

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“

42 Seiten, 24 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.

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.

Inhaltsverzeichnis

1 Einleitung	3
2 Wissenschaftliche Grundlagen	4
2.1 Planare Graphen	4
2.2 Doubly Connected Edge List	4
2.3 Oriented Minimum Bounding Box	6
2.4 AutoCAD	7
2.5 OpenSCAD	7
2.6 3D-Drucker MakerBot Replicator™ 2	9
3 Vorgehen zur Problemlösung	11
3.1 Einlesen des Grundrisses	11
3.1.1 Funktionsweise der Benutzeroberfläche	11
3.1.2 Funktionsweise der Bibliothek <i>kabeja</i>	14
3.2 Erstellen der Doubly Connected Edge List	15
3.2.1 Verarbeitung der Linien	15
3.2.2 Zwillingskantengenerierung	16
3.2.3 Nachfolger- und Vorgängerermittlung	16
3.2.4 Flächenerstellung	18
3.2.5 Vervollständigung der Knoten	19
3.3 Aufbau der Einzelbauteile	20
3.3.1 OpenSCAD Java Interface	20
3.3.2 Eckpfeiler	21
3.3.3 Wandstücke	24
3.3.4 Grundplatten	25
3.3.5 Zuweisen von Berechnungskonstanten	26
3.4 Druckvorgang	29
3.4.1 Minimierung der Druckvorgänge	29
3.4.2 Ermittlung der Größen	31

3.5 Ausgabe der Resultate	34
3.5.1 Erstellung des Ausgabeordners	34
3.5.2 STL Konvertierung	35
4 Ausblick	37
5 Internetquellenverzeichnis	38
6 Abbildungsverzeichnis	39
7 Arbeitsteilung	41
8 Selbstständigkeitserklärung	42

1. Einleitung

3D-Druck gehört zu den beliebtesten technischen Neuerungen der letzten Jahre. Nicht nur im privaten Einsatz, sondern auch im professionellen Bereich finden 3D-gedruckte Objekte immer mehr Anwendung. Die anschauliche Darstellung bestimmter Elemente ermöglicht auch unerfahrenen Nutzern Zugang zu komplexen Objekten.

Naheliegend ist es demzufolge, diese Technik zur Visualisierung von Gebäuden zu verwenden. Auf Basis des Grundrisses, einer einfachen Form der Darstellung eines Gebäudes, sollte es möglich sein, ein Modell zu erstellen. Dieses soll in kleine Grundeinheiten unterteilt sein, die über ein Stecksystem zusammengesetzt werden können.

Die Umsetzung dieses Problems ist das Ziel dieser Arbeit. Zur Lösung wird ein Programm in Java erstellt, welches die Umwandlung des Grundrisses in ein Modell übernimmt.

2. Wissenschaftliche Grundlagen

Für die Erarbeitung der Anwendung ist Vorwissen über graphische Strukturen nötig. Die folgenden Abschnitte dienen der Erarbeitung dieses Vorwissens, sowie der Erklärung von verwendeten Programmen.

2.1 Planare Graphen

Der Grundriss wird mithilfe eines planaren Graphen beschrieben. Ein planarer, auch plättbarer Graph, ist ein Graph, der in einer Ebene mithilfe von Knoten und Kanten dargestellt werden kann, ohne dass sich zwei oder mehr Kanten schneiden (vgl. Quelle [1], siehe Abb. 1). Jede Fläche des Graphen wird durch mindestens drei verschiedene Kanten beschrieben, die den Rand der Fläche markieren. Die Fläche um den Graphen herum, welche unbegrenzt groß ist, wird äußeres Gebiet genannt.

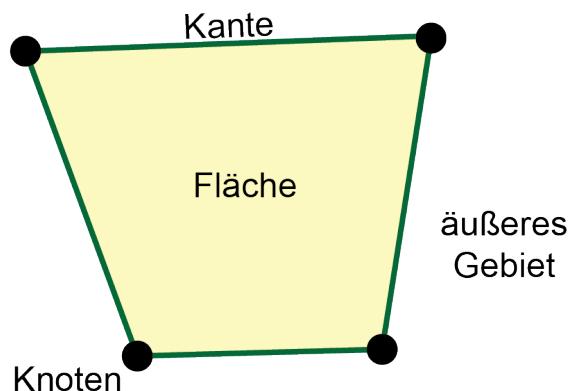


Abbildung 1: Schema eines planaren Graphen (Abbildung der Verfasser)

2.2 Doubly Connected Edge List

Um planare Graphen ohne Informationsverlust zu speichern wird in der Informatik eine „Doubly Connected Edge List“ (DCEL) genutzt. In einer DCEL wird jeder Kante, die aus einem Anfangsknoten und Endknoten

besteht, jeweils eine Vorgänger-, Nachfolger-, Zwillingskante und die angrenzende Fläche zugewiesen. Durch die Darstellung einer Linie des Grundrisses durch zwei zueinander entgegengesetzt laufenden Kanten wird gewährt, dass keine zwei Flächen an einer Kante anliegen. Zudem wird für einen Knoten der DCEL eine ausgehende Kante und für eine Fläche eine anliegende Kante gespeichert (vgl. Quelle [2] und [3]).

Die Kanten der Flächen verlaufen im mathematisch positiven Drehsinn. Als einzige Ausnahme gilt das äußere Gebiet, welches einen mathematisch negativen Drehsinn besitzt.

In Abb. 2 ist eine solche DCEL zu sehen. Die nachfolgenden Tabellen verdeutlichen die Referenzen der Bestandteile der DCEL in Abb. 2.

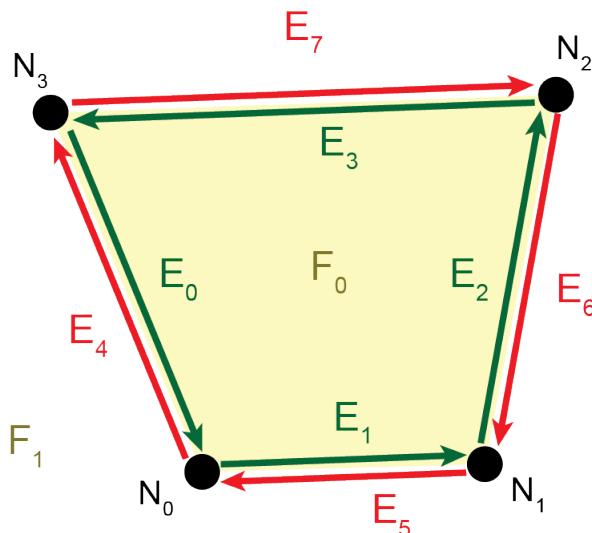


Abbildung 2: Schema einer DCEL (Abbildung der Verfasser)

Knoten	ausgehende Kante
N_0	E_1
N_2	E_6

Fläche	anliegende Kante
F_0	E_3
F_1 (äußeres 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 Oriented Minimum Bounding Box

Die Oriented Minimum Bounding Box (OMBB) bzw. das orientierte minimale Begrenzungsrechteck einer Fläche ist das Rechteck, welches das komplette Polygon umschließt und dabei den kleinstmöglichen Flächeninhalt besitzt (siehe Abb. 3). Sie wird über den „Convex Hull“ (konvexe Hülle) eines Polygons berechnet. Die „Convex Hull“ ist die kleinste konvexe Menge, in der die Punkte der Fläche enthalten sind (siehe Abb. 3). Mindestens eine Seite des minimalen Begrenzungsrechtecks ist kollinear zu einer Seite der konvexen Hülle (vgl. Quelle [4]).

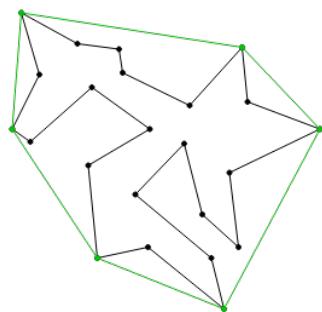


Abbildung 3: Beispiel einer konvexen Hülle (vgl. Quelle [4])

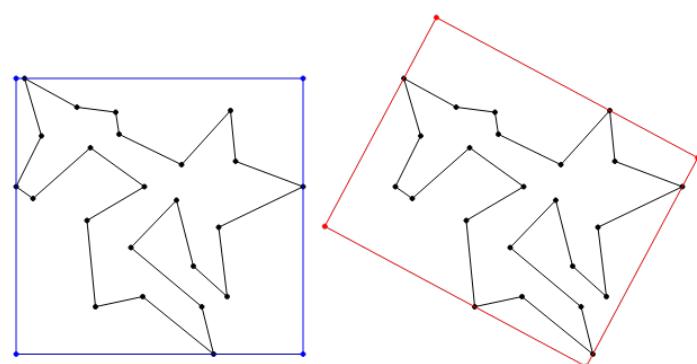


Abbildung 4: Eine mögliche Bounding Box (l.) im Vergleich zur OMBB (r.) (vgl. Quelle [4])

2.4 AutoCAD

AutoCAD ist ein grafischer Zeichnungseditor, welcher zum Erstellen von technischen Zeichnungen und dem Modellieren von Objekten verwendet wird (vgl. Quelle [5]). AutoCAD verwendet dabei einfache Objekte wie Linien, Kreise und Bögen, um auf deren Grundlage kompliziertere Objekte zu bilden. Zu AutoCAD gehörig wurde das Dateiformat „.dxf“ entwickelt, welches als Industriestandard zum Austausch von CAD-Dateien dient.

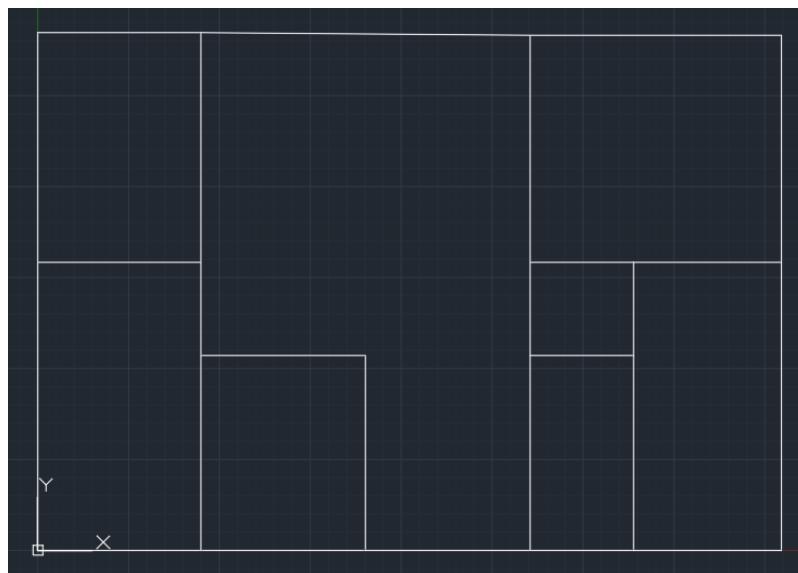


Abbildung 5: Grundriss aus AutoCAD (Screenshot der Verfasser)

Der Grundriss, der später als Ausgangspunkt fungiert, wird in AutoCAD erstellt und liegt als .dxf-Datei vor. Ein Beispiel für einen solchen Grundriss ist in Abb. 5 zu sehen.

2.5 OpenSCAD

OpenSCAD ist eine kostenlos verfügbare CAD-Modellierungssoftware, welche aus einer textbasierten Beschreibungssprache 3D-Modelle erzeugt (vgl. Quelle [6]). OpenSCAD bietet dabei verschiedene Vorteile während des Modellierungsvorganges, beispielsweise das farbige Hervorheben oder

die Modularisierung zusammenhängender Objekte.

Die Modellierung von einfachen Basisobjekten in OpenSCAD erfolgt durch das Verwenden von Anweisungen wie `cube()`, `sphere()` oder `cylinder()` und Parametern in Klammern. Diese Basisobjekte können anschließend durch Mengenoperationen wie Vereinigungen (`union()`), Differenzen (`difference()`) oder Schnittmengen (`intersection()`) und Transformationen wie Skalierungen (`scale()`), Rotationen (`rotate()`) oder Translationen (`translate()`) miteinander verknüpft und kombiniert werden, um neue Objekte nach eigenen Ansprüchen zu erzeugen (siehe Abb. 6).

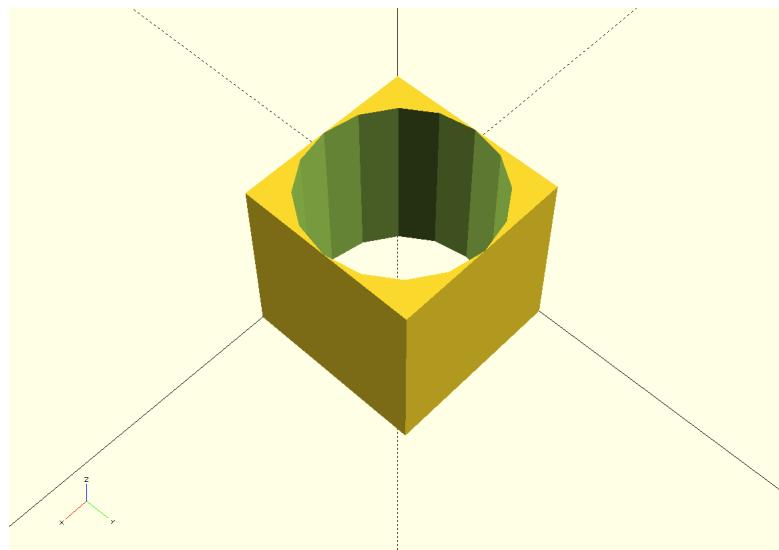


Abbildung 6: Differenzmenge zwischen einem Würfel und einem Zylinder in OpenSCAD (Screenshot der Verfasser)

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 Prismen umgewandelt werden (`linear_extrude()`).

Die Anweisungen, welche OpenSCAD zum Modellieren verwendet, werden in einfachen Textdateien im `.scad`-Format gespeichert. Die Einfachheit dieser Textdateien erlaubt es, die erhaltenen Anweisungen ohne Umwandlung

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

Die Modelle, die mit OpenSCAD erstellt wurden, können anschließend mit einem 3D-Drucker ausgedruckt werden. Dazu werden die Modelle in Dateien des .stl-Formats umgewandelt. Die Konvertierung geschieht entweder über die Benutzeroberfläche von OpenSCAD oder mittels einer Kommandozeilenanweisung.

2.6 3D-Drucker MakerBot Replicator™ 2

Der vorliegende 3D-Drucker ist vom Modell Replicator™ 2 der Firma MakerBot. Dieser Drucker verfügt über eine höhenverstellbare Grundfläche, auf der das Filament¹ aufgetragen und das finale Objekt gedruckt wird. Der Extruder erhitzt dazu das zu druckende Filament und trägt dieses auf die Grundfläche auf. Mithilfe dieser zwei Hauptbestandteile wird schichtweise Filament aufgetragen, welches aushärtet und so das Objekt bildet.

Die Höhe der Grundfläche wird während des Druckvorganges automatisch vom Drucker variiert und nach Abschluss des Drucks wieder auf den Ausgangszustand zurückgesetzt. Um die Beweglichkeit des Extruders zu garantieren, ist dieser auf drei Achsen befestigt, sodass 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 mit einer kürzeren Druckzeit verbunden.

Die Druckzeit wird außerdem von der eingestellten Ausfüllung von geschlossenen Objekten und dem Hinzufügen von Druckhilfen beeinflusst. So kann man Objekte beispielsweise nicht komplett mit Filament füllen

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

lassen, sondern mit einem Bienenwabenmuster durchsetzen, sodass nur ein geringer Teil des Objektes ausgefüllt, aber dennoch Stabilität gewährleistet wird. Indem so also ein stark verringelter Betrag an Filament aufgetragen werden muss, wird auch die Druckzeit drastisch reduziert.

Zu dem eigentlichen Druckergebnis wird unter jedes gedruckte Element eine dünne Schicht 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 („Supports“) gedruckt, um ein Absacken des noch nicht ausgehärteten Filaments zu verhindern.

Beim Drucken von Objekten ist neben Anpassungen zur Kontrolle der Druckqualität und Druckzeit zu beachten, dass die Grundfläche begrenzt ist. Entsprechend dieser vorgegebenen Maße sollten alle Objekte in ihrer Größe angepasst werden.

3. Vorgehen zur Problemlösung

Die Anwendung konvertiert eine .dxf Grundrissdatei in .scad und .stl Dateien, die zum Drucken benutzt werden können. Dafür sind mehrere Zwischenschritte nötig.

3.1 Einlesen des Grundrisses

Den Beginn der Verarbeitung markiert die Grundrissdatei, in welcher sämtliche Werte, die im weiteren Verlauf des Programmes relevant werden, enthalten sind.

3.1.1 Funktionsweise der Benutzeroberfläche

Die Benutzeroberfläche (GUI) setzt sich zusammen aus einem JFrame, in dem zwei JTextField, ein FileChooserButton, ein StartButton und ein ShowResultButton platziert sind. Der Ausgangszustand der Benutzeroberfläche (GUI) ist in Abb. 7 zu sehen. Der StartButton, der ShowResultButton und der FileChooserbutton erben jeweils von der JButton-Klasse.

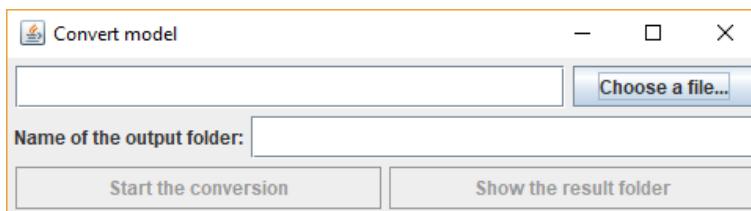


Abbildung 7: Ausgangszustand der Benutzeroberfläche (Screenshot der Verfasser)

Der Nutzer kann im Ausgangszustand über den FileChooserButton 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. Alternativ zum JFileChooser-Dialog kann der Nutzer auch direkt in das links positionierte JTextField den Dateipfad eingeben. Im JFileChooser ist außerdem ein FileFilter implementiert, der dem Nutzer lediglich .dxf-Dateien anzeigt. Der FileFilter verhindert dabei, dass der Nutzer einen anderen Dateitypen auswählt. Im JTextField darunter gibt der Nutzer den Ordnernamen ein, in dem die umgewandelten Dateien ausgegeben werden.

Der StartButton überprüft nach der Interaktion mit dem JTextField, ob eine Datei ausgewählt und ein Ordnername eingegeben wurde. Sollten beide dieser Bedingungen erfüllt sein, wird der StartButton aktiviert und der Nutzer kann die Konvertierung starten (siehe Abb. 8).

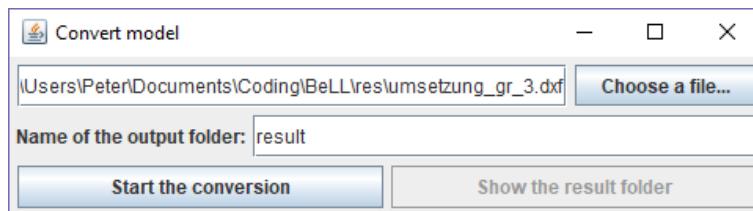


Abbildung 8: Benutzeroberfläche mit aktiviertem StartButton (Screenshot der Verfasser)

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 (siehe Abb. 9).

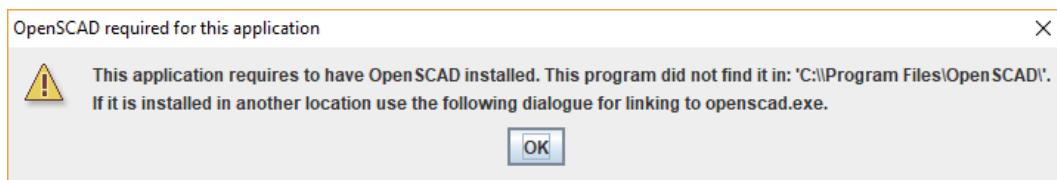


Abbildung 9: Dialog zur Warnung des Nutzers (Screenshot der Verfasser)

Nach dem Schließen des Dialogs öffnet sich nun ein weiterer JFileChooser (siehe Abb. 10). Mit diesem hat der Nutzer die Möglichkeit, den Pfad zur ausführbaren Datei anzugeben.

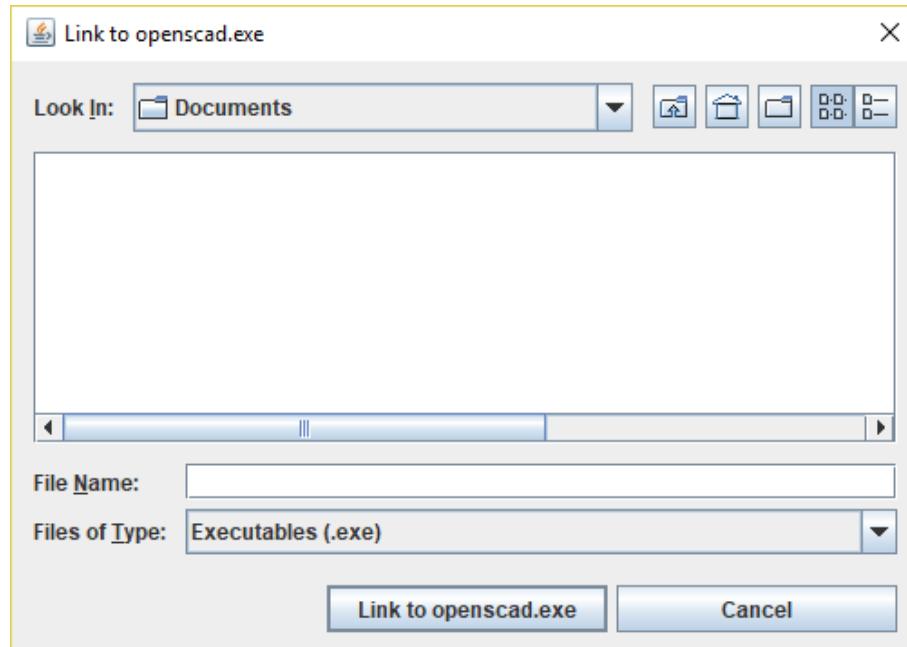


Abbildung 10: Dialog zum Verlinken der openscad.exe (Screenshot der Verfasser)

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

3.1.2 Funktionsweise der Bibliothek *kabeja*

Das Einlesen der Daten eines Grundrisses erfolgt mit der Java-Bibliothek „kabeja“. Diese ermöglicht es, aus .dxf-Dateien alle DXF-Objekte eines bestimmten Typs zu erhalten, deren Werte in einer Liste zu speichern und später zu verarbeiten.

```
1  public static ArrayList<Line> getAutocadFile(String
2      filePath) throws ParseException {
3          ArrayList<Line> vcs = new ArrayList<>();
4          /*Ermitteln des DXFDocuments doc*/
5          List lst = doc.getDXFLayer("0").getDXFEntities(/*
6              ...*/);
7          for (int index = 0; index < lst.size(); index++) {
8              DXFLine dxfline = (DXFLine) lst.get(index);
9              Line v = new Line(<Vektor des Anfangspunktes
10                  >, <Vektor des Endpunktes>);
11              vcs.add(v);
12          }
13      return vcs;
14 }
```

Codeausschnitt 1: DXF File Parser

Aus der .dxf-Datei deren Pfad der Klasse „DXFReader“ übergeben wird, werden alle DXF-Objekte, die mit dem Typen `DXFLine` übereinstimmen, in einer Liste zurückgegeben (vgl. Codeausschnitt 1). Die Koordinaten der Start- und Endpunkte der DXFLines in dieser Liste werden anschließend in eine Liste von Linien übertragen, welche unter anderem bei der Umwandlung des Graphen in die DCEL Verwendung findet.

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. Ausgegangen wird hierbei von

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 zurückgegeben (vgl. Codeausschnitt 2).

```

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

Line-to-Edge Konvertierung

Die `processData()` Funktion greift auf `createNode()` zu und erstellt die Liste aus Kanten mit den jeweiligen Start- und Endknoten (vgl. Codeausschnitt 3). 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         }

```

Codeausschnitt 3: Line-to-Edge Konvertierung

3.2.2 Zwillingskantengenerierung

Durch Vertauschen der Start- und Endknoten wird 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 nachfolgende Operationen in der DCEL vereinfacht, da jede Fläche nun von einer eindeutigen Menge an Kanten begrenzt ist und ein Umlaufsinn dieser festgestellt werden kann.

3.2.3 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 aufsteigend nach den Winkeln, mittels der `angle()`-Methode. 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. Die `angle()`-Funktion benutzt die Methode `Math.atan2(y, x)`, welche anhand der Koordinaten eines Vektors einen eindeutigen Winkel zur x-Achse im Intervall $(-\pi, \pi]$ ausgibt. Mithilfe der geordneten 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 (siehe Abb. 11)
2. E_{i-1} ist der Nachfolger der Zwillingskante von E_i (siehe Abb. 11)

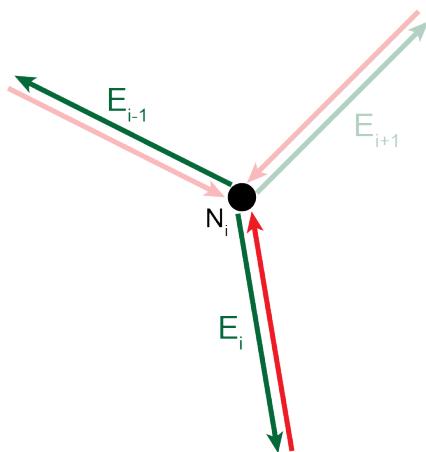
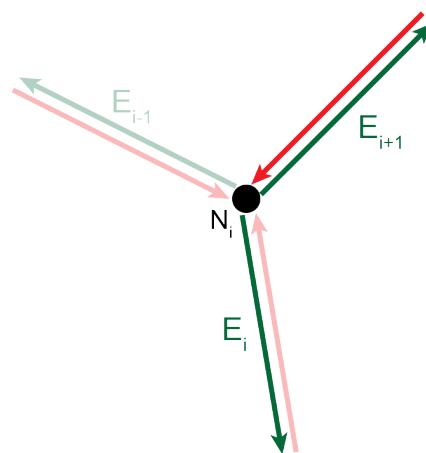


Abbildung 11: Veranschaulichung 1. und 2. (Abbildung der Verfasser)

Falls $i - 1$ bzw. $i + 1$ die Indices der Menge E mit n Elementen überschreiten, wird stattdessen 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 (vgl. Codeausschnitt 4).

```

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 zwischen den Kanten im Programm

3.2.4 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 die Kante E_0 . Alle in der Fläche enthaltenen Kanten werden bei der fortlaufenden Rechnung ignoriert, sodass alle anderen Flächen F analog errechnet werden können (siehe Abb. 12).

Das äußere Gebiet des planaren Graphen, in der DCEL die Umrandungsfläche, kann durch den Drehsinn der Kanten dieser Fläche festgestellt werden, welcher als einziger mathematisch negativ ist. In der Anwendung wird die Gaußsche Trapezformel verwendet, welche bei einem umgekehrten Drehsinn einen negativen Flächeninhalt ausgibt und demzufolge das äußere Gebiet markiert.

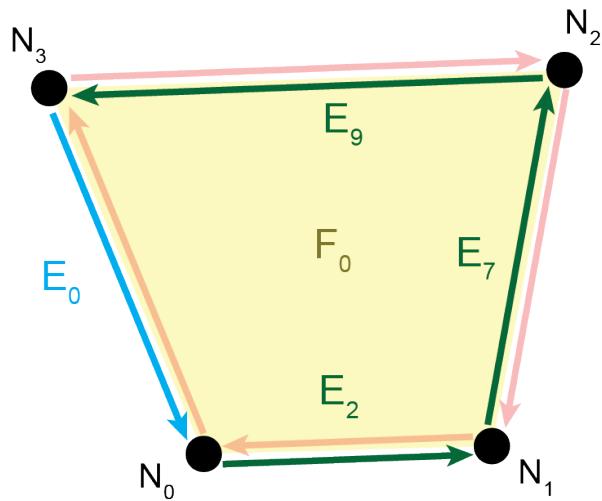


Abbildung 12: Flächenberechnung. Dargestellt: Anliegende Kante (cyan), Kanten (grün) und Fläche (gelb) (Abbildung der Verfasser)

3.2.5 Vervollständigung der Knoten

Die letzte nötige Referenz ist die der 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 nachfolgenden dreidimensionalen Berechnungen.

3.3 Aufbau der Einzelbauteile

Die zweidimensionalen Strukturen der DCEL werden genutzt, um das aus OpenSCAD-Objekten bestehende 3D-Modell zu erstellen. Dabei werden Modifikationen an allen Bauteilen vorgenommen, damit 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 OpenSCAD Java Interface

Für die erleichterte Erstellung von OpenScad Objekten wurde ein Java Interface `ScadObject` erstellt, welches alle für das Projekt wichtigen OpenSCAD Operationen 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])` (siehe Abb. 13).

Die Klassen, welche verschiedene modifikative Operationen darstellen, wie z.B. `Rotate` oder `Difference`, lassen sich verschachteln. Dadurch kann mit wenig Aufwand ein komplexes OpenSCAD-Objekt erstellt werden.

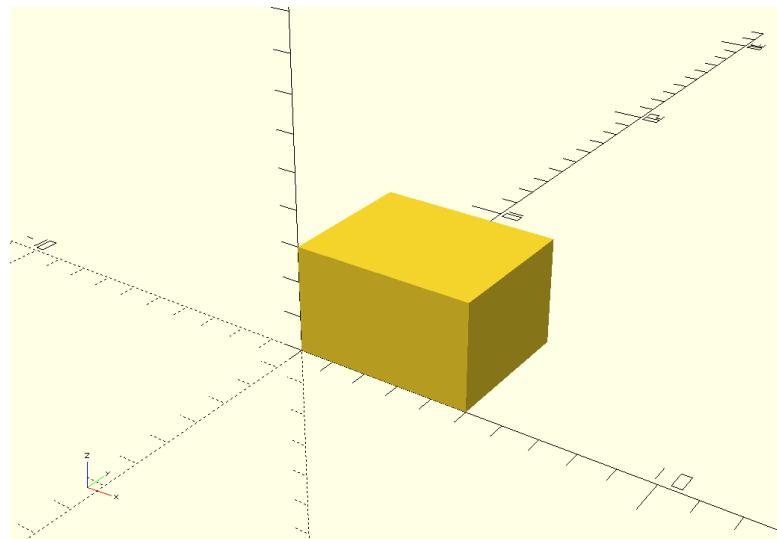


Abbildung 13: Resultat der Eingabe `new Cube(5, 4, 3).toString()`
(Screenshot der Verfasser)

3.3.2 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 worden. Diese sind an zwei voneinander unabhängigen Abschnitten „CornerPin“ und „CornerCylinder“ angebracht, welche zusammengefügt das gesamte 3D-Objekt verkörpern (siehe Abb. 14).

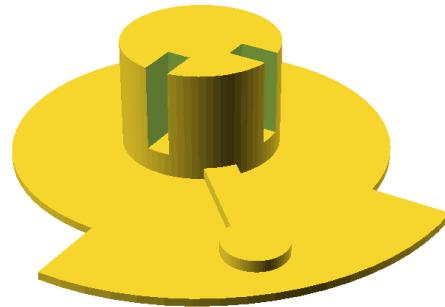


Abbildung 14: Ein Eckpfeiler (Screenshot der Verfasser)

Eckzylinder

Der Eckzylinder im Code `CornerCylinder` stellt den oberen Teil des Eckpfeilers dar. Dieser besteht aus einem zylindrischen Grundbauteil mit Vertiefungen, welche entstehen, indem Schnittmengen zwischen dem Basiszyliner und Quadern durchgeführt werden (siehe Abb. 15). Diese Quadern sind in Richtung der vom Knoten ausgehenden Kanten rotiert, sodass in diese die angrenzenden Wände eingeschoben werden können.

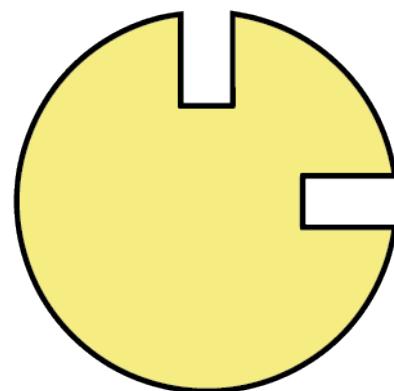


Abbildung 15: Querschnitt eines Eckzylinders mit zwei angrenzenden Wänden (Abbildung der Verfasser)

Eckpin

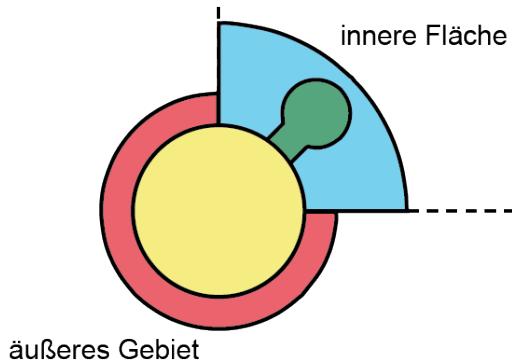


Abbildung 16: Draufsicht auf den unteren Abschnitt (Eckpin) eines Eckpfeilers mit zwei anliegenden Flächen (Abbildung der Verfasser)

Die Summe aller Objekte des Typs `CornerPin` repräsentieren den unteren Abschnitt des Eckpfeilers, welcher die Fixierung der Grundplatten mit diesem gewährleistet. Ein einzelner Eckpin besteht aus einem Zylinder, einer Ausstülpung an dessen Seite und einem Zylindersegment.

Das Grundgerüst wird durch den Basiszylinder des Eckpfeilers festgelegt, welcher auch beim Eckzylinder vorkommt. Für die Ausstülpung (siehe Abb. 16, grün) 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. Der Quader wird stets so lang berechnet, dass die Ausstülpung einen definierten Abstand von den Wänden einnimmt (siehe 3.3.5 „pinDistance“). Für die untere Abgrenzung des Pinkopfes (siehe Abb. 16, 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 stattdessen eine Differenzmenge zwischen der Umrandungsfläche des Grundrisses und dem flachen Zylinder gebildet. Dies resultiert in einem Zylindersegment, wie es in Abb. 16 (rot) zu sehen ist. Jeder Eckpfeiler

hat dadurch einen stabilisierten unteren Abschnitt, auch außerhalb des Grundrisses.

3.3.3 Wandstücke

Eine Wand verkörpert eine Kante der DCEL. Für die Erstellung dieser wird ein rotierter Quader mit der Länge der entsprechenden Kante benutzt. 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 (siehe Abb. 17 und Abb. 18).



Abbildung 17: Querschnitt einer Wand (Abbildung der Verfasser)

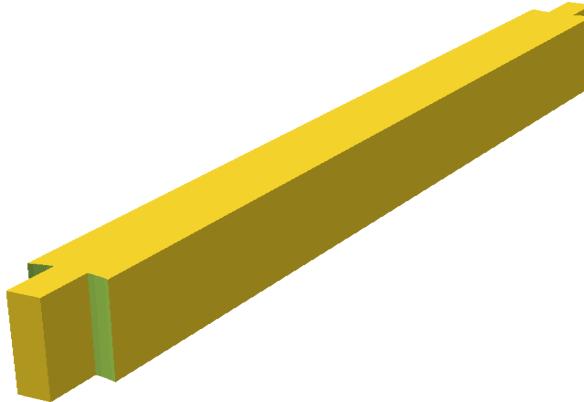


Abbildung 18: Ein Wand in der OpenSCAD Darstellung (Screenshot der Verfasser)

3.3.4 Grundplatten

Eine Grundplatte wird anhand einer Fläche des Grundrisses konstruiert. Sie ist aufgebaut aus einem geradem Prisma mit einer Polygon-Grundfläche, welches eingestülpte Steckmechanismen an der Unterseite besitzt (siehe Abb. 19).

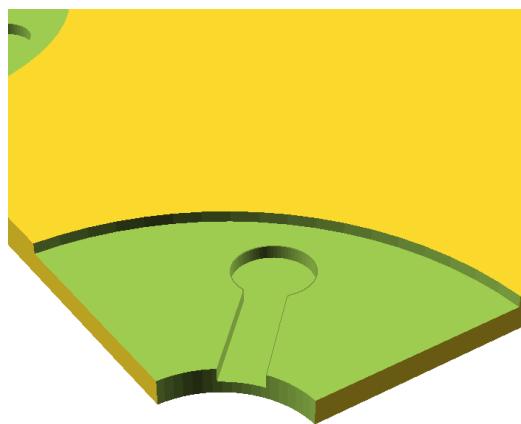


Abbildung 19: Stecker einer Grundplatte (Screenshot der Verfasser)

Für die Berechnung des 3D-Objekts, wird an jedem Knoten der Fläche 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 der außen liegenden Seite um die halbe Wanddicke vergrößert, damit ein abgeschlossener Eindruck des gesamten 3D-Modells entsteht (siehe Abb. 20).

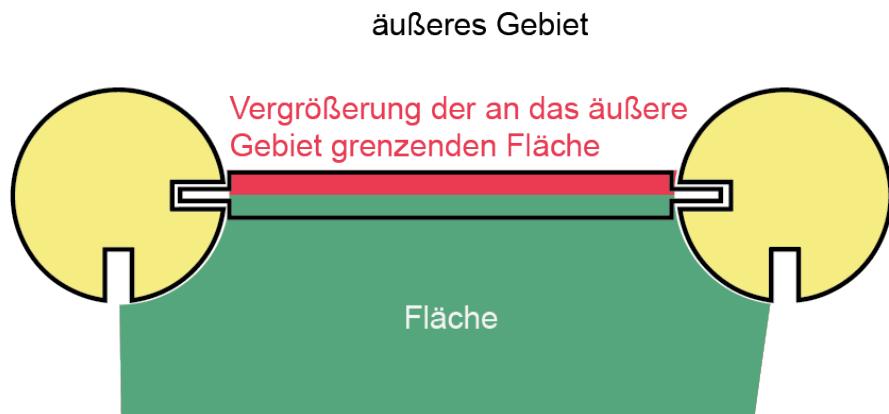


Abbildung 20: Vergrößerung der äußeren Grundplatten (Abbildung der Verfasser)

3.3.5 Zuweisen von Berechnungskonstanten

Die `Params`-Klasse dient als Speicher für alle nötigen Konstanten, welche für die Umrechnung des Grundrisses in das 3D-Modell notwendig sind. Jedes dreidimensionale Objekt benötigt eine Instanz einer solchen Klasse, um den OpenSCAD Code zu generieren. In der Anwendung werden von einer Instanz der `ScadProcessor`-Klasse die angegebenen Parameter an alle Elemente eines 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 (vgl. Codeausschnitt 5) und können über eine `Params`-Objekt aufgerufen werden.

```

1  public Params(double epsilon, ...) {
2      this.epsilon = epsilon;
3      // ...
4 }
```

Codeausschnitt 5: Konstruktor der `Params`-Klasse, in welchem alle Parameter festgelegt werden.

Das Abrufen der Parameter erfolgt mittels der entsprechenden `get()`-Funktionen der `Params`-Klasse, welche für alle Parameter vorhanden sind. Die nötigen Parameter stellen Gleitkommazahlen (`double`) dar, die verschiedene Teile des Grundrisses beeinflussen.

Epsilon

Der Parameter Epsilon (ϵ) bezeichnet einen Abstand zwischen 3D-Objekten, der benötigt wird, um einer möglichen Druckabweichung des 3D-Druckers entgegenzuwirken. Diese kann z.B. bewirken, dass Steckmechanismen nicht ineinander passen. Deswegen tritt der Abstand ϵ 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 negativen (eingestülpten) Stecker (Wand/Eckpfeiler und Grundplatte/Eckpfeiler) ist der negative gleichmäßig um ϵ vergrößert, damit ein abschließendes Stecken ermöglicht wird.
2. Wenn zwei Grundplatten aufeinandertreffen, werden die sich berührenden Seiten um $0.5 * \epsilon$ entgegengesetzt zueinander verschoben zur Einhaltung eines Abstandes von ϵ .
3. Die Höhe einer Wand wird um ϵ verringert, damit die Oberseite des gesamten 3D-Objekts eine glatte Oberfläche besitzt.

Das im Programm definierte Epsilon wurde mittels einem Experiment (siehe Abb. 21) ermittelt. Dafür wurde ein Stecker und Einstülpungen gedruckt, die um ein verschiedenes ϵ vergrößert wurden, sodass getestet werden konnte, welches Epsilon für den MakerBot Replicator™ 2 am geeigneten ist. Ausgewertet wurde daraus ein Wert von 0.25mm. Der ϵ -Parameter ist deswegen ein druckerspezifischer Parameter.



Abbildung 21: Versuch zur Ermittlung eines passenden Epsilon-Wertes (Abbildung der Verfasser)

Weitere Konstanten

Die restlichen Parameter werden in der folgenden Tabelle gelistet.

Konstante	Bedeutung	Wert in mm
cornerRadius	Radius des Basiszylinders der Eckpfeiler	10
pinMinLength	minimale Länge des Pinkopfes am Eckpin	2
pinPWidth	Breite des Quaders des Pinkopfes	4
pinPRadius	Radius des Zylinders am Pinkopf	4
pinDistance	minimaler Abstand zwischen Pinkopf und Wand	2
height	Höhe des 3D-Modells	75
pinHeight	Höhe des Pinkopfes	4
basePlateHeight	Höhe der Grundplatten	4
basePlatePinCircle-Height	Höhe der Kreissegmenten an den Eckpins	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 Grundfläche des 3D-Druckers begrenzt ist.

In unserem Fall wies diese eine Länge von 28,5 cm und eine Breite von 15,2 cm auf (vgl. Quelle [7]).

3.4.1 Minimierung der Druckvorgänge

Um die Dimensionen der Grundfläche 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 werden würde. Anschließend wird eine neue Reihe eröffnet, in der nun neue Elemente platziert werden. Dieser Prozess wird so lange 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. Die Vereinigungen von Grundplatten werden zusätzlich um 180° gedreht, damit keine Überhänge in der Datei vorkommen, was zu einer Verlängerung der Druckzeit führen würde.

Jedes einzelne Element in der Liste kann mit einem Durchgang gedruckt werden(vgl. Abb. 22, 23 und 24).

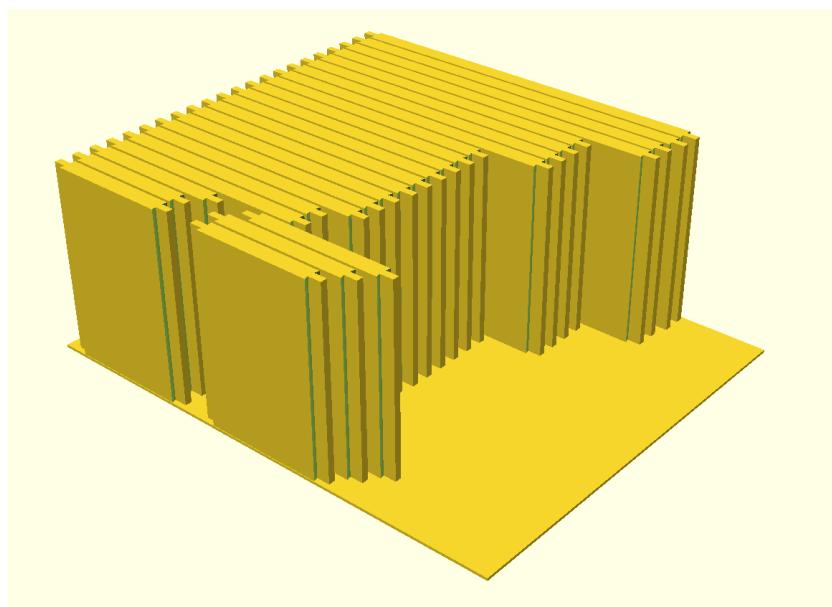


Abbildung 22: platzeffiziente Platzierung von Wänden (Screenshot der Verfasser)



Abbildung 23: platzeffiziente Anordnung von Eckpfeilern auf der Grundfläche (rot) (Abbildung der Verfasser)

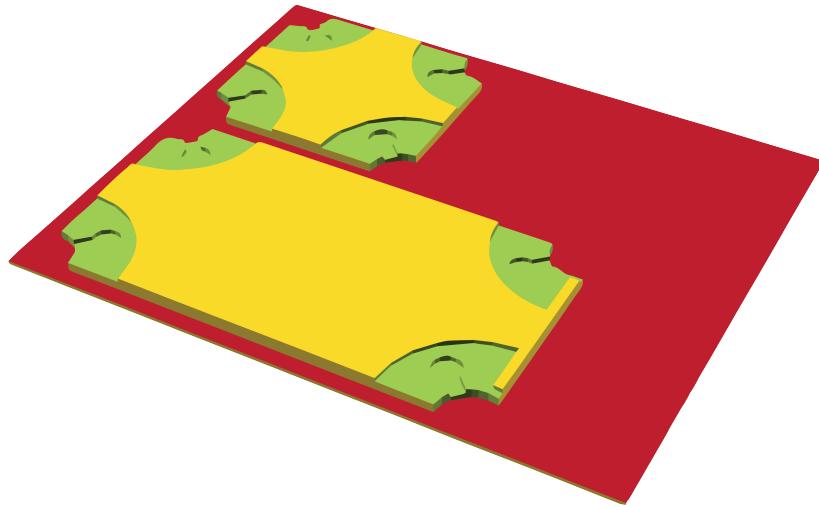


Abbildung 24: platzeffiziente Anordnung von Grundplatten auf der Grundfläche (rot) (Abbildung der Verfasser)

3.4.2 Ermittlung der Größen

Das Einzige, was die Strukturen Wand- und Eckteil, sowie Grundplatte in dem beschriebenen 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 können. Als Länge wird die Länge der zugehörigen Kante verwendet. Die Breite ermittelt sich durch die definierte `wallWidth` der übergebenen `Params`-Klasse.

Eckpfeiler

Für die Größenberechnung der Eckpfeiler wird der längste Eckpin ermittelt, welcher an dem jeweiligen Objekt anliegt. Die Länge dieses Eckpins stellt

dann die Seitenlänge eines Quadrats dar, welche die Ausmaße des Eckpfilers angibt.

Grundplatten

Um die optimale Größe der Grundplatten zu berechnen, wird die OMBB dieser berechnet. Für die Berechnung wird zuerst die konvexen Hülle ermittelt, da nach dem von Freeman und Shapira bewiesenen Satz, eine Seite des minimalen Begrenzungsrechtecks kollinear mit einer der konvexen Hülle sein muss (vgl. Quelle [4]). 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 aus denen 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 gleich 180° , liegt S links von der Geraden 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, bis der Anfangspunkt P_0 wieder erreicht wurde, sodass die konvexe Hülle „geschlossen“ ist.

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

Winkel, mit welchem diese erreicht wird.

Diese Werte werden auf die Grundplatte angewendet, wobei die Breite und Höhe jeweils vergrößert wird, damit die Modifikationen berücksichtigt werden, die an der Grundplatte durchgeführt wurden (vgl. 3.3.4).

3.5 Ausgabe der Resultate

Nachdem die Berechnungen des Modells abgeschlossen ist, müssen die Ergebnisse nun ausgegeben werden. Für die Ausgabe der Dateien wird ein Ordner in dem Verzeichnis erstellt, in dem auch das Programm ausgeführt wird.

3.5.1 Erstellung des Ausgabeordners

Als Name des Ordners wird der Name genutzt, den der Nutzer in der GUI eingegeben hat. Im Unterordner „scad“ dieses Ordners finden sich dann sämtliche .scad-Dateien, im Unterordner „stl“ alle .stl-Dateien.

Zum Erstellen der Unterordner wird die Funktion `createDirs()` verwendet (vgl. Codeausschnitt 6). Diese legt mittels der Anweisung `Files.createDirectories()` die Unterverzeichnisse an. Sollte der übergeordnete Ausgabeordner noch nicht vorhanden sein, wird er durch die Anweisung direkt mit angelegt.

```
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 6: Erstellung der Unterordner

3.5.2 STL Konvertierung

Um die ermittelten .scad-Dateien im .stl-Format auszugeben, wird die Kommandozeilenfunktionalität von OpenSCAD verwendet. Hierzu werden zunächst alle Dateien im „scad“-Unterordner mit der Endung .scad durch die Klasse SCADFinder ermittelt und in einem Feld ausgegeben. Dies geschieht mittels der Funktion findFiles() (vgl. Codeausschnitt 7), welche durch einen FileFilter alle Dateien mit gewünschter Endung zurückgibt.

```
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  
6              filename)  
7          { return filename.endsWith(".scad"); }  
8      });  
9  }
```

Codeausschnitt 7: Ausgabe aller .scad-Dateien aus einem Ordner

Mit den Namen dieser Dateien als Argumente, kann im Anschluss die Umwandlung ins .stl-Format stattfinden. Die verwendete Konsolenanweisung setzt sich 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 . Diese Anweisungen verwendet die Klasse STLConverter mittels eines ProcessBuilder, um die .stl-Dateien im „stl“-Unterordner zu erstellen (vgl. Codeausschnitt 8).

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

Codeausschnitt 8: Ausführung der Kommandozeilenanweisungen

4. Ausblick

Die Anwendung ermöglicht eine vollautomatische Umwandlung eines Grundrisses in einzelne Bauteile, aus denen das Modell zusammen gesetzt werden kann. Entsprechend der ursprünglichen Aufgabe sind damit alle Kriterien erfüllt. Das Drucken der ausgegebenen Dateien liegt jedoch beim Nutzer, da dies noch nicht vollautomatisch geschehen kann.

Als Erweiterungsmöglichkeiten der Anwendung sticht besonders die Aufteilung der Bauteile hervor. Falls Elemente zu groß für die Grundfläche des 3D-Druckers sind, ist eine Teilung notwendig, damit der Grundriss gedruckt werden kann.

Die Möglichkeit, den Nutzer die gewünschten Werte eingeben zu lassen, sowie eine Überarbeitung der Benutzeroberfläche sind weitere mögliche Ansatzpunkte.

Den Hauptteil der Laufzeit nimmt zudem die Umwandlung der .scad-Dateien in das .stl-Format in Anspruch. Eine Optimierung und weitere Vereinfachung der Konvertierung ist äußerst wünschenswert.

5. Internetquellenverzeichnis

- [1] https://de.wikipedia.org/wiki/Planarer_Graph
(Stand: 07.11.2017, 16:00 Uhr)
- [2] 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
(Stand: 25.04.2017, 12:00 Uhr)
- [4] <https://geidav.wordpress.com/2014/01/23/computing-oriented-minimum-bounding-boxes-in-2d/>
(Stand: 17.12.2017, 13:00 Uhr)
- [5] <https://en.wikipedia.org/wiki/AutoCAD>
(Stand: 05.06.2016, 18:00)
- [6] en.wikibooks.org/wiki/OpenSCAD_User_Manual
(Stand: 25.04.2017, 12:00 Uhr)
- [7] https://eu.makerbot.com/fileadmin/Inhalte/Support/Manuals/German_UserManual_V.4_Replicator2.pdf
(Stand: 06.06.2017, 09:00)
- [8] de.wikipedia.org/wiki/Gau%C3%9Fsche_Trapezformel
(Stand: 21.03.2017, 12:00 Uhr)
- [9] kabeja.sourceforge.net/
(Stand: 12.10.2017, 10:00 Uhr)
- [10] <https://en.wikipedia.org/wiki/OpenSCAD>
(Stand: 05.06.2017, 18:00)

6. Abbildungsverzeichnis

1	Schema eines planaren Graphen (Abbildung der Verfasser)	4
2	Schema einer DCEL (Abbildung der Verfasser)	5
3	Beispiel einer konvexen Hülle (vgl. Quelle [4])	6
4	Eine mögliche Bounding Box (l.) im Vergleich zur OMBB (r.) (vgl. Quelle [4])	6
5	Grundriss aus AutoCAD (Screenshot der Verfasser)	7
6	Differenzmenge zwischen einem Würfel und einem Zylinder in OpenSCAD (Screenshot der Verfasser)	8
7	Ausgangszustand der Benutzeroberfläche (Screenshot der Verfasser)	11
8	Benutzeroberfläche mit aktiviertem StartButton (Screen- shot der Verfasser)	12
9	Dialog zur Warnung des Nutzers (Screenshot der Verfasser) .	13
10	Dialog zum Verlinken der openscad.exe (Screenshot der Ver- fasser)	13
11	Veranschaulichung 1. und 2. (Abbildung der Verfasser)	17
12	Flächenberechnung. Dargestellt: Anliegende Kante (cyan), Kanten (grün) und Fläche (gelb) (Abbildung der Verfasser) . .	19
13	Resultat der Eingabe new Cube(5, 4, 3).toString() (Screenshot der Verfasser)	21
14	Ein Eckpfeiler (Screenshot der Verfasser)	22
15	Querschnitt eines Eckzylinders mit zwei angrenzenden Wän- den (Abbildung der Verfasser)	22
16	Draufsicht auf den unteren Abschnitt (Eckpin) eines Eckpfei- lers mit zwei anliegenden Flächen (Abbildung der Verfasser) .	23
17	Querschnitt einer Wand (Abbildung der Verfasser)	24

18	Ein Wand in der OpenSCAD Darstellung (Screenshot der Verfasser)	24
19	Stecker einer Grundplatte (Screenshot der Verfasser)	25
20	Vergrößerung der äußeren Grundplatten (Abbildung der Verfasser)	26
21	Versuch zur Ermittlung eines passenden Epsilon-Wertes (Abbildung der Verfasser)	28
22	platzeffiziente Platzierung von Wänden (Screenshot der Verfasser)	30
23	platzeffiziente Anordnung von Eckpfeilern auf der Grundfläche (rot) (Abbildung der Verfasser)	30
24	platzeffiziente Anordnung von Grundplatten auf der Grundfläche (rot) (Abbildung der Verfasser)	31

7. Arbeitsteilung

Wir versichern hiermit, dass die Besondere Lernleistung und der beigelegte Quelltext von uns zu gleichen Teilen erarbeitet wurde.

Die Aufteilung erfolgte so, dass Johann Bartel die Kapitel „1. Einleitung“, „2.1 Planare Graphen“, „2.2 Doubly Connected Edge List“, „2.3 Oriented Minimum Bounding Box“, „3.3 Aufbau der Einzelteile“ und „3.4 Druckvorgang“ übernahm.

Peter Oehme fertigte die Kapitel „2.4 AutoCAD“, „2.5 OpenSCAD“, „2.6 3D-Drucker MakerBot ReplicatorTM 2“, „3.1 Einlesen Des Grundrisses“, „3.2 Erstellen der Doubly Connected Edge List“, „3.5 Ausgabe der Resultate“ und „4. Ausblick“ 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

8. 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