

# Todo list

Überdenken, ob Code hier wirklich notwendig . . . . . 13

**Wilhelm-Ostwald-Schule, Gymnasium der Stadt Leipzig**

## **Dokumentation zur Besonderen Lernleistung**

**Im Fachbereich**

Informatik

**Thema**

Erzeugung eines 3D-Modells eines Gebäudes

anhand des Grundrisses

**Vorgelegt von**

Johann Bartel und Peter Oehme

**Schuljahr**

2017/2018

**Externe Betreuer**

Herr Prof. Dr. Gerik Scheuermann, Herr Tom Liebmann

Universität Leipzig Fakultät für Mathematik und Informatik

**Interner Betreuer**

Herr Rai-Ming Knospe

Leipzig, den 22.12.2017

## **Bibliographische Beschreibung**

Bartel, Johann und Oehme, Peter

„Erzeugung eines 3D-Modells eines Gebäudes anhand des Grundrisses“

39 Seiten, Y Anlagen, 21 Abbildungen

# **Erzeugung eines druckbaren 3D-Modells eines Gebäudes anhand des Grundrisses**

Die Zielstellung dieser BeLL ist es, den Grundriss eines Hauses, der aus einem Konstruktionsprogramm entnommen wurde, in eine druckbare 3D-Datei zu konvertieren. Diese Umwandlung wird mithilfe eines Programmes mit eingebetteten selbst entworfenen mathematischen Operationen realisiert.

Aus dem Grundriss, welcher eine 2D-Datenmenge darstellt, werden die digitalen Anweisungen für die 3D-Strukturen Wände, Grundflächen und Eckpfeiler berechnet. Diese Anweisungen lassen sich nach der Umwandlung in einem Modellierungsprogramm für den Druckvorgang umwandeln. Die Berechnungen der Umwandlung laufen so ab, dass an allen Elementen des finalen Modells komplementäre Stecker angebracht werden, die zusammen als ein Stecksystem fungieren. Eckpfeiler dienen hierbei als Verbindungsstücke zwischen den Wänden und Bodenplatten, welche somit für die Stabilität des Objektes sorgen. Das Stecksystem ermöglicht ein Zusammensetzen aller Bauteile zu einem stabilen Modell. Dadurch entsteht ein Modell, welches aufgrund der genannten Modifikationen transportabel und geeignet für Präsentationen ist.

Architekten können die 3D-Darstellung der Immobilie nutzen, um mehr Eindruck über das Objekt zu erlangen und eine mögliche Inneneinrichtung zu planen.

Johann Bartel und Peter Oehme

---

# Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>1 Einleitung</b> . . . . .                               | <b>3</b>  |
| <b>2 Wissenschaftliche Grundlagen</b> . . . . .             | <b>5</b>  |
| 2.1 Planare Graphen . . . . .                               | 5         |
| 2.2 Doubly connected edge list . . . . .                    | 5         |
| 2.3 Convex Hull . . . . .                                   | 7         |
| 2.4 Oriented Minimum Bounding Box . . . . .                 | 7         |
| 2.5 AutoCAD . . . . .                                       | 8         |
| 2.6 OpenSCAD . . . . .                                      | 9         |
| 2.7 3D-Drucker MakerBot Replicator™ 2 . . . . .             | 11        |
| <b>3 Vorgehen zur Problemlösung</b> . . . . .               | <b>13</b> |
| 3.1 Einlesen des Grundrisses . . . . .                      | 13        |
| 3.1.1 Funktionsweise der Bibliothek <i>kabeja</i> . . . . . | 13        |
| 3.1.2 Funktionsweise der GUI . . . . .                      | 14        |
| 3.2 Erstellen der Doubly-connected Edge List . . . . .      | 17        |
| 3.2.1 Verarbeitung der Linien . . . . .                     | 17        |
| 3.2.2 Vector-to-Node Konvertierung . . . . .                | 17        |
| 3.2.3 Line-to-Edge Konvertierung . . . . .                  | 18        |
| 3.2.4 Zwillingskantengenerierung . . . . .                  | 18        |
| 3.2.5 Nachfolger- und Vorgängerermittlung . . . . .         | 18        |
| 3.2.6 Flächenerstellung . . . . .                           | 20        |
| 3.2.7 Vervollständigung der Knoten . . . . .                | 21        |
| 3.3 Aufbau der Einzelbauteile . . . . .                     | 22        |
| 3.3.1 Eckpfeiler . . . . .                                  | 22        |

|          |  |           |
|----------|--|-----------|
| 3.3.2    | Wandstücke . . . . .                         | 25        |
| 3.3.3    | Grundplatten . . . . .                       | 26        |
| 3.3.4    | Zuweisen von Berechnungskonstanten . . . . . | 27        |
| 3.4      | Druckvorgang . . . . .                       | 31        |
| 3.4.1    | Minimierung der Druckvorgänge . . . . .      | 31        |
| 3.4.2    | Ermittlung der Größen . . . . .              | 31        |
| 3.5      | Ausgabe der Resultate . . . . .              | 34        |
| 3.5.1    | Erstellung des Ausgabeordners . . . . .      | 34        |
| 3.5.2    | OpenSCAD Java Interface . . . . .            | 34        |
| 3.5.3    | STL Konvertierung . . . . .                  | 35        |
| <b>4</b> | <b>Ausblick . . . . .</b>                    | <b>37</b> |
|          | <b>Internetquellenverzeichnis . . . . .</b>  | <b>38</b> |

# 1. Einleitung

In den letzten Jahren gewannen 3D-Drucker immer mehr Bedeutung, sowohl für wissenschaftliche als auch für wirtschaftliche Zwecke. Sie werden genutzt, um verschiedene Gegenstände oder Bauteile des Eigenbedarfs selbst herzustellen oder nach Belieben anzupassen. Entsprechend naheliegend war es, dass schnell die ersten Modelle nachgebildet wurden, oder man sich an beliebten Steckbausteinsystemen wie LEGO orientierte, um sich eigene Sets zu drucken.

Diese Eignung für den Modellentwurf und Modellbau erweckte auch die Idee, ein Modell eines Hauses zu drucken, welches in sich aus strukturierten Bauteilen zusammengesetzt ist und somit auch das Entfernen einzelner dieser Bauteile erlaubt, um einen einfacheren Einblick in das Modell zu erhalten. Kombiniert mit dem Interesse an der Architektur entstand die Überlegung, ob es möglich wäre, anhand eines Grundrisses, welchen man aus einem Konstruktionsprogramm wie beispielsweise AutoCAD in Form einer .dxf-Datei erhalten kann, ein 3D-Modell des Hauses zu erzeugen, welches mithilfe eines Programmes automatisch in die vorgesehenen Bauteile zerlegt wurde, das im Anschluss von einem 3D-Drucker gedruckt werden kann. Dem Nutzer wird demnach nur zuteil, den Grundriss einzuspeisen und die ausgegebenen Bauteile korrekt auszudrucken, was ihm einen aufwendigen Modellierungs- und Zerlegungsprozess erspart.

Ein solches Modell soll dann Architekten als Möglichkeit vorliegen, um ihren Kunden vor dem Kauf eines Hauses näheren Einblick in die Immobilie zu gewähren und mit ebenfalls 3D-gedruckten Möbeln bereits im Voraus erste Einrichtungsideen zu überprüfen. Diese Methode würde auf ein ausgeprägtes dreidimensionales Vorstellungsvermögen des Kunden verzichten und als Ergänzung zum vorgelegten Grundriss funktionieren.

Die Umsetzung des Programms erfolgt in der Programmiersprache Java. Um die Problemstellung zu bewältigen, musste zunächst eine systematisch

einzuhaltende Zerteilung des Modells festgelegt werden.

## 2. Wissenschaftliche Grundlagen

### 2.1 Planare Graphen

Zur einfacheren Handhabung des Grundrisses wird dieser in einen planaren Graphen umgewandelt. Ein planarer, auch plättbarer Graph ist ein Graph der in einer Ebene mithilfe von Punkten bzw. Knoten und Kanten dargestellt werden kann, ohne dass sich zwei oder mehr Kanten schneiden (vgl. Quelle [9]). Jede Fläche des Graphen wird dabei durch mindestens drei verschiedene Kanten beschrieben, die den Rand dieser Fläche darstellen. Die Fläche um den Graphen herum, welche scheinbar unbegrenzt groß ist, wird äußeres Gebiet genannt.

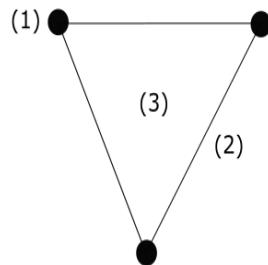


Abbildung 1: Schema eines planaren Graphen (Abbildung der Verfasser)

In Abbildung 1 wird ein solcher planarer Graph dargestellt. Der linke obere Knoten ist hier mit (1) bezeichnet, die rechte Kante mit (2) und die innere Fläche mit (3).

### 2.2 Doubly connected edge list

Um planare Graphen ohne Informationsverlust zu speichern werden in der Informatik Referenzen zwischen den einzelnen Bestandteilen des Graphen eingesetzt.

In der sogenannten „Doubly connected edge list“ (DCEL) erhält eine Kante, die aus einem Anfangsknoten und Endknoten besteht, jeweils eine

Vorgänger-, Nachfolger- und Zwillingskante zugewiesen und zudem die angrenzende Fläche. Durch die Darstellung einer Kante des Grundrisses durch zwei zueinander entgegengesetzt laufenden Kanten wird gewährt, dass sich Flächen keine Kante teilen, was zu Komplikationen in verschiedenen Operationen kommen kann.

Zudem wird für einen Knoten der DCEL eine ausgehende Kante und für eine Fläche eine anliegende Kante gespeichert(vgl. Quelle [2] und [3]).

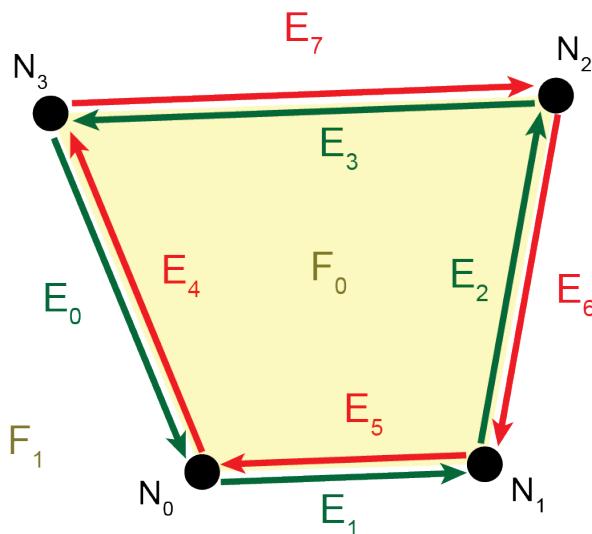


Abbildung 2: Schema einer DCEL (Abbildung der Verfasser)

Folgende Tabellen zeigen beispielhafte Referenzen von den Bestandteilen einer DCEL anhand Abbildung 2.

| Knoten | ausgehende Kante |
|--------|------------------|
| $N_0$  | $E_1$            |
| $N_2$  | $E_6$            |

| Fläche                | anliegende Kante |
|-----------------------|------------------|
| $F_0$                 | $E_3$            |
| $F_1$ (äußere Gebiet) | $E_7$            |

| Kante | Nachfolger | Vorgänger | Zwilling | Fläche |
|-------|------------|-----------|----------|--------|
| $E_0$ | $E_1$      | $E_3$     | $E_4$    | $F_0$  |
| $E_1$ | $E_2$      | $E_0$     | $E_5$    | $F_0$  |

## 2.3 Convex Hull

Die konvexe Hülle einer Punktmenge ist die kleinste konvexe Menge, in der die Punktmenge enthalten ist. Bezogen auf ein Polygon ist die konvexe Hülle folglich ein neues Polygon, welches so wenig wie möglich Punkte beinhaltet, damit alle Elemente des Ursprungspolygons innerhalb des Neuen liegen.

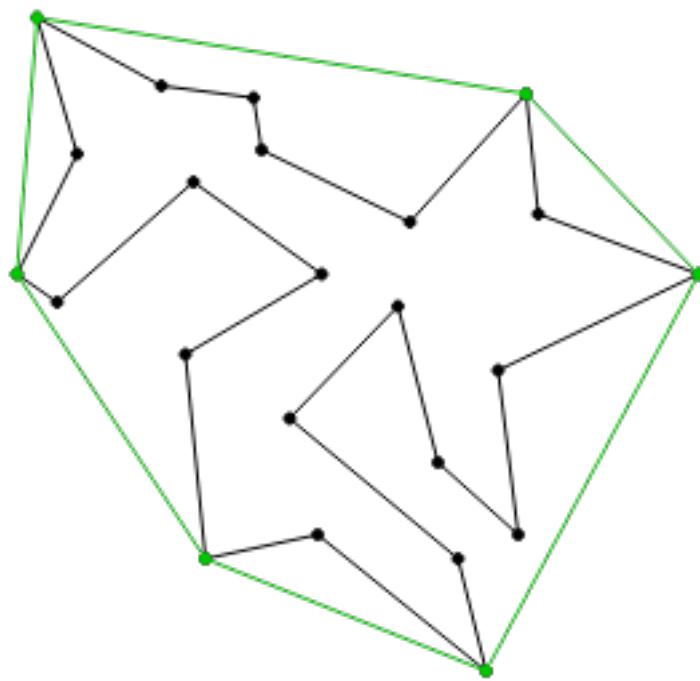


Abbildung 3: Beispiel einer konvexen Hülle

## 2.4 Oriented Minimum Bounding Box

Die Oriented Minimum Bounding(OMBB) Box bzw. orientiertes minimales Begrenzungsrechteck eines Polygons ist das Rechteck, welches das komplette Polygon umschließt und dabei den kleinstmöglichen Flächeninhalt be-

sitzt.

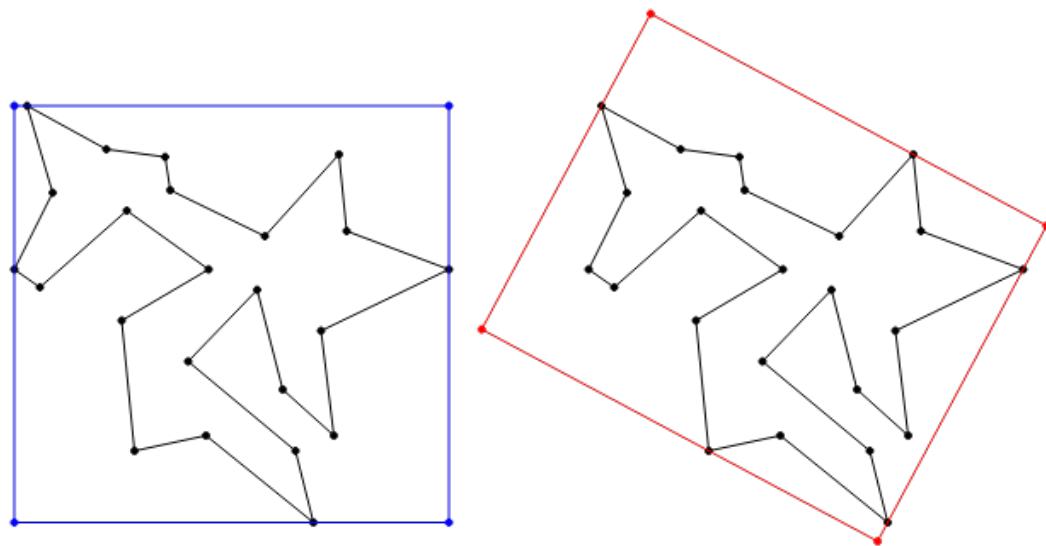


Abbildung 4: Eine mögliche Bounding Box(l.) im Vergleich zur OMBB(r.)

## 2.5 AutoCAD

AutoCAD ist ein grafischer Zeichnungseditor, welcher zum Erstellen von technischen Zeichnungen und dem Modellieren von Objekten verwendet wird (siehe Quelle [6]). AutoCAD verwendet dabei einfache Objekte wie Linien, Kreise und Bögen, um auf deren Grundlage kompliziertere Objekte zu erschaffen. Zu AutoCAD gehörig wurde das Dateiformat „.dxf“ entwickelt, welches als Industriestandard zum Austausch von CAD-Dateien dient.

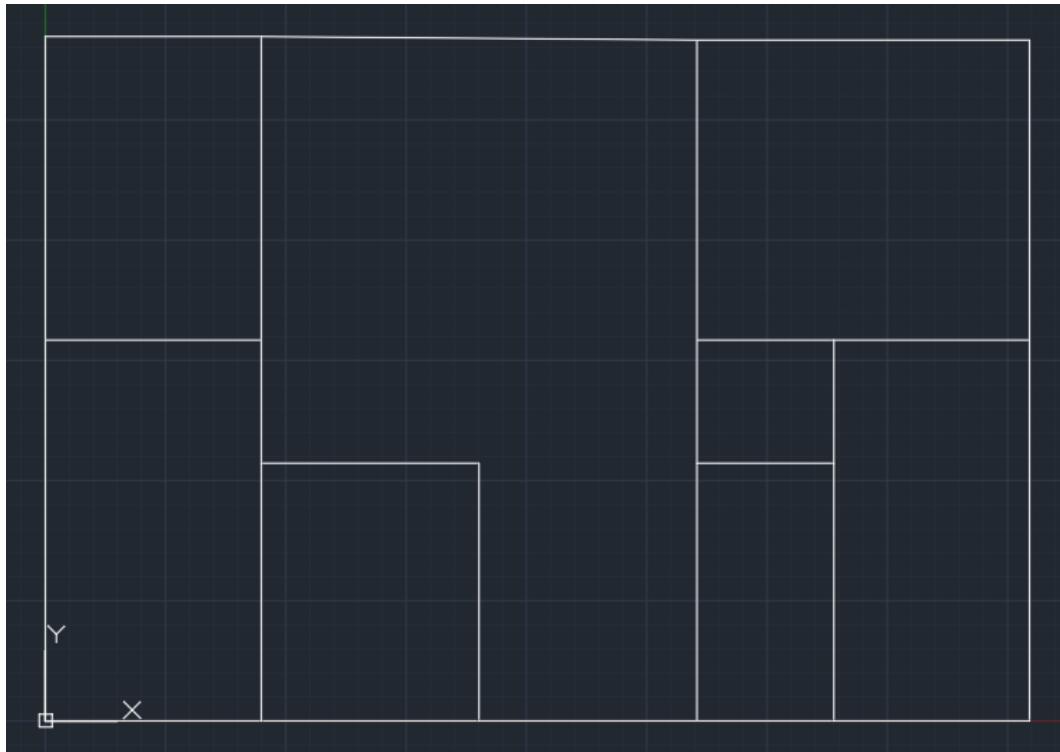


Abbildung 5: Grundriss aus AutoCAD (Screenshot der Verfasser)

Der Grundriss, welcher als Ausgangspunkt dieser Arbeit fungiert, wird in AutoCAD erstellt und dem zu erstellenden Programm in Form einer .dxf-Datei bereitgestellt. Diese Dateien dienen auch dem fertiggestellten Programm als Ausgangspunkt. Ein Beispiel für einen solchen Grundriss ist in Abbildung 5 zu sehen.

## 2.6 OpenSCAD

OpenSCAD ist eine kostenlos verfügbare CAD-Modellierungssoftware, welche aus einer textbasierten Beschreibungssprache 3D-Modelle erzeugt (siehe Quelle [4]). OpenSCAD bietet dabei verschiedene Vorteile während des Modellierungsvorganges. Hierzu gehören beispielsweise das farbige Hervorheben oder die Modularisierung zusammenhängender Objekte.

Die Modellierung von einfachen Basisobjekten in OpenSCAD erfolgt

durch das Verwenden von Schlüsselwörtern wie `cube()`, `sphere()` oder `cylinder()` und Parametern in Klammern. Diese Basisobjekte können anschließend durch Mengenoperationen wie Vereinigungen (`union()`), Differenzen (`difference()`) oder Überschneidungen (`intersection()`) und Transformationen wie Skalierungen (`scale()`), Rotationen (`rotate()`) oder Translationen (`translate()`) miteinander verknüpft und kombiniert werden, um neue Objekte nach eigenen Ansprüchen zu bilden.

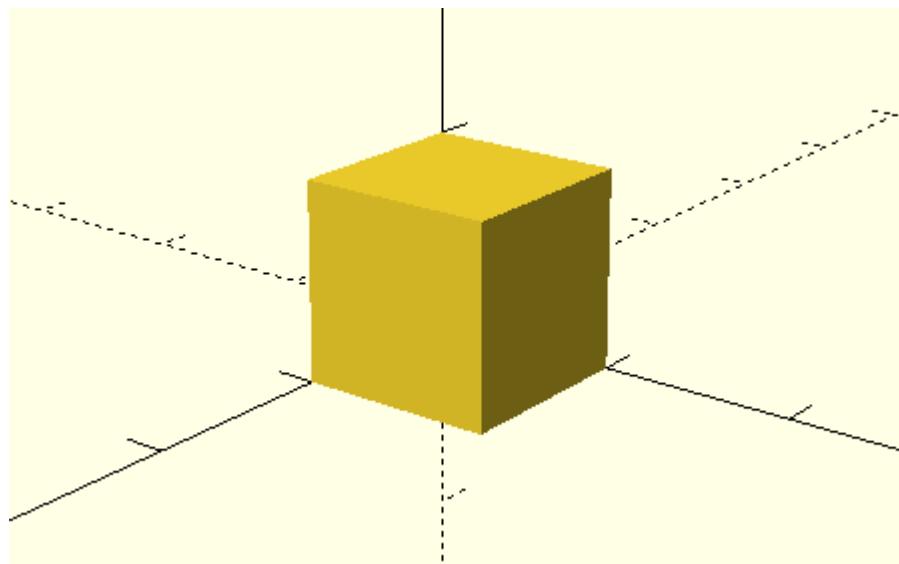


Abbildung 6: Eine Vereinigung zweier Würfel in OpenSCAD (Screenshot der Verfasser)

Neben solchen einfachen Objekten wird außerdem die Möglichkeit geboten, komplexere Objekte wie Polygone (`polygon()`) zu erstellen. Diese können dann ausgehend von einem zweidimensionalen Polygon in dreidimensionale Polygone umgewandelt werden (`linear_extrude()`), welche vor allem das Umwandeln von komplexen Formen in Objekte erleichtert.

Die Anweisungen, welche OpenSCAD zum Modellieren verwendet, werden in einfachen Textdateien im „.scad“-Format gespeichert. Die Simplizität dieser Textdateien erlaubt es, die aus dem zu entwickelnden Programm

erhaltenen Anweisungen in .scad-Dateien zu speichern, welche von OpenSCAD eingelesen, eingesehen und bearbeitet werden können.

Die Modelle, die so mit OpenSCAD erstellt wurden, können anschließend mit dem 3D-Drucker ausgedruckt werden. Dazu werden die Modelle in Dateien des „.stl“-Formats konvertiert. Dies geschieht entweder über die Oberfläche von OpenSCAD oder mittels einer Kommandozeilenanweisung.

## 2.7 3D-Drucker MakerBot Replicator™ 2

Der vorliegende 3D-Drucker ist das Modell Replicator™ 2 der Firma MakerBot. Dieser Drucker verfügt über eine höhenverstellbare Grundplatte, auf der das Filament<sup>1</sup> aufgetragen und das finale Objekt gedruckt wird und einen sogenannten „Extruder“, welcher die Funktion übernimmt, das zu druckende Filament zu erhitzen und mit einer konstanten Filamentbreite auf die Grundplatte bzw. das gedruckte Objekt aufzutragen. Mithilfe dieser zwei Hauptbestandteile wird schichtweise Filament aufgetragen, welches aushärtet und so nach und nach das Objekt bildet.

Die Höhe der Grundplatte wird während des Druckvorganges automatisch vom Drucker variiert und nach Abschluss des Druckens wieder auf den Ausgangszustand zurückgesetzt. Um die Beweglichkeit des Extruders zu garantieren, ist dieser auf drei Achsen befestigt, sodass drei Motoren ihn auf diesen Achsen verschieben können.

Abhängig vom Filament bzw. der Temperatur, bei der dieses aufgetragen wird, der Bewegungsgeschwindigkeit des Extruders und der Filamentstärke, die der Extruder aufträgt, lässt sich die gewünschte Druckqualität anpassen. Eine niedrige Qualität ist dabei in den meisten Fällen mit einer erheblich kürzeren Druckzeit verbunden.

Die Druckzeit wird außerdem von der eingestellten Ausfüllung von ge-

---

<sup>1</sup>Filament bezeichnet das Material, welches der 3D-Drucker zum Drucken verwendet.

schlossenen Objekten und dem Hinzufügen von Druckhilfen beeinflusst. So kann man Quader zum Beispiel nicht komplett mit Filament füllen lassen, sondern mit einem Bienenwabenmuster durchsetzen, sodass nur ein geringer Teil des Objektes ausgefüllt wird. Zusätzlich zu dem eigentlichen Druckergebnis wird unter jedes gedruckte Element ein dünner Untergrund gedruckt, welcher leicht von der Grundplatte und vom gedruckten Modell zu trennen ist und so eine Beschädigung beim Entfernen des Objekts vom Drucker verhindert. Außerdem werden bei Überhängen zusätzliche Stützen angebracht, um ein Absacken des noch nicht fest gewordenen Filaments zu verhindern. Indem so also ein stark verringelter Betrag an Filament aufgetragen werden muss, wird auch die Druckzeit drastisch reduziert.

Beim Drucken von Objekten ist neben Anpassungen zur Kontrolle der Druckqualität und Druckzeit außerdem zu beachten, dass die Grundplatte nur auf einer begrenzten Fläche bedruckbar ist. Entsprechend dieser möglichen Maße sollten also alle Objekte in ihrer Größe angepasst werden.

### 3. Vorgehen zur Problemlösung

#### 3.1 Einlesen des Grundrisses

##### 3.1.1 Funktionsweise der Bibliothek *kabeja*

Den Beginn der Verarbeitung markiert hierbei die Grundrissdatei, in welcher sämtliche Werte, welche im weiteren Verlauf des Programmes relevant werden, enthalten sind. Das Einlesen der Daten eines Grundrisses, wie in Abb. 5, erfolgt mit der Java-Bibliothek „kabeja“. Diese ermöglicht es, aus .dxf-Dateien alle DXF-Objekte eines bestimmten Typs zu erhalten und deren Werte in einer Liste zu speichern und später zu verarbeiten [5].

```
1 public static ArrayList<Line> getAutocadFile(String filePath)
2     throws ParseException {
3
4     ArrayList<Line> vcs = new ArrayList<>();
5
6     Parser parser = ParserBuilder.createDefaultParser();
7
8     parser.parse(filePath, DXFParser.DEFAULT_ENCODING);
9
10    DXFDocument doc = parser.getDocument();
11
12    List lst = doc.getDXFLayer("0").getDXFEntities(
13        DXFConstants.ENTITY_TYPE_LINE);
14
15    for (int index = 0; index < lst.size(); index++) {
16
17        DXFLine dxfline = (DXFLine) lst.get(index);
18
19        Line v = new Line(
20
21            new Vector(round2(dxfline.getStartPoint().getX()),
22
23                round2(dxfline.getStartPoint().getY())),
24
25            new Vector(round2(dxfline.getEndPoint().getX()),
26
27                round2(dxfline.getEndPoint(). getY())));
28
29        vcs.add(v);
30
31    }
32
33
34    return vcs;
35 }
```

Codeauschnitt 1: DXF File Parser

**Überdenken, ob Code hier wirklich notwendig**

In dieser Anwendung wird eine Funktion der Klasse „DXFReader“ verwendet, welche den Pfad zur .dxf-Datei als Parameter übergeben bekommt. Aus dieser Datei werden alle DXF-Objekte, die mit dem Typen `DXFLine` übereinstimmen, in einer Liste zurückgegeben. Die Koordinaten der Start- und Endpunkte der DXFLines in dieser Liste werden anschließend in eine Liste von Linien übertragen, welche im weiteren Programmablauf unter anderem bei der Umwandlung des Graphen in die DCEL Verwendung findet.

### 3.1.2 Funktionsweise der GUI

Die GUI setzt sich zusammen aus einem `JFrame`, in dem zwei `JTextField`, ein `FileChooser`, ein `StartButton` und ein `ShowResultButton` platziert sind.

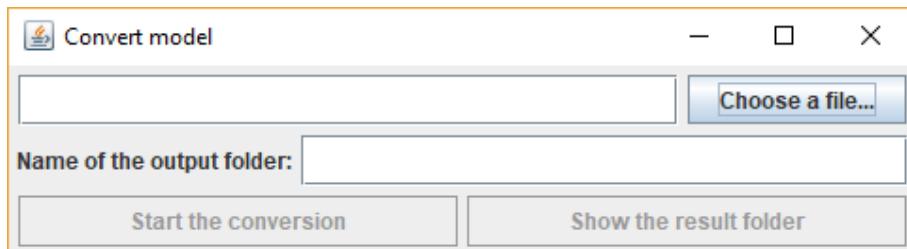


Abbildung 7: Ausgangszustand der GUI (Screenshot der Verfasser)

Der Nutzer kann im Ausgangszustand über den `FileChooser` einen `JFileChooser`-Dialog öffnen, mit dem er die Datei, die er umwandeln möchte, auswählen kann. Sobald er dann eine Datei ausgewählt hat, wird der Dateipfad zu dieser Datei zusätzlich im `JTextField` angezeigt. In diesem `JFileChooser` ist außerdem ein `FileFilter` implementiert, der dem Nutzer lediglich .dxf-Dateien anzeigt. Alternativ zum `JFileChooser`-Dialog kann der Nutzer auch direkt in das links positionierte `JTextField` den Dateipfad eingeben. Im `JTextField` unterhalb gibt der Nutzer dann den Ordnernamen ein, in dem die umgewandelten Dateien ausgegeben werden.

Nachdem der Nutzer eine Datei ausgewählt und einen Namen für den Ergebnisordner hat, wird nun der StartButton aktiviert und er kann den Konvertierungsprozess starten.

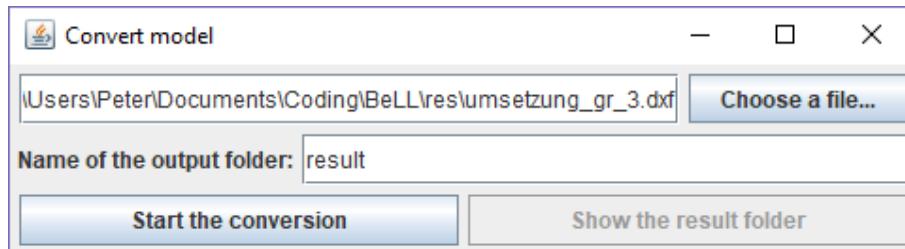


Abbildung 8: GUI mit aktiviertem StartButton (Screenshot der Verfasser)

Beim Klicken des StartButton wird überprüft, ob das Programm die ausführbare Datei von OpenSCAD unter C:\\Program Files\\OpenSCAD\\ finden kann. Diese Datei wird benötigt, um die Umwandlung der .scad-Dateien in .stl-Dateien zu ermöglichen. Sollte diese dort nicht gefunden werden, wird der Nutzer mit einem Dialog darauf hingewiesen.

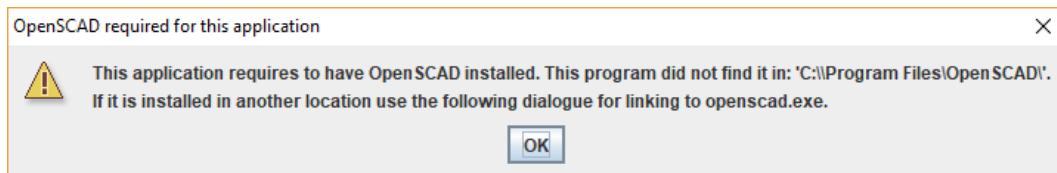


Abbildung 9: Dialog zur Warnung des Nutzers (Screenshot der Verfasser)

Anschließend hat der Nutzer die Möglichkeit, den Pfad zur ausführbaren Datei mit einem weiteren JFileChooser-Dialog anzugeben.

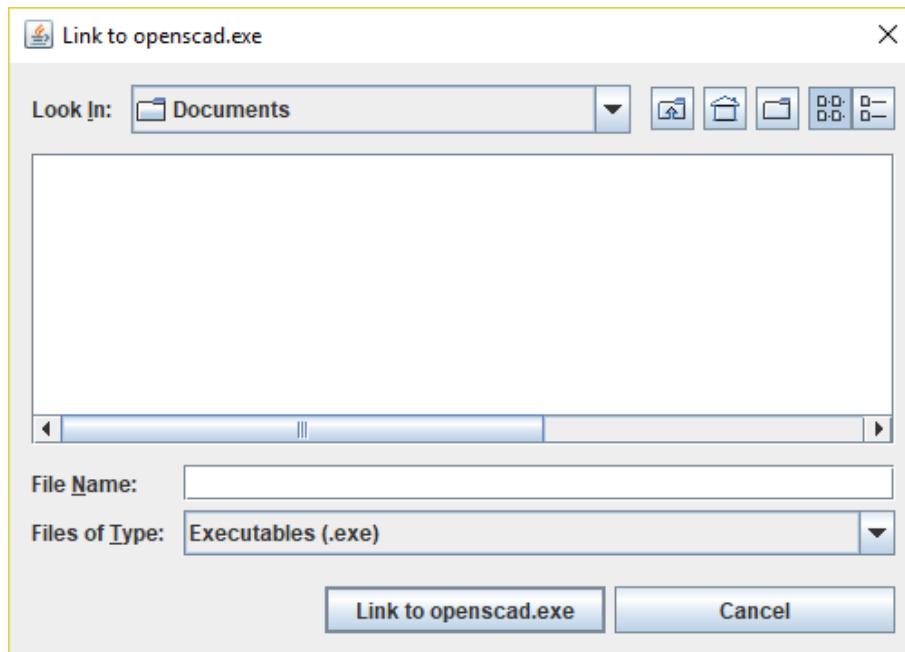


Abbildung 10: Dialog zum Verlinken der openscad.exe (Screenshot der Verfasser)

Sobald dieser Vorgang beendet ist, bleiben dem Nutzer nun die Möglichkeiten, den Ordner mit den Ausgabedateien anzuzeigen oder einen anderen Grundriss umzuwandeln.

## 3.2 Erstellen der Doubly-connected Edge List

Die zweidimensionalen Operationen der Anwendung werden von der Graph-Klasse durchgeführt. Übergibt man dieser eine Liste aus Linien, welche jeweils aus einem Start- und Endortsvektor bestehen, wird auf diesen basierend eine DCEL berechnet. Dafür werden die drei graphischen Elemente der Typen `Edge(Kante)`, `Node(Knoten)` und `Face(Fläche)` gespeichert.

### 3.2.1 Verarbeitung der Linien

Beginnend werden gleichzeitig die Knoten und Kanten der zu generierenden DCEL erstellt.

### 3.2.2 Vector-to-Node Konvertierung

Aus einem Ortsvektor kann die Funktion `createNode()` einen kongruenten Knoten mit gleichen Ursprungskoordinaten, aber ohne Referenz auf eine anliegende Kante, erstellen. Des Weiteren wird, falls ein Knoten zu einem abgefragten Punkt schon generiert wurde, dieser zurückgeben.

```
1  private Node createNode(Vector p) {
2      for (Node n : nodes) {
3          if (n.getOrigin().equals(p)) {
4              return n;
5          }
6      }
7      nodes.add(new Node(p));
8      return (nodes.get(nodes.size() - 1));
9  }
```

Codeausschnitt 2: `createNode()` Funktion

### 3.2.3 Line-to-Edge Konvertierung

Die `processData` Funktion greift auf `createNode()` zu und erstellt die Liste aus Kanten mit den jeweiligen Start- und Endknoten. Auch hier gibt es noch keine abgespeicherten Zusammenhänge zwischen den einzelnen Kanten.

```
1     private void processData(ArrayList<Line> ls) {  
2         for (Line l : ls) {  
3             edges.add(new Edge(createNode(l.getP1()),  
4                                 createNode(l.getP2())));  
5         }  
6     }
```

Codeausschnitt 3: Line-to-Edge Konvertierung

Basierend auf dieser Grundlage müssen alle folgenden Kalkulationen der Graph-Klasse durchgeführt werden.

### 3.2.4 Zwillingskantengenerierung

Durch Vertauschen der Start- und Endknoten wird nun für jede existente Kante eine entgegengesetzt-laufende komplementäre „Zwillingskante“ gebildet. In der Anwendung wird so die Liste der Kanten um ihre Größe erweitert. Direkt nach dem Hinzufügen der Zwillingskante wird jeweils eine Referenz erstellt, welche beide Zwillinge miteinander verknüpft. Durch die Zwillingskanten werden folgende Operationen in der DCEL vereinfacht, da jede Fläche nun von einer eindeutigen Menge an Kanten begrenzt ist und ein Umlaufsinn dieser festgestellt werden kann.

### 3.2.5 Nachfolger- und Vorgängerermittlung

Für die Erstellung der Nachfolger- und Vorgängerreferenzen zwischen den Kanten, werden alle ausgehenden Kanten  $E$  eines Knotens  $N_i$  betrachtet. Anschließend sortiert die Anwendung diese mittels der `angle()`-Methode

anhand des Winkels. Dabei wird die jeweilige Kante  $E_i$  in einen Vektor, welcher zwischen die beiden Kantenknoten gespannt werden kann, konvertiert, damit die `angle()`-Funktion aufrufbar ist. Diese gibt einen eindeutigen Winkel des Vektors zur x-Achse im Intervall  $(-\pi, \pi]$  aus. Aus den angeordneten Kanten  $E$  lassen sich unter Beachtung des mathematisch positivem Umlaufsinnes der Kanten an den Flächen nun folgende Beziehungen ableiten:

1. Die Zwillingskante von  $E_{i+1}$  ist der Vorgänger von  $E_i$
2.  $E_{i-1}$  ist der Nachfolger der Zwillingskante von  $E_i$

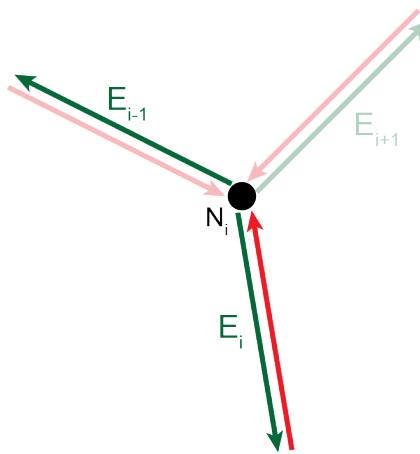
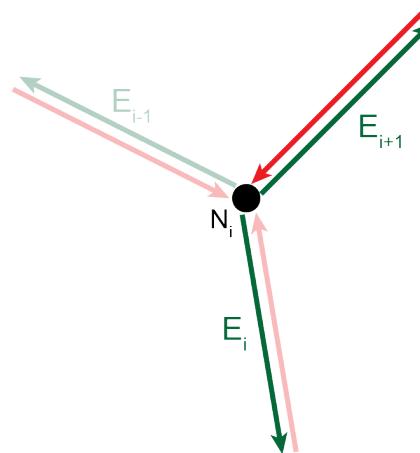


Abbildung 11: Veranschaulichung 1. und 2.

Falls  $i-1$  bzw.  $i+1$  die Indices der Menge  $E$  mit  $n$  Elementen überschreiten, wird anstattdessen  $E_n$  bzw.  $E_0$  gewählt.

Der aufgeführte Algorithmus wird für alle Knoten  $N$  der DCEL fortgeführt,

sodass alle Referenzen zwischen den Kanten fertiggestellt werden. r jeden Knoten eine sortierte ArrayList aus Kanten enthält,

```
1     for (ArrayList<Edge> e : nodeEdges) {  
2         for (int i = 0; i < e.size(); i++) {  
3             e.get(i).setPrev(e.get((i + 1) % e.size()).  
4                           getTwin());  
5             e.get(i).getTwin().setNext(e.get(Math.  
6                             floorMod((i - 1), e.size())));  
7         }  
8     }
```

Codeausschnitt 4: Zuweisung der Beziehungen im Programm; `nodeEdges` ist eine zweidimensionale ArrayList

### 3.2.6 Flächenerstellung

Durch das Vorliegen der DCEL Kanten können alle Flächen des Graphen erschlossen werden. Dabei wird bei dem Element  $E_0$  der Kantenliste begonnen und solange der Nachfolger über die Verknüpfung ermittelt, bis die Ursprungskante wieder erreicht wurde. Die ermittelte Fläche  $F_0$  besitzt als anliegende Kante so die Kante  $E_0$ . Alle in der Fläche enthaltenen Kanten werden bei der fortlaufenden Rechnung ignoriert, sodass die alle anderen Flächen  $F$  analog errechnet werden können. Das äußere Gebiet des planaren Graphen, also in der DCEL der Umriss, kann durch den Drehsinn der Kanten dieser Fläche festgestellt werden, denn dieser ist als einziger mathematisch negativ. In der Anwendung wird dafür die Gaußsche Trapezformel verwendet, welche bei einem umgekehrten Drehsinn einen negativen Flächeninhalt ausgibt.

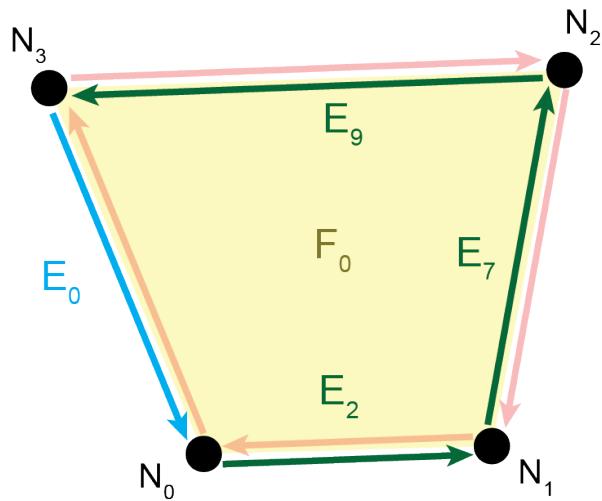


Abbildung 12: Flächenberechnung. Dargestellt: Anliegende Kante(Cyan), Kanten(grün) und Fläche(gelb)

### 3.2.7 Vervollständigung der Knoten

Die letzte nötige Referenz ist die einer anliegenden Kante eines Knotens. Dafür wird für die Knoten  $N$  eine Kante ermittelt, die ihren Ursprung in dem jeweiligen Knoten hat. Die DCEL ist mit diesem Schritt fertiggestellt und bildet die Grundlage für alle folgenden dreidimensionalen Berechnungen.

### 3.3 Aufbau der Einzelbauteile

Die zweidimensionalen Strukturen der DCEL werden genutzt, um das 3D-Model, welches aus SCAD-Objekten besteht, zu erstellen. Dabei werden Modifikationen an allen Bauteilen vorgenommen, folgende Kriterien erfüllt sind:

Das 3D-Modell soll...

1. ...einfach aufgebaut werden können.
2. ...einen angemessenen Einblick in die Immobilie gewährleisten.
3. ...stabil sein, auch nachdem einzelne Elemente entfernt wurden.

Für die Erfüllung dieser Kriterien wurden die drei Strukturen Wand, Eckpfeiler und Grundplatte entworfen. Weiterführend sind diese so modifiziert, dass sie ein oben beschriebenes 3D-Konstrukt darstellen.

#### 3.3.1 Eckpfeiler

Die Eckpfeiler des Modells symbolisieren die Knoten des Grundrisses. Sie sind das Schlüsselement des implementierten Stecksystems, welches für Stabilität und Variabilität sorgt. Damit anliegende Wände und Grundplatten in einen Eckpfeiler greifen können, sind zwei verschiedene Arten von Steckern entworfen wurden. Diese sind an zwei voneinander unabhängigen Abschnitten „CornerPin“ und „CornerCylinder“ angebracht, welche zusammengefügt das gesamte 3D-Objekt verkörpern.

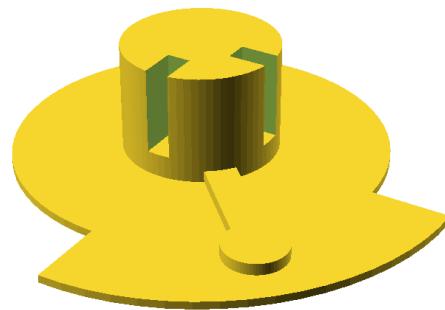


Abbildung 13: Ein Eckpfeiler (Screenshot der Verfasser)

### Eckzylinder

Der Eckzylinder, im Code `CornerCylinder` stellt den oberen Teil des Eckpfeilers dar. Dieser besteht aus einem zylindrischen Grundbauteil mit Vertiefungen. Diese entstehen, indem Schnittmengen zwischen dem Zylinder und in Richtung der ausgehenden Kanten gedrehten Quader gebildet werden. Nun können Wände in diese eingeschoben werden.

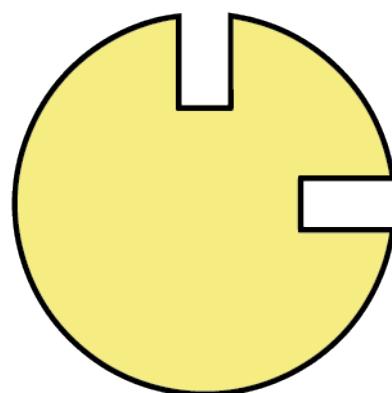


Abbildung 14: Querschnitt eines Eckzylinders mit zwei angrenzenden Wänden

## Eckpin

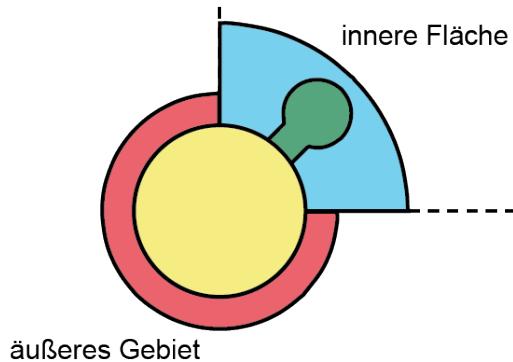


Abbildung 15: Draufsicht auf den unteren Abschnitt eines Eckpfeilers mit zwei anliegenden Flächen

Die Summe aller Objekte des Typs `CornerPin` repräsentieren den unteren Abschnitt des Eckpfeilers, welcher die Verbindungen der Grundplatten mit diesem gewährleistet. Hierfür besteht ein einzelner Eckpin aus einem Zylinder, einer Ausstülpung seitens diesem und einem Zylindersegment. Das Grundgerüst wird durch den Basiszylinder des Eckpfeilers festgelegt, welcher auch beim Eckzylinder vorkommt. Für die Ausstülpung gilt, dass sie aus einem Quader mit angrenzendem Zylinder aufgebaut ist. Die Länge des Quaders ist dabei abhängig von dem Winkel zwischen den beiden Wänden, welche die Grundplatte an dem Eckpfeiler begrenzen. Er wird stets so lang berechnet, dass die Ausstülpung (siehe Abbildung, grün dargestellt) einen definierten Abstand von den Wänden einnimmt (siehe 3.3.4). Für die untere Abgrenzung des Pinkopfes (siehe Abbildung 15, hellblau) berechnet die Anwendung eine Schnittmenge zwischen der angrenzenden Fläche und einem flachen Zylinder.

Falls ein Eckpin für das äußere Gebiet berechnet werden soll, wird anstatt der oben beschriebenen Elementen eine Differenzmenge zwischen der Umrandungsfläche des Grundrisses und dem flachen Zylinder gebildet. Dies resultiert in ein Zylindersegment, wie es in Abbildung 15 (rot) zu sehen ist.

Ein Eckpfeiler hat so einen stabilisierten unteren Abschnitt, auch außerhalb des Grundrisses.

### 3.3.2 Wandstücke

Eine Wand verkörpert eine Kante der DCEL. Für die Erstellung dieser wird ein Quader mit der Länge der entsprechenden Kante gebildet und nach dieser rotiert. Jeweils ein Eckzylinder wird anschließend an den Start- bzw. Endknoten des Quaders verschoben, sodass zwischen diesem und den beiden Eckzylindern eine Differenzmenge gebildet werden kann. Das Resultat ist eine Wand, welche abschließend in einen Eckpfeiler eingesetzt werden kann.

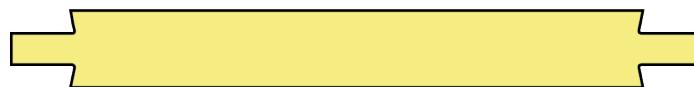


Abbildung 16: Querschnitt einer Wand

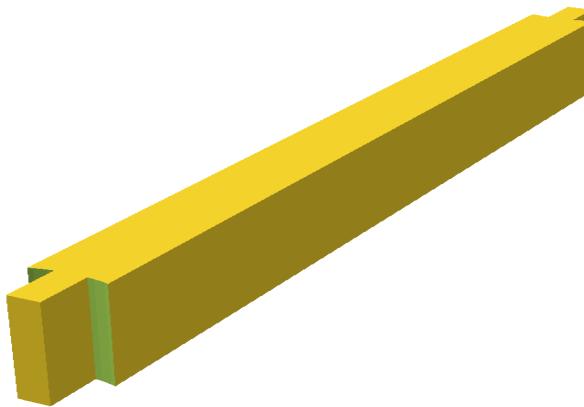


Abbildung 17: Ein Wand in der OpenSCAD Darstellung (Screenshot der Verfasser)

### 3.3.3 Grundplatten

Eine Grundplatte wird anhand einer Fläche des Grundrisses konstruiert. Sie sind aufgebaut aus einem geradem Prisma mit einer Polygon-Grundfläche, welches eingestülpte Steckmechanismen an der Unterseite besitzt.

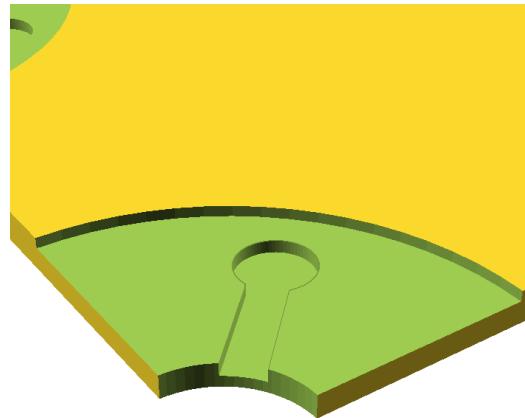


Abbildung 18: Stecker einer Grundplatte

Für die Berechnung des 3D-Objekts, wird an jedem Knoten der Fläche wird ein Eckpin gebildet und von dem polygonalen Prisma abgezogen, sodass genannte Einkerbungen durch eine Differenzmenge entstehen. Jede Grundplatte, die an das äußere Gebiet angrenzt wird zudem an dieser Seite um die halbe Wanddicke vergrößert, damit ein abgeschlossener Eindruck des gesamten 3D-Modells entsteht.(Siehe Abbildung 18)

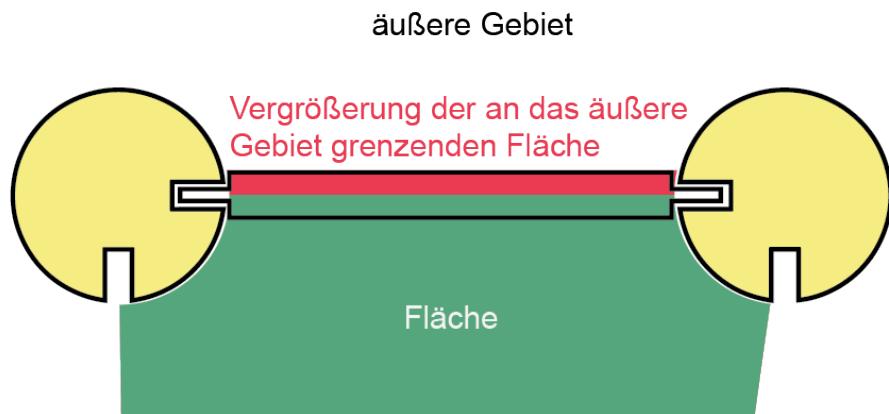


Abbildung 19: Vergrößerung der äußeren Grundplatten(vereinfacht)

### 3.3.4 Zuweisen von Berechnungskonstanten

Die `Params`-Klasse dient als Speichermöglichkeit für alle nötigen Konstanten, welche für die Umrechnung des Grundrisses in das 3D-Modell benötigt werden. Jedem dreidimensionalen Objekt benötigt eine Instanz einer solchen Klasse, um den OpenSCAD Code zu generieren. In der Anwendung wird von der Klasse `ScadProcessor` die angegebenen Parameter an alle Elemente des Grundrisses weitergegeben, damit verhindert wird, dass unterschiedliche Komponenten desselben Grundrisses verschiedene Konstanten zur Verfügung gestellt bekommen. Sie werden bei Beginn des Programms in der `Main`-Klasse gesetzt und können über eine Instanz der `Params`-Klasse aufgerufen werden.

```

1  public void setParams(double E, ...){
2      e = E;
3      // ...
4 }
```

Codeausschnitt 5: Die `setParams()`-Funktion zum Setzen der Parameter

Das Abrufen der Parameter erfolgt dann mittels der entsprechenden `get()`-Funktionen der `Params`-Klasse, welche für alle Parameter vorhanden sind. Der Aufbau der `get()`-Funktionen folgt dem generellen

Aufbau des nachfolgenden Codebeispiels, jedoch werden die Parameterbezeichnungen jeweils entsprechend ersetzt:

```
1 public double getE() {  
2     return e;  
3 }
```

Codeauschnitt 6: Die `get()`-Funktion für den Parameter  $e$

Die nötigen Parameter stellen dabei Gleitkommazahlen(`double`) dar, die verschiedene Teile des Grundrisses beeinflussen. Theoretisch können sie alle variiert werden.

### $\epsilon$ /Epsilon

Der Parameter Epsilon( $\epsilon$ ) bezeichnet einen Abstand zwischen 3D-Objekten, der benötigt wird, um einem möglichen Fehler des 3D-Druckers entgegenzuwirken. Dieser kann z.B. bewirken, dass Steckmechanismen nicht ineinander passen. Deswegen tritt der Abstand  $\epsilon$  generell dort auf, wo zwei Objekte aufeinander oder ineinander geschoben werden müssen. In Folgenden Teilen des 3D-Modells tritt Epsilon auf:

1. Bei dem Vorkommen von einem positiven und einem negativem Steckern (Wand/Eckpfeiler und Grundplatte/Eckpfeiler) ist der negative gleichmäßig um  $\epsilon$  vergrößert, damit ein abschließendes Stecken ermöglicht wird.
2. Wenn zwei Grundplatten aufeinandertreffen, wird die sich berührenden Seiten um  $0.5 * \epsilon$  entgegengesetzt zueinander verschoben zur Einhaltung eines Abstandes von  $\epsilon$ .
3. Die Höhe einer Wand wird um  $\epsilon$  verringert, damit die Oberseite des gesamten 3D-Objekts eine glatte Oberfläche besitzt.

Das im Programm definierte Epsilon wurde mittels einem Experiment(siehe Abbildung 19) ermittelt. Dafür wurde ein Stecker und Einstülpungen, die um ein verschiedenes  $\epsilon$  vergrößert wurden, gedruckt, sodass getestet wurden

konnte, welches Epsilon für den MakerBot Replicator™ 2 am geeignetsten ist. Ausgewertet wurde daraus ein Wert von 0.25mm.

Der  $\epsilon$ -Parameter ist deswegen ein druckerspezifischer Parameter.



Abbildung 20: Versuch zur Ermittlung eines passenden Epsilon-Wertes

## Weitere Konstanten

Die restlichen Parameter werden in der folgenden Tabelle gelistet.

| Konstante                | Bedeutung   | Wert in mm |
|--------------------------|---|------------|
| CornerRadius             | Radius des Basiszylin-<br>ders der Eckpfeiler     | 10         |
| PinMinLength             | minimale Länge des<br>Pinkopfes am Eckpin         | 2          |
| PinPWidth                | Breite des Quaders<br>des Pinkopfes               | 4          |
| PinPRadius               | Radius des Zylinders<br>am Pinkopf                | 4          |
| PinDistance              | minimaler Abstand<br>zwischen Pinkopf und<br>Wand | 2          |
| Height                   | Höhe des 3D-Modells                               | 75         |
| pinHeight                | Höhe des Pinkopfes                                | 4          |
| BasePlateHeight          | Höhe der Grundplat-<br>ten                        | 4          |
| basePlatePinCircleHeight | Höhe der Kreisseg-<br>menten an den Eck-<br>pins  | 1          |
| wallWidth                | Breite der Wände                                  | 6          |
| maxPrintWidth            | Breite der Druckfläche                            | 185        |
| maxPrintHeight           | Höhe der Druckfläche                              | 153        |

## 3.4 Druckvorgang

Nachdem die einzelnen Bestandteile des Modells berechnet wurden, können diese gedruckt werden. Hierbei ist allerdings zu bedenken, dass alle einzelnen Bauteile nicht in einem Druckvorgang gedruckt werden können, da die Grundplatte des 3D-Druckers zu klein ist.

In unserem Fall wies diese eine Länge von 28,5 cm und eine Breite von 15,2 cm auf (siehe Quelle [8]).

### 3.4.1 Minimierung der Druckvorgänge

Um die Dimensionen der Grundplatte effektiv auszunutzen, werden einzelne Elemente des Modells nach der Größe sortiert und dann solange nebeneinander angeordnet, bis die Breite der Druckfläche mit dem nächsten Element überschritten würde. Anschließend wird eine neue Reihe eröffnet, in der nun neue Elemente platziert werden. Dieser Prozess wird solange fortgeführt, bis die Länge der Druckfläche überschritten werden würde.

Dann wird eine neue Gruppierung von Objekten bzw. `Union` erstellt, in welcher der Algorithmus weiterläuft. Das Resultat zeigt eine Liste von Objekten, die eine platzeffiziente Anordnung dieser repräsentiert. Jedes einzelne Element in der Liste kann mit einem Durchgang gedruckt werden.

### 3.4.2 Ermittlung der Größen

Das Einzige, was die Strukturen Wand- und Eckteil, sowie Grundplatte in diesem Algorithmus voneinander unterscheidet, ist die Größe, welche unterschiedlich bestimmt wird. Da die Anordnung ein zweidimensionales Problem ist, wird die Höhe der Objekte nicht berücksichtigt.

## Wände

Die Wandelemente werden durch eine Kante dargestellt und so modifiziert, dass sie in die Eckpfeiler gesteckt werden kann. Die Wände sind durch den notwendigen Abstand zwischen den Objekten immer kürzer als die zugehörige Kante. Die Breite ermittelt sich durch die definierte `wallWidth` der übergebenen `Params`-Klasse.

## Eckpfeiler

Für die Größenberechnung der Eckpfeiler wird der längste Pin, welcher an dem jeweiligen Objekt anliegt, festgestellt. Die Länge dieses Pins stellt dann die Seitenlänge eines Quadrats dar, welche die Ausmaße des Eckpfeilers angibt.

## Grundplatten

Um die optimale Größe der Grundplatten zu berechnen, muss die OMBB der Grundplatte berechnet werden. Dafür wird die OMBB der Grundfläche ermittelt und anschließend vergrößert, damit Modifikationen, die an der dieser durchgeführt wird, berücksichtigt werden.

**Bestimmung der Konvexen Hülle** Die Berechnung der OMBB wird über diesen Zwischenschritt realisiert, da nach dem von Freeman und Shapira bewiesenen Satz, eine Seite des minimalen Begrenzungsrechtecks kollinear mit einer der konvexen Hülle sein muss.

Für die Ermittlung wird der „Gift Wrapping Algorithm“ herangezogen. Bei diesem Algorithmus wird zuerst der Punkt mit der kleinsten Ordinate als Startpunkt  $P_0$  festgelegt. Gibt es mehrere dieser Punkte, wird der Punkt mit der kleinsten Abszisse gewählt. Von  $P_0$  ausgehend wird eine Gerade durch einen beliebigen Punkt  $P$  des Polygons gelegt und anschließend überprüft, ob es einen Punkt  $S$  gibt, der links der Geraden liegt. Im Programm geschieht das durch die Berechnung des Winkels zwischen den Vektoren  $\overrightarrow{P_0P}$

und  $\overrightarrow{P_0S}$  mithilfe der `Vector.angleTo(Vector v)` Methode. Ist dieser Winkel kleiner als  $180^\circ$ , liegt  $S$  links von der Gerade durch  $P_0$  und  $P$ .  $S$  wird folglich als neuer Punkt  $P$  gesetzt und die Berechnung fortgesetzt. Der Vorgang wird solange durchgeführt bis alle Punkte überprüft wurden und ein nächster Punkt  $P_1$  der konvexen Hülle feststeht. Der Algorithmus wird nun fortgeführt mit  $P_1$  als Ausgangspunkt, solange bis der Anfangspunkt  $P_0$  wieder erreicht wurde, so dass die konvexe Hülle „geschlossen“ ist.

**Bestimmung der OMBB** Nun wird die konvexe Hülle fortlaufend rotiert, so dass eine Seite parallel zur x-Achse ist. Dadurch kann man leicht das Begrenzungsrechteck durch Feststellen der Extremwerte des rotierten Polygons ermitteln. Der Flächeninhalt des Rechtecks aus den Punkten  $A(x_{min}, y_{min})B(x_{min}, y_{max})C(y_{max}, x_{max})D(x_{max}, y_{min})$  wird gespeichert. Der Algorithmus wiederholt diese Abfolge für jede Gerade, bis alle Seiten der konvexen Hülle behandelt wurden. Als Ergebnis erhält das Programm die Breite, Höhe und den nötigen Winkel der OMBB. Diese Werte werden dann in der Platzierung genutzt.

## 3.5 Ausgabe der Resultate

### 3.5.1 Erstellung des Ausgabeordners

Alle ausgegebenen Dateien des Programms werden in einem Ordner zusammengefasst. Dieser Ordner wird im gleichen Verzeichnis erstellt, in dem auch das Programm ausgeführt wird. Als Name des Verzeichnisses wird der Name genutzt, den der Nutzer in der GUI eingegeben hat.

```
1  private static void createDirs(String folderName) {  
2      try {  
3          Files.createDirectories(Paths.get("./" +  
4              folderName + "/scad"));  
5          Files.createDirectories(Paths.get("./" +  
6              folderName + "/stl"));  
7      } catch (IOException e) {  
8          e.printStackTrace();  
9      }  
10 }
```

Codeausschnitt 7: Erstellung der Unterordner

Im Unterordner „scad“ dieses Ordners finden sich dann sämtliche .scad-Dateien, im Unterordner „stl“ alle .stl-Dateien.

### 3.5.2 OpenSCAD Java Interface

Für die erleichterte Erstellung von OpenScad Objekten wurde ein Java Interface `ScadObject` erstellt, welches alle für das Projekt wichtigen Befehle enthält. Die Methode `toString()` stellt in den Klassen des Interfaces die Übergabe des OpenSCAD Befehlsstrings dar. So kann man z.B. mit der Klasse `Cube` einen Quader mit gegebener Länge, Höhe und Breite erstellen, der dann wie folgt mit `Cube.toString()` in einen String konvertiert wird: `cube([Länge, Breite, Höhe]);`

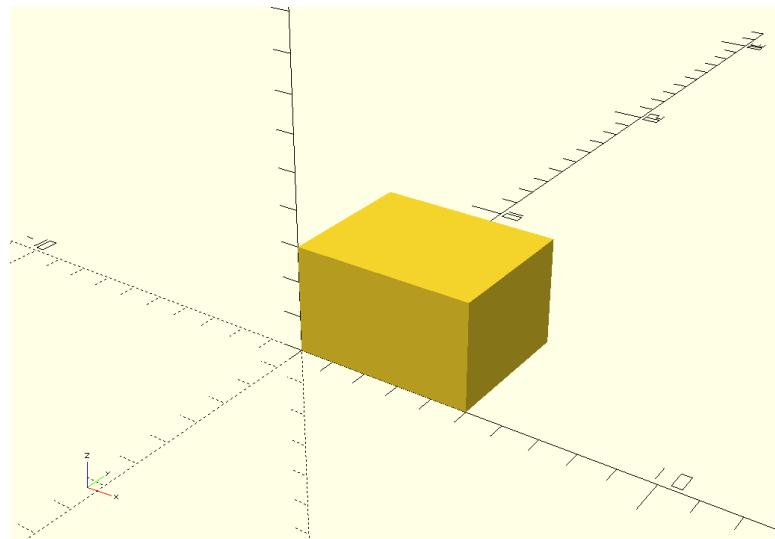


Abbildung 21: Resultat der Eingabe: `new Cube(3, 4, 5).toString()`  
(Screenshot der Verfasser)

### 3.5.3 STL Konvertierung

Um die ermittelten .scad-Dateien im .stl-Format auszugeben, wird die Kommandozeilenfunktionalität von OpenSCAD verwendet. Die verwendete Anweisung setzt sich hierbei aus dem Pfad zur `openscad.exe`, dem Parameter `-o` und zwei weiteren Parametern zusammen. Die beiden weiteren Parameter stehen zum einen für den Namen der Ausgabedatei und zum anderen für den Namen der Eingabedatei. Eine vollständige Anweisung könnte wie folgt aussehen: `C:\Program Files\OpenSCAD\openscad.exe -o \stl\walls1.stl \scad\walls1.scad`

Hierzu werden zunächst alle Dateien im „scad“-Unterordner mit der Endung .scad durch die Klasse `SCADFinder` ermittelt und in einem Feld ausgegeben.

```
1  public static File[] findFiles(String folderName) {  
2      File dir = new File(".\\" + folderName + "\\scad\\");  
3  
4      return dir.listFiles(new FilenameFilter() {  
5          public boolean accept(File dir, String
```

```
        filename)
6            { return filename.endsWith(".scad"); }
7        } );
8    }
```

Codeauschnitt 8: Ausgabe aller .scad-Dateien aus einem Ordner

Diese Dateinamen verwendet dann die Klasse `STLConverter` um mittels eines `ProcessBuilder` die Kommandozeilenanweisungen auszuführen und die .stl-Dateien im „stl“-Unterordner zu erstellen.

```
1   public static void convert (String fileName, String
2       folderName) throws InterruptedException {
3
4       Process p;
5
6       ProcessBuilder b;
7
8       try {
9
10           b = new ProcessBuilder(/*Anweisung*/);
11
12           p = b.start();
13
14           p.waitFor();
15
16       } catch (IOException e) {
17
18           e.printStackTrace();
19
20       }
21
22   }
```

Codeauschnitt 9: Ausführung der Kommandozeilenanweisungen

## 4. Ausblick

Im aktuellen Entwicklungsstand ist es nur möglich, alle Bauteile einzeln auszudrucken. Dies erhöht jedoch den Filamentverbrauch des 3D-Druckers um ein Vielfaches, weshalb eine Kombination mehrerer Bauteile für einen Druckvorgang zwecks der Reduktion des verwendeten Filaments für den Druck unterstützende Elemente als sinnvoll anzusehen ist. Dafür bietet sich beispielsweise ein gemeinsamer Druck von Wandteilen oder Eckpfeilern anbieten, da diese Objekte weitestgehend ähnliche Ausmaße besitzen und somit eine recht effektive Kombination möglich ist. Außerdem liegen momentan lediglich Bauteile vor, welche nur auf einer Druckplatte fester Größe gedruckt werden können. Sollte das zu druckende Objekt größer als die Druckplatte sein, muss es zum Drucken skaliert werden, was jedoch unbedingt vermieden werden soll, da dadurch die Verhältnisse der Stecker zueinander verändert werden und so ein sachgemäßer Aufbau verhindert wird. Um diesen Umstand zu verhindern, soll es in der weiteren Entwicklung möglich sein, überdimensionierte Bauteile weiter in kleinere Untereinheiten zu teilen und so eine Wahrung des Maßstabs zu garantieren. Hierfür muss jedoch ein weiteres Stecksystem, sowie weitere Logik zur Umsetzung und Umwandlung der alten Bauteile konzipiert und implementiert werden. Als ferne Zukunftskonzeption, die an den Rahmen der Besonderen Lernleistung anschließt, lässt sich die Umsetzung von 3D-Modellen festmachen. Hierzu zählen kompliziertere Wände mit Schrägen, Fenstern oder Verstrebungen und Dachgestelle, welche als Abschluss auf dem Modell angebracht werden können. Die Komplexität der Aufgabenstellung wird dadurch aber um ein Vielfaches gesteigert, weshalb diese Problematik kein Bestandteil der Besonderen Lernleistung sein wird.

# Internetquellenverzeichnis

- [1] [de.wikipedia.org/wiki/Gau%C3%9Fsche\\_Trapezformel](https://de.wikipedia.org/wiki/Gau%C3%9Fsche_Trapezformel)  
(Stand: 21.03.2017, 12:00 Uhr)
- [2] [cs.sfu.ca/~binay/813.2011/DCEL.pdf](https://cs.sfu.ca/~binay/813.2011/DCEL.pdf)  
(adaptiert von M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf: Computational Geometry: Algorithms and Applications,  
Stand: 25.04.2017, 12:00 Uhr)
- [3] [en.wikipedia.org/wiki/Doubly\\_connected\\_edge\\_list](https://en.wikipedia.org/wiki/Doubly_connected_edge_list)  
(Stand: 25.04.2017, 12:00 Uhr)
- [4] [en.wikibooks.org/wiki/OpenSCAD\\_User\\_Manual](https://en.wikibooks.org/wiki/OpenSCAD_User_Manual)  
(Stand: 25.04.2017, 12:00 Uhr)
- [5] [kabeja.sourceforge.net/](https://kabeja.sourceforge.net/)  
(Stand: 12.10.2017, 10:00 Uhr)
- [6] <https://en.wikipedia.org/wiki/AutoCAD>  
(Stand: 05.06.2016, 18:00)
- [7] <https://en.wikipedia.org/wiki/OpenSCAD>  
(Stand: 05.06.2017, 18:00)
- [8] [https://eu.makerbot.com/fileadmin/Inhalte/Support/Manuals/German\\_UserManual\\_V.4\\_Replicator2.pdf](https://eu.makerbot.com/fileadmin/Inhalte/Support/Manuals/German_UserManual_V.4_Replicator2.pdf)  
(Stand: 06.06.2017, 09:00)
- [9] [https://de.wikipedia.org/wiki/Planarer\\_Graph](https://de.wikipedia.org/wiki/Planarer_Graph)  
(Stand: 07.11.2017, 16:00 Uhr)

# Abbildungsverzeichnis

|    |   |    |
|----|---|----|
| 1  | Schema eines planaren Graphen (Abbildung der Verfasser) . . . . .                               | 5  |
| 2  | Schema einer DCEL (Abbildung der Verfasser) . . . . .   | 6  |
| 3  | Beispiel einer konvexen Hülle . . . . .   | 7  |
| 4  | Eine mögliche Bounding Box(l.) im Vergleich zur OMBB(r.) . . . . .                              | 8  |
| 5  | Grundriss aus AutoCAD (Screenshot der Verfasser) . . . . .                                      | 9  |
| 6  | Eine Vereinigung zweier Würfel in OpenSCAD (Screenshot der Verfasser) . . . . .                 | 10 |
| 7  | Ausgangszustand der GUI (Screenshot der Verfasser) . . . . .                                    | 14 |
| 8  | GUI mit aktiviertem StartButton (Screenshot der Verfasser)                                      | 15 |
| 9  | Dialog zur Warnung des Nutzers (Screenshot der Verfasser) . .                                   | 15 |
| 10 | Dialog zum Verlinken der openscad.exe (Screenshot der Verfasser) . . . . .                      | 16 |
| 11 | Veranschaulichung 1. und 2. . . . .   | 19 |
| 12 | Flächenberechnung. Dargestellt: Anliegende Kante(Cyan), Kanten(grün) und Fläche(gelb) . . . . . | 21 |
| 13 | Ein Eckpfeiler (Screenshot der Verfasser) . . . . .   | 23 |
| 14 | Querschnitt eines Eckzylinders mit zwei angrenzenden Wänden                                     | 23 |
| 15 | Draufsicht auf den unteren Abschnitt eines Eckpfeilers mit zwei anliegenden Flächen . . . . .   | 24 |
| 16 | Querschnitt einer Wand . . . . .  | 25 |
| 17 | Ein Wand in der OpenSCAD Darstellung (Screenshot der Verfasser) . . . . .                       | 25 |
| 18 | Stecker einer Grundplatte . . . . .   | 26 |
| 19 | Vergrößerung der äußeren Grundplatten(vereinfacht) . . . . .                                    | 27 |

|    |  |    |
|----|--|----|
| 20 | Versuch zur Ermittlung eines passenden Epsilon-Wertes . . .                                | 29 |
| 21 | Resultat der Eingabe: new Cube(3, 4, 5).toString()<br>(Screenshot der Verfasser) . . . . . | 35 |

## Dank an die Betreuer

Wir möchten hiermit sowohl unserem internen Betreuer, als auch den externen Betreuern der Fakultät für Mathematik und Informatik der Universität Leipzig danken.

Herrn Rai-Ming Knospe als unseren internen Betreuer des Wilhelm-Ostwald-Gymnasiums verdanken wir die Beratung bezüglich Inhalt und Ausarbeitung dieser Besonderen Lernleistung. Außerdem fungierte er als zentraler schulinterner Ansprechpartner und lieferte uns Hinweise beim Überarbeiten der Besonderen Lernleistung.

Die Idee für das Thema lieferte uns Herr Prof. Dr. Gerik Scheuermann der Fakultät für Mathematik und Informatik der Universität Leipzig. Er ermöglichte uns ebenso das Arbeiten mit dem 3D-Drucker und stellte uns die notwendigen Materialien bereit. Weiterhin möchten wir einen besonderen Dank an Herrn Tom Liebmann richten, der uns während unserer Arbeitszeit an der Universität Leipzig betreute und mit umfangreichen Rat sowie Ideen zur Problemlösung zur Seite stand. Er unterstützte uns dabei maßgeblich in unserer Arbeits- und Herangehensweise zur Umsetzung des Problems und gab uns Hinweise zur Lösung mathematischer Operationen.

## Selbstständigkeitserklärung

### **Johann Bartel**

Ich versichere, dass ich die Arbeit selbstständig angefertigt, nur die angegebenen Hilfsmittel benutzt und alle Stellen, die dem Wortlaut und dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht habe.

Mit der schulinternen Verwendung der Arbeit bin ich einverstanden.

Leipzig, den 22.12.2017

Johann Bartel

### **Peter Oehme**

Ich versichere, dass ich die Arbeit selbstständig angefertigt, nur die angegebenen Hilfsmittel benutzt und alle Stellen, die dem Wortlaut und dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht habe.

Mit der schulinternen Verwendung der Arbeit bin ich einverstanden.

Leipzig, den 22.12.2017

Peter Oehme

---

## Arbeitsteilung

Wir versichern hiermit, dass die Besondere Lernleistung von uns zu gleichen Teilen erarbeitet wurde. Die Aufteilung erfolgte so, dass Johann Bartel die „, die „, ... und die „ übernahm. Peter Oehme fertigte die „, die „, ... und die „ an. Hierbei wurde darauf abgezielt, dass jeder von uns etwa die Hälfte der Arbeit erarbeitet hat.

Leipzig, den 22.12.2017

Johann Bartel

Peter Oehme