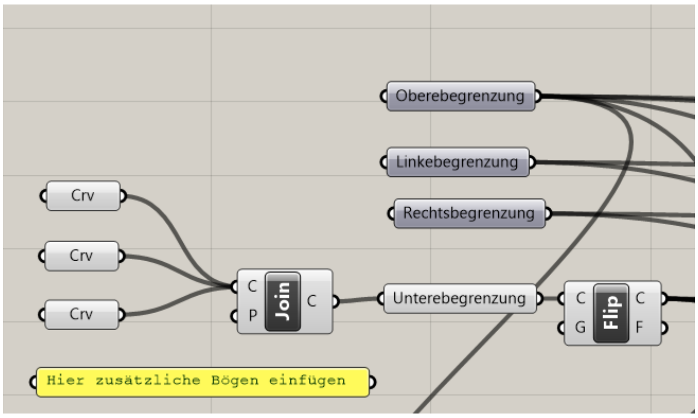


Abbildung 34: Randbedingungen für den Überbau

Eine Erweiterung des Überbaus mit mehreren Feldern ist ebenfalls denkbar. Dabei müssen nur die Randbedingungen am Anfang des Skriptes angepasst werden. Die zusätzlichen Bögen müssen in einer Reihenfolge angeschlossen werden, dass eine C0 stetige Kurve gebildet wird (siehe Abbildung 34). Mit den Join Block werden diese Kurven zu einer unteren Begrenzung zusammengeführt.

Eine weitere potzenzielle Ergänzung wäre es mit nativen Grasshopper Bausteinen alle Linien, die nach einem Iterrationsschritt von Regel 2 und Regel 3 erstellt werden, auf die Länge zu prüfen. Falls diese Länge unter einem Schwellenwert liegt, wird die Linie gelöscht. Damit wird indirekt eine minimale Segmentierungsgröße erreicht.

6 Bewertungskriterien

Im folgenden Abschnitt wird die Bewertung der Segmentierung verdeutlicht. Dabei werden drei Beispiele, die durch den Ansatz von Generative Design erzeugt wurden, verwendet. Die Bewertung fokussiert sich dabei auf drei Punkte:

1. Regelmäßigkeit der Winkel in einem Segment,
2. Gleichheit der Kantenlängen,
3. Hauptspannungsrichtungen an Kantenmittelpunkten.

Zu jedem Kriterium wird eine Kostenfunktion erstellt, damit die Beispiele verglichen werden können. Diese Beurteilungsfunktionen sind in Matlab implementiert. Die Gründe dafür sind die umfangreichen Bibliotheken zur Graphentheorie, der implementierte FE-Solver sowie der Umstand das Matlab eine imperative Programmiersprache ist. Grasshopper bietet wenig bzw. keine dieser Funktionalitäten.

6.1 Auswertung

Für die Auswertung der Geometrie ist es zunächst notwendig diese von Grasshopper nach Matlab zu exportieren. Dafür wird das Datenformat JSON verwendet, da es einen einfachen Export sowie Import ermöglicht. Für den Export der Geometrie werden alle Linien an den Schnittpunkten geteilt.

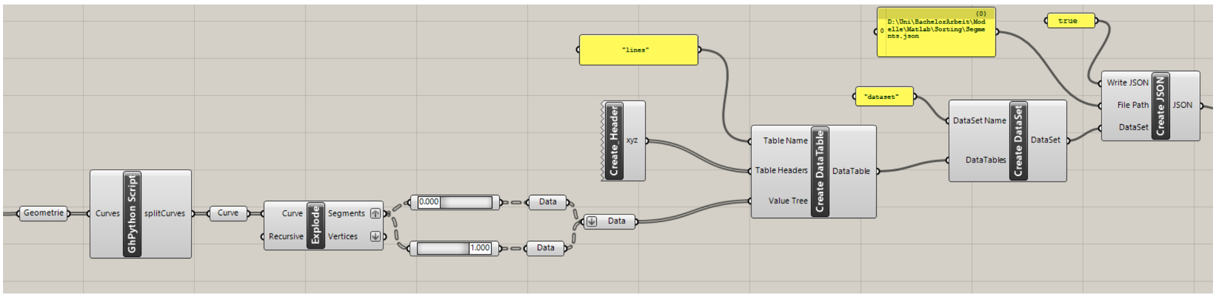
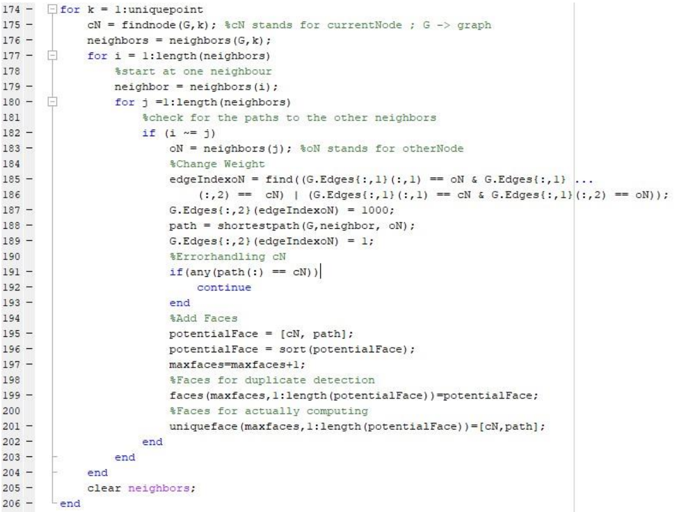


Abbildung 35: Codeausschnitt für den Export der Geometrie in das JSON Datenformat

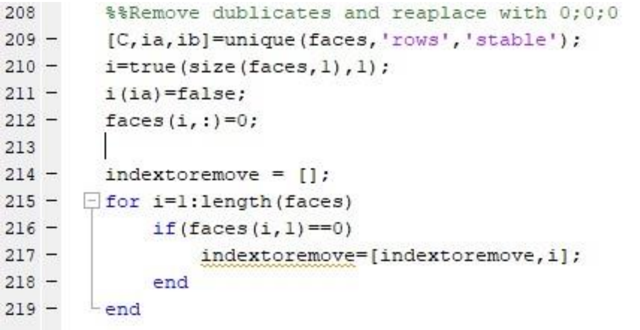
Zusätzlich wird der Bogen an der Unterseite entfernt. Sortal bietet weitere die Möglichkeit Differenzen von Formen zu bilden, damit kann die Bezierkurve entfernt werden. Somit repräsentiert die Diskretisierung den unteren Rand der Geometrie, diese wird exportiert und in Matlab eingelesen. Analog zu dem Export der Segmentierung werden ebenso die Ränder exportiert. Diese Information ist später für die Hauptspannungsrichtungsanalyse (Abschnitt 6.1.3) notwendig. In der Auswertung der Winkel sowie der

Längen ist es erforderlich jedes Segment einzeln zu erkennen. Durch den Einsatz von Formgrammatiken werden nur Linien erzeugt aber keine neuen Flächen. Das bedeutet, dass es für CAD Programme nicht ersichtlich ist, welche Linien ein Segment ergeben. Diese Information wird mittels Matlab ermittelt. Darin werden die exportierten Linien eingelesen. Anschließend werden den Endpunkten der Linien IDs zugewiesen. Mit diesen Informationen der Linien und der IDs der Punkte ist es möglich, einen ungerichteten Graphen zu erstellen. In diesen Graph sind alle Kanten mit dem Wert 1 gewichtet. Die im Folgenden beschriebene Faceerkennung ist in Abbildung 36 zu sehen. Um die Flächen aus dem Graphen abzuleiten, wird die Funktion „shortestpath“ verwendet. Wie der Name bereits beschreibt, gibt diese Funktion den kürzesten Weg zwischen zwei Knoten eines Graphen zurück. Um die Flächen zu finden, wird jeder Punkt auf dessen Nachbarn geprüft. Anschließend wird der kürzeste Pfad zwischen den Nachbarn gesucht. Dabei wird der Weg über den ursprünglichen Knoten ausgeschlossen d.h. die direkte Kante zwischen den beiden Knoten. Dies geschieht mittels einer Erhöhung der Gewichtung, sowie das Entfernen von gefundenen Pfaden, falls der aktuelle Knoten in dem Pfad enthalten ist (vgl. Abbildung 36 Zeile 191). Zusätzlich wird dem zurückgegeben Pfad den aktuellen Knoten hinzugefügt (vgl. Abbildung 36 Zeile 195). Anschließend werden aus der Liste die Duplikate entfernt (vgl Abbildung 37). Dafür wird ein zweites Array mit den erkannten Faces erstellt (vgl. Abbildung 37



```
174 -   for k = 1:uniquepoint
175 -       cN = findnode(G,k); %cN stands for currentNode ; G -> graph
176 -       neighbors = neighbors(G,k);
177 -       for i = 1:length(neighbors)
178             %start at one neighbour
179 -           neighbor = neighbors(i);
180 -           for j =1:length(neighbors)
181                 %check for the paths to the other neighbors
182 -               if (i ~= j)
183 -                   oN = neighbors(j); %oN stands for otherNode
184                     %Change Weight
185 -                   edgeIndexoN = find((G.Edges{:,1}(:,1) == oN & G.Edges{:,1} ...
186                         (:,2) ==  cN) | (G.Edges{:,1}(:,1) == cN & G.Edges{:,1}(:,2) == oN));
187 -                   G.Edges{:,2}(edgeIndexoN) = 1000;
188 -                   path = shortestpath(G,neighbor, oN);
189 -                   G.Edges{:,2}(edgeIndexoN) = 1;
190                     %Errorhandling cN
191 -                   if(any(path(:) == cN))
192 -                       continue
193 -                   end
194                     %Add Faces
195 -                   potentialFace = [cN, path];
196 -                   potentialFace = sort(potentialFace);
197 -                   maxfaces=maxfaces+1;
198                     %Faces for duplicate detection
199 -                   faces(maxfaces,1:length(potentialFace))=potentialFace;
200                     %Faces for actually computing
201 -                   uniqueface(maxfaces,1:length(potentialFace))=[cN,path];
202 -               end
203 -           end
204 -       end
205 -       clear neighbors;
206 -   end
```

Abbildung 36: Quellcode für die Flächenerkennung der exportierten Geometrie



```
208     %%Remove dublicates and reaplace with 0;0;0
209 -   [C,ia,ib]=unique(faces,'rows','stable');
210 -   i=true(size(faces,1),1);
211 -   i(ia)=false;
212 -   faces(i,:)=0;
213
214 -   indextoremove = [];
215 -   for i=1:length(faces)
216 -       if(faces(i,1)==0)
217 -           indextoremove=[indextoremove,i];
218 -       end
219 -   end
```

Abbildung 37: Quellcode für die Entfernung doppelter Flächen

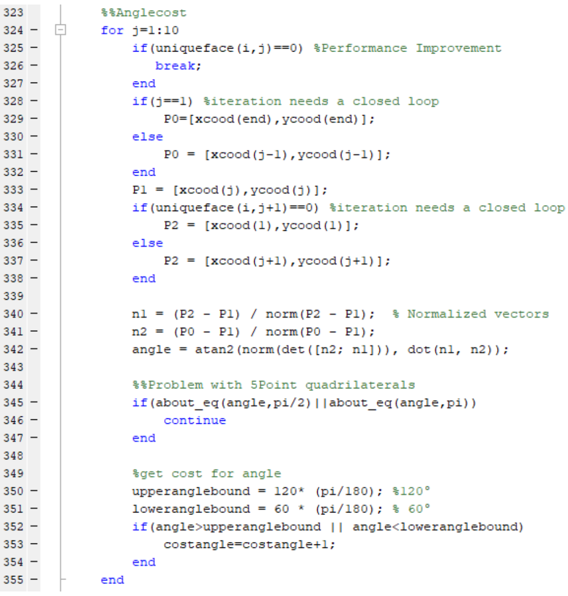
Zeile 198 – 201), diese Einträge werden aufsteigend sortiert. Für die Duplikatentfernung wird der Befehl „unique“ verwendet. Damit werden doppelt vorhandene Flächen zunächst mit 0 ersetzt und im Weiteren entfernt. Zusätzlich ist es an den Rändern möglich, dass zwei Flächen fälschlicherweise als eine zusammenhängende interpretiert werden. Diese Fälle werden ebenfalls nachträglich entfernt (vgl. Abbildung 38), indem alle Flächen geprüft werden, ob bereits ein Bereich darin vorhanden ist. Beispielsweise ist die Fläche mit den Knoten [1,2,3,4] in [1,2,3,5,6,4] enthalten. Die größere Fläche wird entfernt (vgl. Abbildung 38 Zeile 256). Die Flächen werden in der Variable uniqueface abgespeichert. Das Format dabei ist wie folgt, in einer Zeile sind die Knoten einer Fläche gespeichert. Zellen die nicht belegt sind werden mit 0 gefüllt. Dies ist notwendig, da Matlab nur Vektoren gleicher Längen zu einem 2D- Vektor zusammenführen kann.

```
232 - for i=1:length(uniqueface)
233       %Performance improvment If Face already is min -> can´t contain a
234       %larger Face
235 -     if(nonzeroentry(i)==minsize)
236 -        continue
237 -     end
238       %iterate throw all following faces
239 -     for j=i+1:length(uniqueface)
240 -         count=0;
241           %check if Face if larger Face is already contained
242 -         for m=1:10
243 -             if(uniqueface(i,m)==0)
244 -                     break
245 -             end
246 -             for n=1:10
247 -                 if( uniqueface(j,n)==0)
248 -                     break
249 -                 end
250 -                 if(uniqueface(i,m)==uniqueface(j,n))
251 -                     count=count+1;
252 -                     break;
253 -                 end
254 -             end
255 -         end
256 -         if(count>3)
257 -             if(nonzeroentry(i)<nonzeroentry(j))
258 -                 indextoremove=[indextoremove,j];
259 -             else
260 -                 indextoremove=[indextoremove,i];
261 -             end
262 -         end
263 -     end
264 - end
```

Abbildung 38: Quellcode zur Entfernung von doppelten Randflächen

6.1.1 Regelmäßigkeit der Winkel in einem Segment

Die Winkel werden segmentweise berechnet. Dafür ist es notwendig zeilenweise durch uniqueface zu iterieren. In jedem Segment werden zwei benachbarte Kanten betrachtet und dabei der eingeschlossene Winkel berechntet (vgl. Abbildung 39 Zeile 340ff). Die Kostenfunktion für regelmäßige Winkel stetzt sich wie folgt zusammen: als Basis wird für jede Ecke ein 90° Winkel angesetzt, die zugelassene Abweichung davon ist +-30° (vgl. Abbildung 39 Zeile 350f).



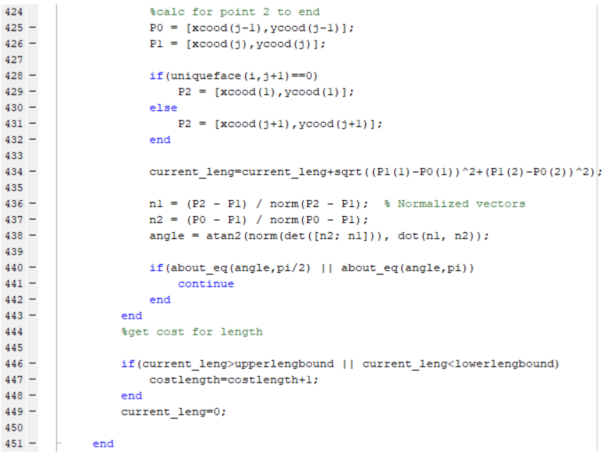
```
323       %%Anglecost
324 -     for j=1:10
325 -         if(uniqueface(i,j)==0) %Performance Improvement
326 -            break;
327 -         end
328 -         if(j==1) %iteration needs a closed loop
329 -             P0=[xcood(end),ycood(end)];
330 -         else
331 -             P0 = [xcood(j-1),ycood(j-1)];
332 -         end
333 -         P1 = [xcood(j),ycood(j)];
334 -         if(uniqueface(i,j+1)==0) %iteration needs a closed loop
335 -             P2 = [xcood(1),ycood(1)];
336 -         else
337 -             P2 = [xcood(j+1),ycood(j+1)];
338 -         end
339
340 -         n1 = (P2 - P1) / norm(P2 - P1);  % Normalized vectors
341 -         n2 = (P0 - P1) / norm(P0 - P1);
342 -         angle = atan2(norm(det([n2; n1])), dot(n1, n2));
343
344           %%Problem with 5Point quadrilaterals
345 -         if(about_eq(angle,pi/2)||about_eq(angle,pi))
346 -             continue
347 -         end
348
349           %get cost for angle
350 -         upperanglebound = 120* (pi/180); %120°
351 -         loweranglebound = 60 * (pi/180); % 60°
352 -         if(angle>upperanglebound || angle<loweranglebound)
353 -             costangle=costangle+1;
354 -         end
355 -     end
```

Abbildung 39: Quellcode der Winkel-Kostenfunktion

Falls sich der berechnete Winkel außerhalb dieser Grenzen befindet wird die Variable „Costangle“ um den Wert 1 erhöht. Eine Ausnahme davon ist, wenn sich ein Winkel von 180° bzw. 360° ergibt. In diesen Fall besitzt die Fläche fünf Punkte ist aber geometrisch gesehen ein Viereck, weshalb diese Werte ignoriert werden.

6.1.2 Gleichheit der Kantenlängen

Ebenso wie die Winkel werden auch die Kantenlängen segmentweise geprüft. Die Kostenfunktion der Kantenlänge wird in diesem Anwendungsfall nicht auf die absolute Länge bezogen, da sich die Segmentierung durch mehrere Iterationen des Workflows in Abschnitt 5.3 beliebig einstellen lässt. Deshalb wird die durchschnittliche Länge einer Kante als Bezugsgröße gewählt. Diese wird wie folgt errechnet: die Kantenlängen jedes Segmentes werden aufsummiert und durch die vierfache Anzahl der Flächen geteilt. Hier wird ausgenutzt, dass jede geometrische Fläche aus genau vier Seiten besteht. Die Kostenfunktion wird um 1 erhöht falls eine Kantenlänge nicht im Bereich vom 0,66 bis zum 1,5 fachen der durchschnittlichen Länge liegt. Analog zu dem Problem in Abschnitt 6.1.1, in dem Vierecke mit fünf Eckepunkten existieren, müssen Linien, die von Winkeln von 180° bzw. 360° eingeschlossen werden, als eine zusammenhängende Linie betrachtet werden. Da die Berechnung durch die Kanten iteriert, werden nur Längen addiert die Kollinear sind (vgl. Abbildung 40 Zeile 434, Zeile 440, Zeile 449) In Abbildung 40 gibt es einen Fall der separat abgefragt wird. Falls die erste und letzte Kante kollinear sind, müssen in P0 die Werte des letzten Eintrags der Koordinaten zugewiesen werden. Zusätzlich muss die Länge der letzten Kante in diesen Fall berechnet werden, da man für die Längenberechnung der Kanten die Knoten i und i-1 verwendet.



```
424             %calc for point 2 to end
425             P0 = [xcood(j-1),ycood(j-1)];
426             P1 = [xcood(j),ycood(j)];
427
428             if(uniqueface(i,j+1)==0)
429                 P2 = [xcood(1),ycood(1)];
430             else
431                 P2 = [xcood(j+1),ycood(j+1)];
432             end
433
434             current_leng=current_leng+sqrt((P1(1)-P0(1))^2+(P1(2)-P0(2))^2);
435
436             n1 = (P2 - P1) / norm(P2 - P1);  % Normalized vectors
437             n2 = (P0 - P1) / norm(P0 - P1);
438             angle = atan2(norm(det([n2; n1])), dot(n1, n2));
439
440             if(about_eq(angle,pi/2) || about_eq(angle,pi))
441                 continue
442             end
443         end
444         %get cost for length
445
446         if(current_leng>upperlengbound || current_leng<lowerlengbound)
447             costlength=costlength+1;
448         end
449         current_leng=0;
450
451     end
```

Abbildung 40: Quellcode der Kantenlängen-Kostenfunktion

6.1.3 Hauptspannungsrichtungen an Kantenmittelpunkten

Anders als für die Winkel und Längen Kostenfunktionen, werden die Kosten für die Hauptspannungsrichtungen pro Scherfuge also pro Linie berechnet. Für die Berechnung der Hauptspannungsrichtungen wird ein FE - Model verwendet. Da es sich in dieser Bachelor Arbeit um einen frühen Entwurf der Segmentierung handelt, werden einige Vereinfachungen angenommen.

- Die Berechnungen basieren auf einem Kontinuum.
- Es wird eine homogene Gleichstreckenlast über das gesamte Feld angesetzt.
- Das Eigengewicht der Konstruktion wird nicht berücksichtigt.
- Die Auflager der Konstruktion werden als Punkte modelliert
- Die Hauptspannungen werden für die Mitte der Scherfuge berechnet.
- Für die Berechnung wird die Hauptspannungsrichtung verwendet die am günstigsten zum tatsächlichen Winkel liegt.

Die Berechnung des FE Models wird in Matlab mittels eines PDE Solvers realisiert. Dabei werden nur Punkte innerhalb des Kontinuums berücksichtigt(vgl. Abbildung 42 Zeile 492ff). In dieser Kostenfunktion wird, wie bereits beschriebenen, durch die Scherfugen iteriert. Dabei wird der tatsächliche Winkel mit dem idealen Winkel aus der FE Berechnung verglichen. Dabei wird die Kostenfunktion um 1 erhöht, falls der tatsächliche Winkel eine Abweichung zwischen +/- 15° und +/- 25° besitzt. Falls die Abweichung mehr als +/- 25° beträgt, wird die Kostenfunktion um 2 erhöht (vgl. Abbildung 41 Zeile 506-531). In dieser If Abfragen werden drei Fälle behandelt, in dem der kleinste Zwischenwinkel ermittelt wird. Diese sind zwischen der Steigung und der berechneten Hauptspannungsrichtung, der Steigung und der Hauptspannungsrichtung +/- 90°. Bei der Auswertung wichtig zu beachten ist, dass durch die erste Segmentierung ca. 56 Scherfugen (Zahl ist stark abhängig von den Teilsegmenten und der Anzahl der Regelanwendungen) entstehen. Diese sind immer vertikal und gehen daher auch meist +2 in die Kostenfunktion mit ein, da sie nicht zufällig generiert werden (vgl. Abbildung 42).

```
492  for i=1:2:length(points)
493      if(any(ismember(boundaraindex(:)*2-1,i)))
494          continue
495      end
496      j=j+1;
497      m=(points(2,i+1)-points(2,i))/(points(1,i+1)-points(1,i));
498      %convert in to an format to evaluate
499      angleface=rad2deg(atan(-1/m)) ;
500      %get rid of - sign
501      angleface=angleface+360;
502      angleface=mod(angleface,180);
503      geradenangel=[geradenangel,angleface];
504
505      %%check both angles
506      if(abs(angleface-phil(j))<abs(angleface-(phil(j)+90)))
507         if(abs(angleface-phil(j))<abs(angleface-(phil(j)-90)))
508             if(abs(angleface-phil(j))>15)
509                 costfem=costfem+1;
510                 if(abs(angleface-phil(j))>25)
511                     costfem=costfem+1;
512                 end
513             end
514             continue
515         end
516         if(abs(angleface-(phil(j)-90))>15)
517                 costfem=costfem+1;
518                 if(abs(angleface-(phil(j)-90))>25)
519                     costfem=costfem+1;
520                 end
521         end
522         continue
523      end
524      if(abs(angleface-phil(j)+90)>15)
525             costfem=costfem+1;
526             if(abs(angleface-phil(j)+90)>25)
527                 costfem=costfem+1;
528             end
529      end
530
531  end
```

Abbildung 41: Kostenfunktion für die Hauptspannungsrichtung

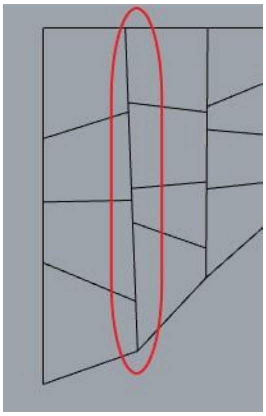


Abbildung 42: Vertikale Scherfugen durch Segmentierung

6.2 Beispiele Segmentierungen

Im Folgenden werden exemplarisch 3 Varianten der Segmentierung gezeigt, analysiert und bewertet.

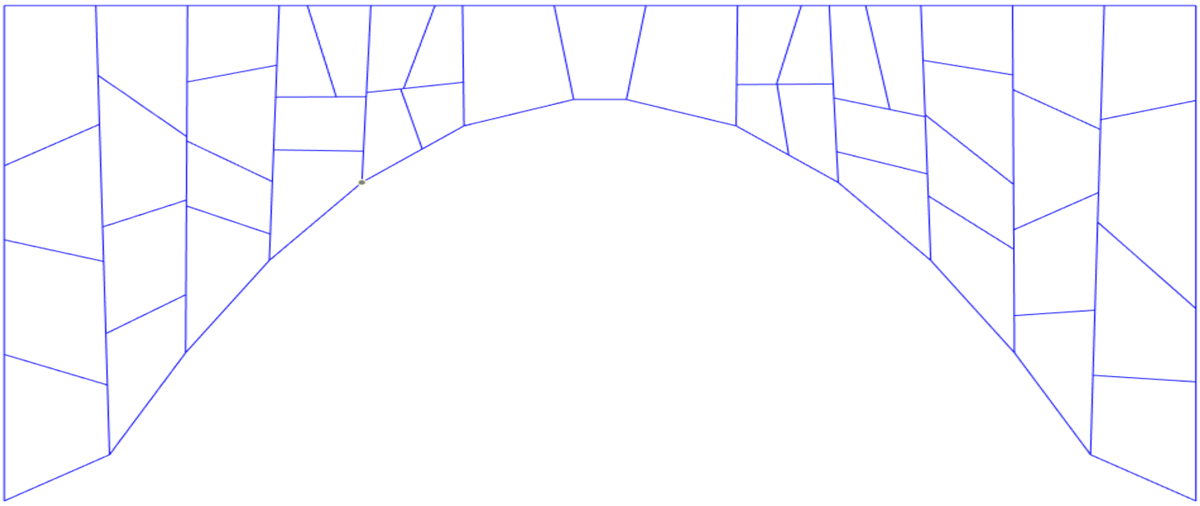


Abbildung 43: Variante 1 der Segmentierung

Kostenfunktion: Hauptspannungsrichtung: 145

Winkel: 26

Längen: 46

In Abbildung 43 ist eine gute Segmentierung hinsichtlich der Hauptspannungsrichtungen zu sehen. Auf der linken Seite und in der Mitte verlaufen die Hauptspannungen kollinear bzw. senkrecht zu den Scherfugen. Zum Vergleich sind in Abbildung 44 die Hauptspannungsrichtungen dargestellt.

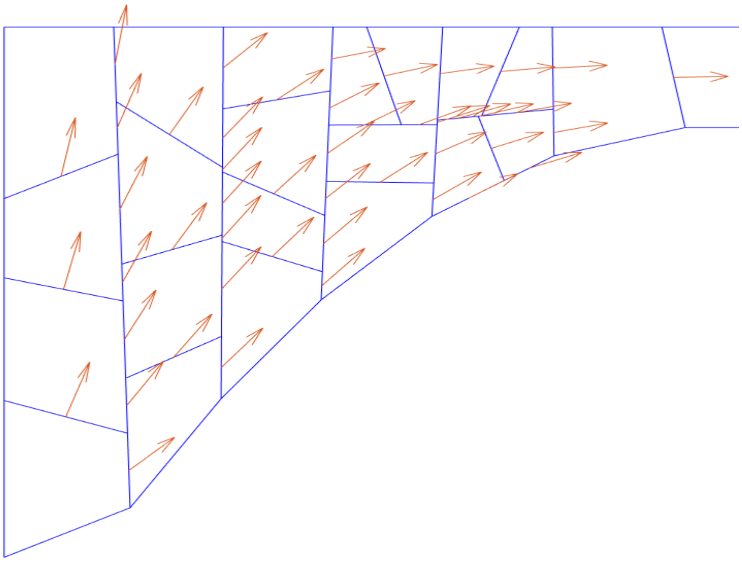


Abbildung 44: Hauptspannungsrichtungen für Variante 1

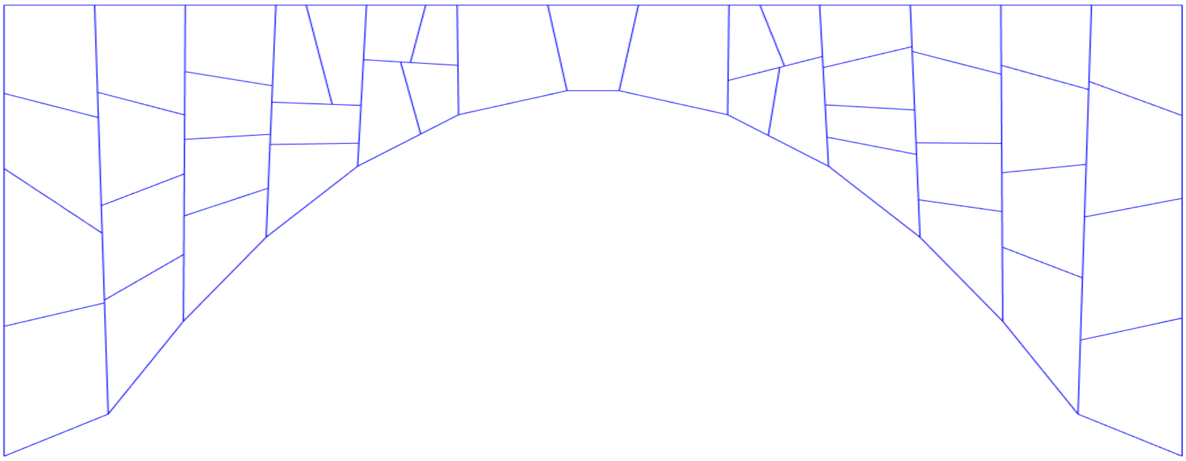


Abbildung 45: Variante 2 der Segmentierung

Kostenfunktion: Hauptspannungsrichtung: 157

Winkel: 14

Längen: 41

In Abbildung 45 ist Variante 2 zu sehen. Auffällig sind die regelmäßigen Vierecke. Die durch die meist horizontalen Linien der Segmentierung entstehen. Im Vergleich dazu ist in Abbildung 46 eine Segmentierung mit der gleichen Bewertung der Hauptspannungsrichtungen aber unregelmäßigeren Vierecken. Durch die steileren Winkel ist ebenfalls die Kostenfunktion der Längen gestiegen.

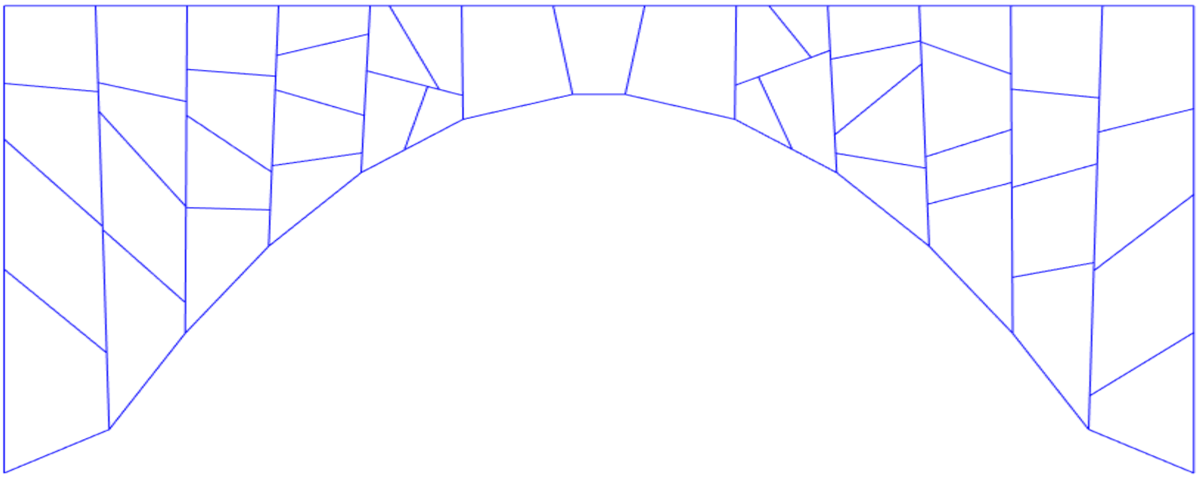


Abbildung 46: Variante 3 der Segmentierung

Kostenfunktion: Hauptspannungsrichtung: 157

Winkel: 48

Längen: 54

7 Zusammenfassung und Fazit

In dieser Bachelorarbeit wurde ausgehend von einer abstrahierten Geometrie, ein Brückenüberbau automatisch segmentiert. Dazu wurde eine Formgrammatik entwickelt, die dadurch entstehenden Varianten werden anhand von statischen und geometrischen Faktoren beurteilt.

Die größte Herausforderung war dabei die Formulierung der Regeln der zu verwendeten Formgrammatik. Für diese Anwendung existiert kein Debugger, daher war die Fehlersuche zeitaufwändig. Zusätzlich ist der Aufwand eine Regel zu erstellen, die exakt die erwarteten Resultate liefert, hoch. Da zum einen durch die kaskadierende Anwendung der Regeln komplexe Formen entstehen und es dann schwierig ist sicherzustellen, dass alle entstehenden Formen geschlossene Polygone sind. Zum anderen ist das Finden der richtigen *Predicates* nicht trivial, damit nur die gewünschten Matches gefunden werden.

Ein großes Problem dieses Ansatzes ist, die lange Rechenzeit. Die auf die serielle Implementierung von Grasshopper sowie der fehlenden Optimierung des SGI zurückzuführen ist. Ein weiteres Problem ist die Restriktion einer visuellen Programmiersprache. Damit ist ein pipelining der Daten nicht möglich, da keine Schleifen programmiert werden können. Somit muss jede Segmentierung per Hand gestartet und abgespeichert werden.

Das Potenzial zur Segmentierung mittels Formgrammatik ist hoch. Allerdings müssen dafür noch mehrere alternative Regelsets getestet werden. Sowie eine optimierte Ausführung in einem CAD – Programm, hinsichtlich Laufzeit und anschließender Bewertung. Zusätzlich werden mehr Features (*Predicates* und *Directives*) benötigt, um eine bessere Segmentierung zu realisieren. Alternativ können auch gut segmentierte Teile der Geometrie per Hand zusammengefügt werden, wenn von dem starren Ansatz der Formgrammatiken abgewichen wird.

Literaturverzeichnis

Benrós, Deborah; Duarte, José P.; Hanna, Sean (2012): A New Palladian Shape Grammar. In: *International Journal of Architectural Computing* 10 (4), S. 521–540. DOI: 10.1260/1478-0771.10.4.521.

Grasl, Thomas; Economou, Athanassios (2013): From topologies to shapes: parametric shape grammars implemented by graphs. In: *Environ. Plann. B* 40 (5), S. 905–922. DOI: 10.1068/b38156.

Krish, Sivam (2011): A practical generative design method. In: *Computer-Aided Design* 43 (1), S. 88–100. DOI: 10.1016/j.cad.2010.09.009.

Sortal.org (2017). Online verfügbar unter http://www.sortal.org/grammars/index.html, zuletzt aktualisiert am 01.12.2017, zuletzt geprüft am 22.05.2021.

Stiny, George (1980): Introduction To Shape And Shape Grammars.

Stouffs, Rudi (2019 - 2019): Predicates and Directives for a Parametric-associative Matching Mechanism for Shapes and Shape Grammars. In: Blucher Design Proceedings. 37 Education and Research in Computer Aided Architectural Design in Europe and XXIII Iberoamerican Society of Digital Graphics, Joint Conference (N. 1). Porto, Portugal, 11.09.2019 - 13.09.2019. São Paulo: Editora Blucher, S. 403–414.

Zwettler, Monika (2020): Was ist eigentlich Generatives Design? In: *konstruktionspraxis*, 19.05.2020. Online verfügbar unter https://www.konstruktionspraxis.vogel.de/was-ist-eigentlich-generatives-design-a-931380/, zuletzt geprüft am 02.09.2021.

Anhang A

Auf der beigefügten CD befindet sich folgender Inhalt:

- Der schriftliche Teil der Arbeit als Worddokument
- Grasshopper Skript der Formgrammatik
- Rhino Datei der Beispielgeometrie
- Matlab Skipt für die Auswertung

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 26. Oktober 2021

Vorname Nachname

Stefan Huber

Unterarbing 7

D-84494 Niedertaufkrichen

Stefan.1995.huber@tum.de