# Todo list

**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 6.06.2017

# Bibliographische Beschreibung

Bartel, Johann und Oehme, Peter

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

28 Seiten, Y Anlagen, 11 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.

Immobilienkäufer und -verkäufer 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

# 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. Grundlagen

## 2.1 Planare Graphen

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. 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.

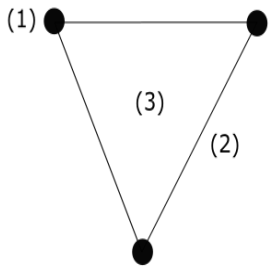


Abbildung 1: Schema eines planaren Graphen (Abbildung der Verfasser)

In Abbildung 1 wird ein solcher planarer Graph dargestellt. Der linke obere Knoten wir 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-, eine Nachfolger- und eine Zwillingskante. Die jeweiligen Zwillingskanten beschreiben hierbei die invertierten Varianten der

betrachteten Kanten. Jedem Knoten wird außerdem eine ausgehende Kante und allen Flächen eine anliegende Kante zugewiesen.

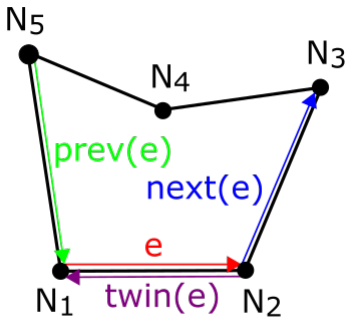


Abbildung 2: Schema einer DCEL (Abbildung der Verfasser)

In Abbildung 2 wird die Kante „e“ durch den Anfangsknoten durch „$N_1$“ und den Endknoten „$N_2$“ gebildet. Die Zwillingskante wird mit „twin(e)“ bezeichnet, die Nachfolgerkante mit „next(e)“ und die Vorgängerkante mit „prev(e)“. Diese Referenzierungen ermöglichen es, ausgehend von einem Element ohne umfangreiche Berechnungen auf alle anderen Objekte zu schließen, indem bei der Betrachtung von Knoten und Flächen die zugehörigen Kanten, beziehungsweise bei der Betrachtung von einzelnen Kanten deren Vorgänger und Nachfolger betrachtet werden.

Johann wollte DCEL noch mal überarbeiten

## 2.3 AutoCAD

AutoCAD ist ein grafischer Zeichnungseditor, welcher zum Erstellen von technischen Zeichnungen und dem Modellieren von Objekten verwendet wird. 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.

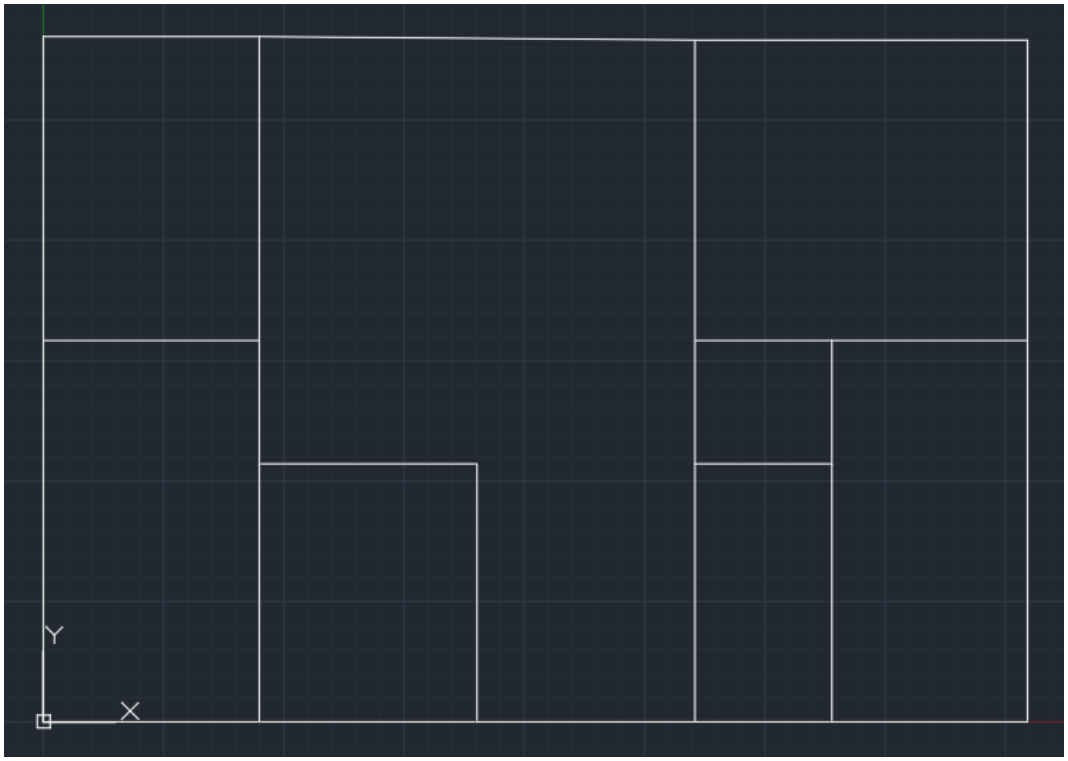


Abbildung 3: Grundriss aus AutoCAD (Screenshot der Verfasser)

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 3 zu sehen.

## 2.4 OpenSCAD

OpenSCAD ist eine kostenlos verfügbare CAD-Modellierungssoftware, welche aus einer textbasierten Beschreibungssprache 3D-Modelle erzeugt. 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()`) mit einander verknüpft und kombiniert werden, um neue Objekte nach eigenen Ansprüchen zu bilden.

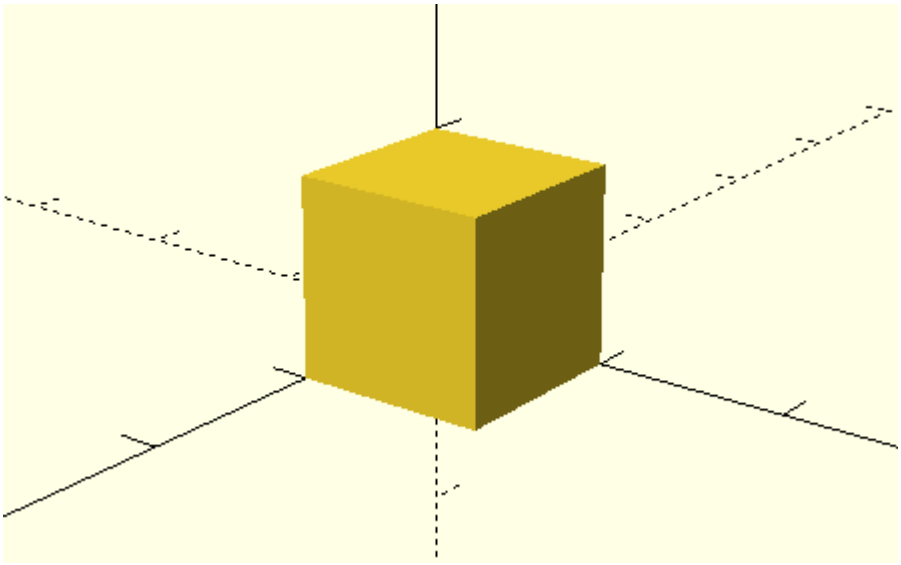

Abbildung 4: Eine Vereinigung zweier Würfel in OpenSCAD (Screenshot der Verfasser)

Neben solchen einfachen Objekten, wird außerdem die Möglichkeit geboten, komplexere Objekte wie Polygone (`polygon()`) zu erstellen und diese dann ausgehend vom zweidimensionalen Polygon in ein dreidimensionales Polygon umzuwandeln (`linear_extrude()`), welches 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, welche schlussendlich mittels der dem 3D-Drucker beiliegenden Software entweder durch einen USB-Anschluss des 3D-Druckers oder auf einer SD-Karte gespeichert ausgedruckt werden. Auch an dieser Stelle des Modellierungsvorganges bietet sich OpenSCAD wieder an, da dieses die Option zur Konvertierung vom .scad-Format zu .stl bietet.

diesen Druckteil evtl. in den Hauptteil verlagern

## 2.5 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[1] 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

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

erheblich kürzeren Druckzeit verbunden.

Die Druckzeit wird außerdem von der eingestellten Ausfüllung von geschlossenen 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 verringerter 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 [6].

```
1  public static ArrayList<Line> getAutocadFile(String filePath)
       throws ParseException {
2        ArrayList<Line> vcs = new ArrayList<>();
3        Parser parser = ParserBuilder.createDefaultParser();
4        parser.parse(filePath, DXFParser.DEFAULT_ENCODING);
5        DXFDocument doc = parser.getDocument();
6        List lst = doc.getDXFLayer("0").getDXFEntities(
             DXFConstants.ENTITY_TYPE_LINE);
7        for (int index = 0; index < lst.size(); index++) {
8              DXFLine dxfline = (DXFLine) lst.get(index);
9              Line v = new Line(
10             new Vector(round2(dxfline.getStartPoint().getX()),
                   round2(dxfline.getStartPoint().getY())),
11             new Vector(round2(dxfline.getEndPoint().getX()),
                   round2(dxfline.getEndPoint().getY())));
12             vcs.add(v);
13       }
14
15       return vcs;
16 }
```

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`, indem ein `JTextField`, ein `FileChooserButton`, ein `StartButton` und ein `ShowResultButton` platziert sind.

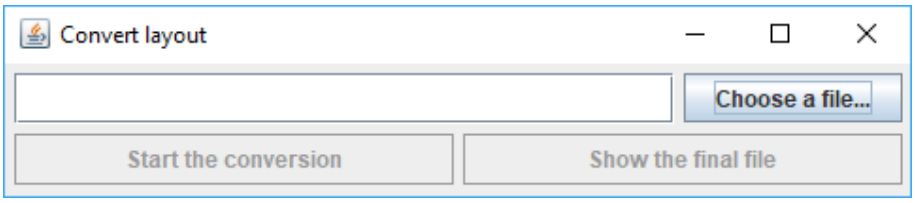


Abbildung 5: Ausgangszustand der GUI (Screenshot der Verfasser)

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. 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 `JTextField` den Dateipfad eingeben.

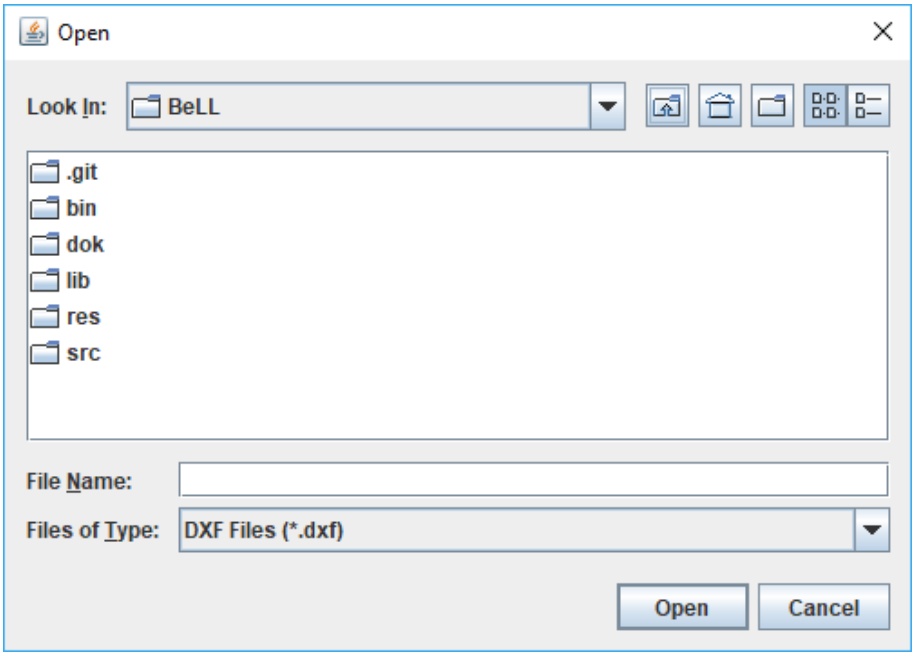


Abbildung 6: Dateiöffnungsdialog (Screenshot der Verfasser)

Nachdem der Nutzer eine existente Datei ausgewählt hat, wird nun der `StartButton` aktiviert und er kann den Konvertierungsprozess starten.

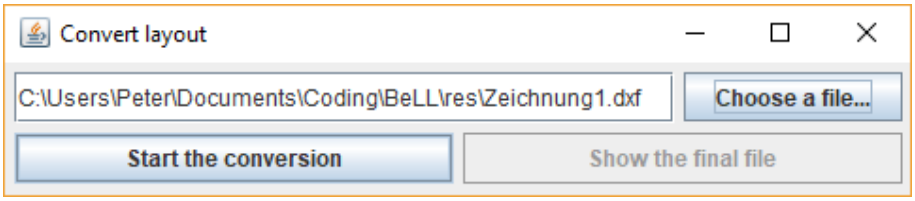


Abbildung 7: GUI mit aktiviertem `StartButton` (Screenshot der Verfasser)

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

Bild der finalen GUI einfügen

## 3.2 Erstellen der Doubly-connected Edge List

### 3.2.1 Graph Klasse

Die wichtigste Klasse der Anwendung ist `Graph`. Sie verbindet alle grafischen Bestandteile und kann mit einer übergebenen Liste aus Linien mittels mehrerer Zwischenoperationen eine vollständige DCEL, also einen planaren Graph erstellen. Dafür sind drei Listen vom Typ `Edge`(Kante), `Node`(Knoten) und `Face`(Fläche) gespeichert, die eine DCEL repräsentieren.

### 3.2.2 Line-to-Edge Konvertierung

Initialisierend werden aus den Koordinaten der eingelesenen Start- und Endpunkte in Knoten umgewandelt. Aus diesen noch unreferenzierten Knoten definiert man dann die Kanten mit entsprechendem Start- und Endknoten. Um für die Eindeutigkeit der Knoten zu sorgen, gibt die Funktion `createNode()` ein äquivalenten Knoten zu einem Punkt zurück und fügt ihn zu der Liste von Knoten hinzu, wenn er noch nicht in ihr vorhanden ist.

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

Codeauschnitt 2: Line-to-Edge Konvertierung

```
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   }
```

Codeauschnitt 3: `createNode()` Funktion

Hier könnte man noch was zu sagen

### 3.2.3 Zwillingskantengenerierung

Weiterhin werden für sämtliche Kanten die invertierten Gegenstücke, auch als „Zwillingskanten“ bezeichnet, durch Vertauschen der Start- und Endknoten gebildet. Direkt nach dem Hinzufügen der Zwillingskante wird jeweils eine Referenz erstellt, die in beiden Kanten auf den jeweils zugehörigen Zwilling verweist. In der Kantenliste existiert nun für jede Linie die der DXF-Reader eingelesen hat, zwei zueinander komplementäre Kanten. Die Zwillingskanten dienen dem Aufbau der DCEL, um eine Iteration in beide Richtungen um die Flächen und so auch eine einfachere Umwandlung zu ermöglichen.

```
1  private void computeTwins() {
2         int amount = edges.size();
3         for (int i = 0; i < amount; i++) {
4               edges.add(edges.get(i).generateTwin());
5               edges.get(i).setTwin(edges.get(edges.size() - 1));
6               edges.get(edges.size() - 1).setTwin(edges.get(i));
7         }
8   }
```

Codeauschnitt 4: Hinzufügen der Zwillingskanten

### 3.2.4 Nachfolger- und Vorgängerermittlung

Zum Erstellen der Referenzen werden zuerst alle ausgehenden Kanten der Knoten, das heißt die den jeweiligen Knoten als ihre Startkante besitzen,

in einer zweidimensionalen `ArrayList` gespeichert. Die erste Dimension steht für den Index des Knotens in der erstellten Knotenliste. In der zweiten Dimension liegen die den Indizes zugehörigen ausgehenden Kanten vor. Da diese durch eine `for()` Schleife mit der oben stehenden Bedingung aufgerufen werden, liegen die Kanten im `Array` in zufälliger Reihenfolge vor. Sortiert werden die Kanten mithilfe des Winkels, den die Funktion `atan2()` der `java.Math`-Bibliothek liefert, im positiven Drehsinn. Begonnen wird dabei mit dem kleinsten zurückgegebenen Winkel. Daraus ergibt sich, dass das vorherige Element die Kante, die mit einem größeren Winkel „links“ der Betrachteten darstellt, beziehungsweise das nachfolgende Element, die mit einem kleineren Winkel „rechts“ der Ausgewählten liegt darstellt.
Für jede ausgehende Kante `e` können nun folgende Referenzen gesetzt werden:

- Das vorherige Element der `ArrayList` stellt den Nachfolger der Zwillingskante von `e` dar.
- Der Vorgänger von `e` ist der Zwilling der in der `ArrayList` nachfolgenden Kante.

Genannte Referenzen werden nun gesetzt, sodass die Verknüpfungen zwischen den Kanten fertiggestellt sind.

code eventuell bilder definitiv

### 3.2.5 Flächenerstellung

Mithilfe einer Schleife können die einzelnen Flächen erschlossen werden. Es muss lediglich fortlaufend alle Nachfolger einer gewählten Kante gesucht werden bis die gefundene Kante einen Nachfolger darstellt, bis alle Kanten genau einmal behandelt wurden. Dafür wurde eine repräsentative `Boolean-ArrayList` mit der Länge der Kantenliste erstellt, die mit ihren Werten den Status der Kanten darstellt. Folglich besteht diese am Anfang

nur aus `false` Werten die während der Schleife alle auf `true` gesetzt werden. Die Speicherung der Flächen folgt dem DCEL-Standard, sodass nur eine anliegende Kante zugewiesen wird, mit der man, aufgrund von den Referenzen alle Kanten und Knoten der Fläche ermitteln kann. Das äußere Gebiet des planaren Graphen lässt sich an der Umlaufrichtung der Kanten erkennen. Es ist die einzige Fläche, die im mathematisch negativen Drehsinn ausgerichtet ist. Unterschieden wird sie im weiteren Programmablauf mithilfe der Gaußschen Trapezformel. Bei negativer Umlaufrichtung ist das Ergebnis dieser negativ.

Bitte noch mal überarbeiten, total wirr

### 3.2.6 Vervollständigung der Knoten

Die letzte nötige Operation ist die Speicherung einer anliegenden Edge in den Nodes.

man könnte das im code auch vorher tun denke ich

## 3.3 Aufbau der Einzelbauteile

Die drei Kriterien, die die Unterteilung des Modells einhalten soll, sind wie folgt festgelegt:

- Die Einzelteile sollen möglichst simpel gestaltet sein, um unnötig komplizierten Konflikten vorzubeugen.
- Die Untereinheiten sollen durch so geringe Modifikation wie möglich einen guten seitlichen Einblick in das Modell gewähren.
- Die Untereinheiten sollen auch nach dem Entfernen einzelner Bauteile eine möglichst stabile Einheit bilden.

Diese Kriterien lassen sich mit herausnehmbaren Wandstücken realisieren, welche im restlichen Modell verankert sind, so dass sie herausgenommen und auch wieder eingesetzt werden können. Um diese Wände auch weiterhin im Modell fixieren zu können, werden Eckpfeiler verwendet, in welche die Wandstücke, eingesetzt werden. Da die Einheit aus Eckpfeilern und leicht entfernbaren Wandstücken nicht sehr stabil ist, werden nun am unteren Teil der Eckpfeiler noch Stecker hinzugefügt, um eine Grundplatte zu fixieren und so eine stabile Einheit zu erhalten. Zur Realisierung des Stecksystems müssen anschließend für beide Verankerungsmechanismen geeignete Designs ausgearbeitet werden.

Ausdruck

### 3.3.1 Eckpfeiler

Die Eckpfeiler des Modells symbolisieren die Schnittpunkte der Wände des Grundrisses. Sie dienen dem Befestigen der Grundplatten und Wandstücke. Um das Stecksystem zu realisieren, verfügen die Eckpfeiler über die meisten Verbindungen zu den anderen Objekten. Um diese Verbindungen jedoch problemlos herstellen zu können, ist ein komplexes Modell vonnöten.

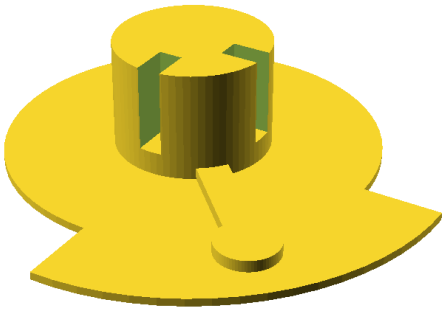

Abbildung 8: Ein Eckpfeiler (Screenshot der Verfasser)

**CornerCylinder**

Der „CornerCylinder“ stellt den oberen Teil eines Eckpfeilers dar. Dieser besteht aus einem zylindrischen Grundbauteil mit Vertiefungen, in denen die Wände befestigt werden. Dieser ist auf der Oberseite des Eckpfeilers in Abbildung 8 zu sehen.
In dessen Berechnung werden alle an dem Knoten anliegenden Kanten betrachtet und eine Schnittmenge zwischen einem Grundzylinder und in die Richtung der Kanten gedrehten Quadern gebildet.

**CornerPin**

Der „CornerPin“ ist der untere Abschnitt des Eckpfeilers, welcher der Verbindung des Eckpfeilers mit den Grundplatten dient. Hierfür verfügt er über positive Steckmechanismen, die auf der Unterseite des Eckpfeilers in Abbildung 8 zu sehen ist.
Für jede anliegende Fläche wird dabei ein neues Objekt berechnet. Es ist dabei zu unterscheiden, ob die Fläche das äußere Gebiet oder einen Teil des inneren Gebietes darstellt. Auf der Innenseite des Grundrisses sollten Steckmechanismen angebracht werden, um die Verankerung der Grundplatten zu gewähren, was außen nicht notwendig ist, da dort keine reale

Fläche angelegt wird.

Wenn an einen Knoten das äußere Gebiet angrenzt, wird der entsprechende Eckpfeiler dessen nicht mit einem Pin, sondern nur mit einer Umrandung versehen. Diese entsteht durch eine Differenzmenge zwischen einem Basiszylinder, der den kompletten Eckpfeiler umschließt und der äußeren Fläche. Gewährleistet wird das durch die Verarbeitung von Polygonen mit OpenSCAD. Obwohl die unendliche Fläche in der Differenz mit einbezogen ist, repräsentiert diese immer auch den kompletten inneren Raum des Grundrisses, so dass eine Differenz möglich ist.

Bei einem Knoten, der nicht an das äußere Gebiet angrenzt, wird ein positiver Steckmechanismus errechnet. Er setzt sich zusammen aus einer Basis, einem Quader und einem Zylinder. Die Länge des Pins kann dabei variieren. Sie ist so definiert, dass zum Rand der Fläche immer ein gewisser Abstand vorherrscht. Dies vermeidet Komplikationen nach und während des Druckens, wo sonst Hohlräume überlappen könnten. Die Basis entsteht durch eine Schnittmenge der betrachteten Fläche mit dem Basiszylinders des Pins.

CornerPin

### 3.3.2 Wandstücke

Die Wände des Modells stellen die Wände des Grundrisses dar. Eine Wand setzt sich aus jeweils drei Quadern zusammen.

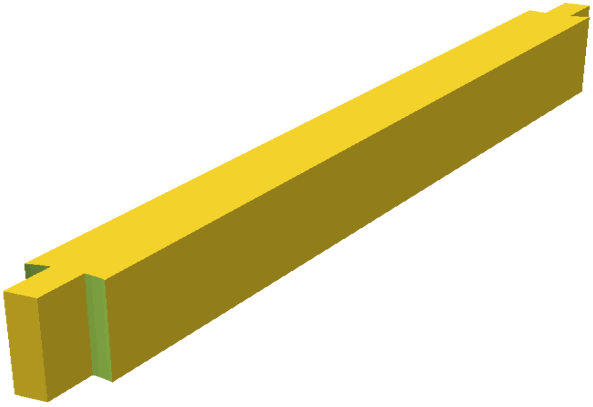

Abbildung 9: Ein Wandteil (Screenshot der Verfasser)

Der größte dieser Quader stellt, wie in Abbildung 9 zu sehen, den mittleren Hauptteil dar. Dieser Teil verfügt über die tatsächliche Länge der Wand, welche im Grundriss angegeben ist. Die Höhe und Breite der Wand werden jedoch mit konstanten Parametern zu Beginn des Programmes festgelegt.
Die zwei kleineren Quader dienen dem Befestigen der Wand am Eckpfeiler. Dafür wurden die beiden Quader mittig an die beiden schmalen Seiten des Mittelteils gesetzt. Die Verbindungsstücke verfügen dabei über die gleiche Höhe wie die eigentliche Wand, ihre Breite beträgt aber nur ein Drittel der Wandbreite.

Details über die Länge der Pins hinzufügen!

$\varepsilon$?

### 3.3.3 Grundplatten

Grundplatten repräsentieren die Flächen des Grundrisses. Sie sind aufgebaut aus einem extrudierten Polygon, welches inverse Steckmechanismen am Boden aufweist.

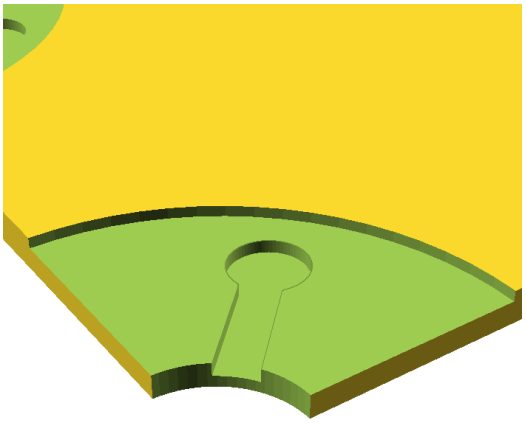

Abbildung 10: Eine Grundplatte (Screenshot der Verfasser)

Ein Ausschnitt einer solchen Grundplatte ist in Abbildung 10 zu sehen. Die inversen Steckmechanismen dienen dem Anbringen der Grundplatte an die Eckpfeiler. Für jeden Knoten, der an der Fläche anliegt wird hierbei je eine Einkerbung vorgenommen, die auf das Stecksystem des zu dem Knoten zugehörigen Eckpfeiler passt.

Abstand zwischen Flächen; Vergrößerung von an das äußere Gebiet angrenzende Fläche

### 3.3.4 Zuweisen von Berechnungskonstanten

#### Funktion der `Params`-Klasse

Die `Params`-Klasse wird als statische Zugriffsmöglichkeit auf bestimmte Konstanten des Programms verwendet, welche bei der Berechnung der Bauteile vonnöten sind. Diese Klasse verfügt hierbei über öffentliche statische Funktionen, mit der aus allen anderen Klassen ohne eine Instanziierung der `Params`-Klasse deren Parameter gesetzt oder auf bereits vorhandene Parameter zugegriffen werden kann. Die Statik der Variablen und Funktionen verhindert hierbei, dass während des Programmablaufes verschiedene Berechnungskomponenten unterschiedliche Konstanten zur

Verfügung gestellt bekommen.

Das Setzen der Parameter findet zu Beginn des Programmes in der `Main`-Klasse statt. Hierbei wird die Funktion `setParams()` aufgerufen. Dieser Funktion werden sämtliche Werte als Parameter des Datentyps `double` übergeben. In der `Params`-Klasse werden dann innerhalb der Funktion allen privaten Variablen ihre Werte entsprechend der Parameter zugewiesen und abrufbar gemacht.

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

Codeauschnitt 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. Ein Überschreiben einzelner Parameter wird an dieser Stelle verhindert, da für die privaten Variablen keine `set()`-Funktionen vorliegen. Der Aufbau der `get()`-Funktionen folgt dem generellen Aufbau des nachfolgenden Codebeispiels, jedoch werden die Parameterbezeichnungen jeweils entsprechend ersetzt:

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

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

Diese Funktionen werden dann aus den Programmteilen, in denen sie für Berechnungen benötigt werden, statisch mittels des Aufrufs der `Params`-Klasse aufgerufen.

Die Bedeutung der einzelnen Parameter erklärt sich wie folgt:

**E ($\epsilon$/Epsilon)**

Der Parameter „E“ entspricht der Konstante $\epsilon$ (Epsilon), welcher aus Gründen der vorteilhaften Kürze der Parameternamen hier verwendet wurde. Somit muss nicht jedes mal `Epsilon` ausgeschrieben werden, es wird auf `E` reduziert. $\epsilon$ bezeichnet den Abstand, welcher zwischen zwei Bauteilen mit einberechnet werden muss, um ein einfaches Zusammenstecken zu gewährleisten.

**CornerRadius**

Der Parameter „CornerRadius“ entspricht der Konstante, welche den Radius des Grundzylinders der Eckstücken angibt.

**PinMinLength**

Der Parameter „PinMinLength“ entspricht der Konstante, welche die minimale Länge des Quaders des positiven Eckstücks angibt, welcher zwischen dem Eckzylinder und dem Pinzylinder platziert wird.

**PinPWidth**

Der Parameter „PinPWidth“ entspricht der Konstante, welche die Weite für den Quader des positiven Eckstücks angibt, welcher zwischen dem Eckzylinder und dem Pinzylinder platziert wird.

**PinPRadius**

Der Parameter „PinPRadius“ entspricht der Konstante, welche den Radius des Pinzylinders des positiven Eckstücks angibt.

**PinDistance**

Der Parameter „PinDistance“ entspricht der Konstante, welche die Distanz zwischen dem positiven Pin und den anliegenden Wandstücken angibt, welche für jeden Pin eingehalten werden muss.

**Height**

Der Parameter „Height“ entspricht der Konstante, welche die Höhe der Wandteile und der Eckzylinder angibt.

**PinHeight**

Der Parameter „PinHeight“ entspricht der Konstante, welche die Höhe des positiven Pinzylinders angibt.

**BasePlateHeight**

Der Parameter „BasePlateHeight“ entspricht der Konstante, welche die Höhe der Grundplatte angibt.

**BasePlatePinCircleHeight**

Der Parameter „BasePlateCircleHeight“ entspricht der Konstante, welche die Höhe der Kreisflächen angibt, die unter den positiven Eckstücken angebracht werden und der Stabilisierung und Verankerung von Grundplatter und Eckstück dienen.

### 3.3.5 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 Länge, Höhe und Breite erstellen der dann wie folgt mit `Cube.toString()` in einen String konvertiert wird: `cube([Länge, Breite, Höhe]);`

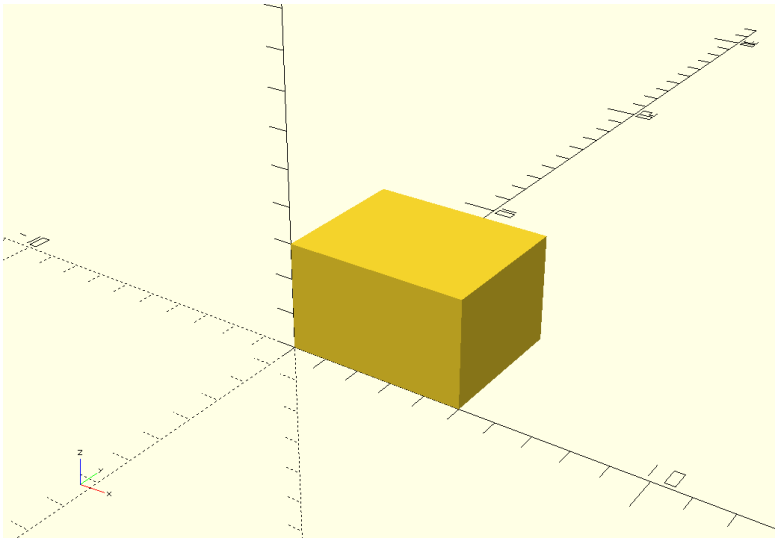

Abbildung 11: Resultat der Eingabe: `new Cube(3, 4, 5).toString()`
(Screenshot der Verfasser)

## 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. Um die Dimensionen der Grundplatte effektiv auszunutzen, unterteilen wir die Grundplatten, welche aufgrund ihrer Größe nicht in einem Durchgang druckbar sind. Außerdem werden einzelne Elemente des Modells zu Objektgruppen zusammengefasst, die nebeneinander gedruckt werden können.

### 3.4.1 Aufteilen der Grundplatten

Beschreiben, wie das gemacht wird

### 3.4.2 Verteilung der Bauteile

Beschreibung der Verteilung einzelner Bauteile für passenden Druckvorgang

# 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.

Veraltet; Überarbeiten!

# Literaturverzeichnis

[1] de.wikipedia.org/wiki/Gau%C3%9Fsche_Trapezformel (Stand: 21.03.2017, 12:00 Uhr)

[2] cs.sfu.ca/~binay/813.2011/DCEL.pdf (Stand: 25.04.2017, 12:00 Uhr)

[3] en.wikipedia.org/wiki/Doubly_connected_edge_list (Stand: 25.04.2017, 12:00 Uhr)

[4] en.wikibooks.org/wiki/OpenSCAD_User_Manual (Stand: 25.04.2017, 12:00 Uhr)

[5] docs.oracle.com/javase/8/docs/api/index.html (Stand: 25.04.2017, 12:00 Uhr)

[6] kabeja.sourceforge.net/ (Stand: 12.10.2017, 10:00 Uhr)

[7] http://www.journaldev.com/864/java-open-file (Stand: 28.05.2017, 17:15)

[8] https://en.wikipedia.org/wiki/AutoCAD (Stand: 05.06.2016, 18:00)

[9] https://en.wikipedia.org/wiki/OpenSCAD (Stand: 05.06.2017, 18:00)

[10] https://eu.makerbot.com/fileadmin/Inhalte/Support/Manuals/Germar (Stand: 06.06.2017, 09:00)

# Abbildungsverzeichnis

# Dank an die Betreuer

Danke schoin!

# 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

Datum ändern

## 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