Todo list

Johann wollte DCEL noch mal überarbeiten 8
Hier könnte man noch was zu sagen 17
code eventuell bilder definitiv 18
keene Ahnung wie ich das mit der repräsentativen Liste formulieren soll 18
man könnte das im code auch vorher tun denke ich 19
CornerPin 24

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

2016/2017

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“

30 Seiten, Y Anlagen, 4 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. Die Berechnungen laufen so ab, dass an allen Elementen komplementäre Stecker angebracht werden, die zusammen als ein Stecksystem fungieren. Eckpfeiler dienen hierbei als Verbindungsstücke zwischen den Wänden und Bodenplatten die somit für die Stabilität des Objektes sorgen. Mit dem Stecksystem kann der gesamte 3D-Druck aus seinen einzelnen Bestandteilen aufgebaut werden. Dadurch entsteht ein Modell, welches aufgrund der genannten Modifikationen transportabel und geeignet für Präsentationen ist.

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

Johann Bartel und Peter Oehme

Inhaltsverzeichnis

1 Einleitung **6**

2 Grundlagen **8**

2.1 Planare Graphen 8

2.2 Doubly connected edge list 8

2.3 AutoCAD 9

2.4 OpenSCAD 9

2.5 3D-Drucker MakerBot Replicator™ 2 10

3 Vorgehen zur Problemlösung **12**

3.1 Einlesen des Grundrisses 12

3.1.1 Funktionsweise der Bibliothek *kabeja* 12

3.1.2 Funktionsweise der GUI 13

3.2 Erstellen der DCEL 16

3.2.1 Graph Klasse 16

3.2.2 Line-to-Edge Konvertierung 16

3.2.3 Zwillingskantengenerierung 17

3.2.4 Nachfolger- und Vorgängerermittlung 17

3.2.5 Flächenerstellung 18

3.2.6 Vervollständigung der Knoten 19

3.3 Aufbau der Einzelbauteile 20

3.3.1 Zuweisen von Berechnungskonstanten 20

3.3.2 OpenSCAD Java Interface 23

3.3.3 Corner 24

3.3.4 Wall 25

3.3.5 BasePlate 25

3.4 Druckvorgang 26

4 Ausblick **27**

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 (1) und Kanten (2) dargestellt werden kann, ohne dass sich zwei oder mehr Kanten schneiden. Jede Fläche (3) 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 äußerstes Gebiet genannt.

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

Diese zwischenobjektlichen Referenzierungen ermöglichen es, ausgehend von einem Element ohne umfangreiche Berechnungen auf alle anderen Objekte zu schließen, indem bei Knoten und Flächen die zugehörigen Kanten, beziehungsweise bei Kanten deren Vorgägner 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 wurden das Dateiformat „.dxf“ entwickelt, welches als Industriestandard zum Austausch von CAD-Dateien dient.
Der Grundriss, welcher als Ausgangspunkt dieser Arbeit fungiert, ist in AutoCAD erstellt wurden und wird dem zu erstellenden Programm in Form einer .dxf-Datei bereitgestellt.

2.4 OpenSCAD

OpenSCAD ist eine freie CAD-Modellierungssoftware, welche auf Basis einer textbasierten Beschreibungssprache 3D-Modelle erzeugt. OpenSCAD bietet dabei verschiedene Vorteile während des Modellierungsvorganges, wie beispielsweise dem farbigen Hervorheben oder der Modularisierung bestimmter Objekte.
Die Modellierung von einfachen Basisobjekten in OpenSCAD erfolgt durch das Verwenden von bestimmten Schlüsselwörtern wie `cube()`, `sphere()` oder `cylinder()` und dem anschließenden Übergeben von Parametern in Klammern. Diese Basisobjekte werden 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, um neue und komplexere Objekte nach eigenen Ansprüchen zu erhalten. Neben solchen einfachen Objekten, wird außerdem die Möglichkeit geboten, komplexere Objekte wie Polygone (`polygon()`) zu erstellen und diese dann ausgehend vom zweidimensiona-

len 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 Programm erhaltenen Anweisungen in .scad-Dateien zu speichern, welche dann 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 es von Haus aus die Option zur Konvertierung vom .scad-Format zu .stl mitliefert.

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 aufgetragen und so auch 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 ga-

rantieren, 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 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 Fläche von 28,5 $\times$ 15,3 cm und einer maximalen Höhe von 15,5 cm bedruckbar ist. Entsprechend dieser möglichen Maße sollten also alle Objekte 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

In dieser Anwendung wird eine Funktion der Klasse „DXFReader“ verwendet, welche den Pfad zur .dxf-Datei als Parameter übergeben bekommt. Aus dieser Datei werden dann 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 für die Umwandlung des Graphen in die DCEL verwendet werden.

3.1.2 Funktionsweise der GUI

Die GUI dient dem Nutzer als Schnittstelle, mit der er die .dxf-Datei auswählt, die er umwandeln möchte. Die GUI setzt sich hierbei zusammen aus einem `JFrame`, indem ein `JTextField`, ein `FileChooserButton` mit der Aufschrift „Choose a file...“, ein `StartButton` mit der Aufschrift „Start the conversion“ und ein `ShowResultButton` mit der Aufschrift „Show the text file“ platziert sind.

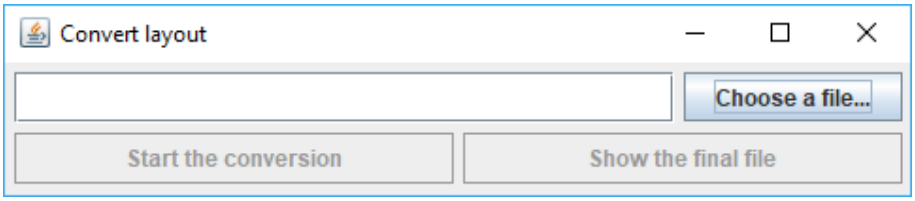

Abbildung 1: Ausgangszustand der GUI (Screenshot der Verfasser)

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`.

```
1  JFileChooser fc = new JFileChooser();
2
3  int returnVal = fc.showOpenDialog(Main.getGui());
4  if (returnVal == JFileChooser.APPROVE_OPTION) {
5        File file = fc.getSelectedFile();
```

```
6         Main.getGui().getFilePathField().setText(file.
              getAbsolutePath());
7   }
```

Codeauschnitt 2: `JFileChooser` zum Öffnen von Dateien

In diesem `JFileChooser` ist außerdem ein `FileFilter` implementiert, der dem Nutzer lediglich .dxf-Dateien anzeigt.

```
1  fc.setFileFilter(new FileFilter() {
2         public String getDescription() {
3               return "DXF Files (*.dxf)";
4         }
5         public boolean accept(File f) {
6               if (f.isDirectory()) {
7                     return true;
8               } else {
9                     String filename = f.getName().toLowerCase();
10                    return filename.endsWith(".dxf");
11              }
12        }
13 });
```

Codeauschnitt 3: `FileFilter` innerhalb des `JFileChooser`-Dialogs

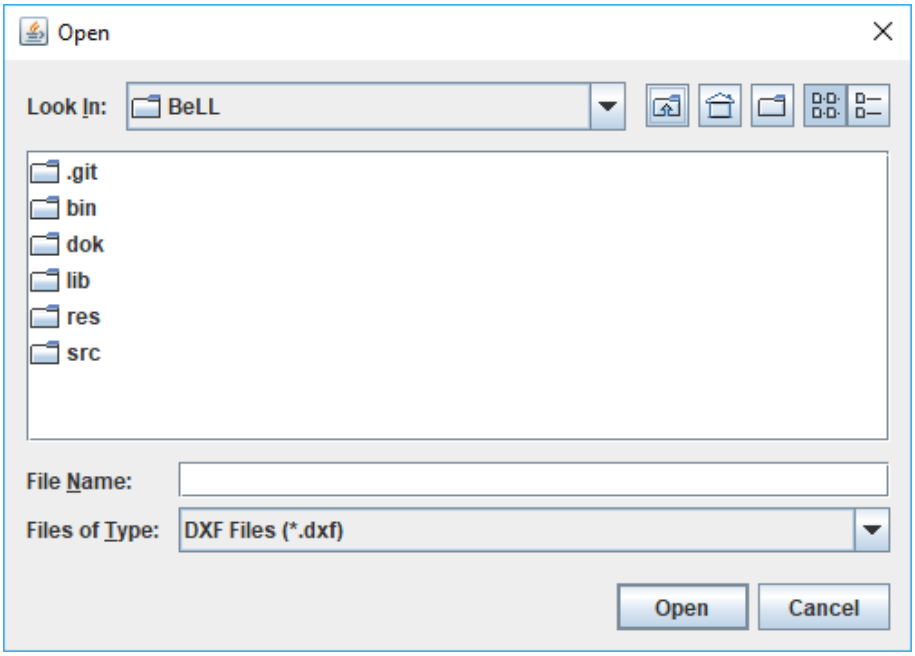

Abbildung 2: Dateiöffnungsdialog (Screenshot der Verfasser)

Alternativ zum `JFileChooser`-Dialog kann der Nutzer auch direkt in das `JTextField` den Dateipfad eingeben.

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

```
1 if (returnVal == JFileChooser.APPROVE_OPTION) {
2         File file = fc.getSelectedFile();
3         Main.getGui().getFilePathField().setText(file.
              getAbsolutePath());
4         Main.getGui().getStartButton().setEnabled(true);
5         Main.startConversion(file.getAbsolutePath());
6 }
```

Codeauschnitt 4: Aktivierung des `StartButton`

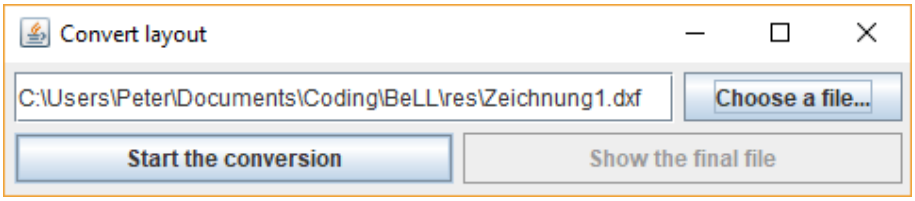

Abbildung 3: GUI mit aktiviertem `StartButton` (Screenshot der Verfasser)

3.2 Erstellen der DCEL

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 die Start- und Endpunkte der eingelesenen Linien in Knoten ohne eine Referenz auf eine anliegende Kante umgewandelt und diese als Start- und Endknoten der entsprechenden Kante gesetzt. 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 an Knoten hinzu, wenn er noch nicht in ihr existiert.

```
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 5: 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 6: `createNode()` Funktion

Hier könnte man noch was zu sagen

3.2.3 Zwillingskantengenerierung

Um aus den entstandenen Kanten die invertierten Gegenstücke, auch als „Zwillingskanten“ bezeichnet, zu erhalten, werden alle Kanten, die in der Liste bereits vorhanden sind, betrachtet und neue Kanten hinzugefügt, die im Vergleich zu den ursprünglichen Kanten vertauschte Start- und Endknoten besitzen. Direkt nach dem Hinzufügen der neuen Kante 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.

```
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 7: Hinzufügen der Zwillingskanten

3.2.4 Nachfolger- und Vorgängerermittlung

Das Erstellen der Referenzen werden zuerst alle ausgehenden Edges der Nodes, das heißt alle Edges, die den jeweiligen Node als ihren Startnode besitzen, in einer zweidimensionalen ArrayList gespeichert. Die erste Dimension steht für den Index des Nodes in der erstellten Nodeliste für den

in der zweiten Dimension die jeweiligen ausgehenden Edges vorliegen. Da diese durch eine `for()` Schleife mit der oben stehenden Bedingung herausgesucht werden, sind die Edges im Array in zufälliger Reihenfolge, also nicht nach der Anordnung am Node gegenwärtig. Jetzt werden die Edges anhand des `atan2()` Winkel am vorliegenden Node im mathematisch positiven Drehsinn sortiert. Daraus ergibt sich, dass das vorherige bzw. nachfolgende Element einer Edge die Edge, die „links“ bzw. „rechts“ der Betrachteten liegt darstellt.
Für jede ausgehende Edge `e` können nun folgende Referenzen gesetzt werden:

- Das vorherige bzw. letzte Element der ArrayList, wenn die betrachtete Edge den Index 0 hat, stellt den Nachfolger der Twinedge von `e` dar.
- Der Twin der in der ArrayList nachfolgenden Edge bzw. ersten Edge, wenn die betrachtete Edge das letzte Element ist, ist der Vorgänger von `e`

Genannte Referenzen werden nun gesetzt, sodass die Verknüpfungen zwischen den Edges fertiggestellt sind.
code eventuell bilder definitiv

3.2.5 Flächenerstellung

Durch eine Schleife können die einzelnen Flächen herausgefiltert werden. Zuerst wird eine Boolean-ArrayList mit der selben Länge der Edgeliste erstellt, welche die Indices der schon behandelten Kanten repräsentiert. Folglich besteht die Liste anfangs nur aus `false` Werten. Fortlaufend wird von einer anfänglichen Kante
keene Ahnung wie ich das mit der repräsentativen Liste formulieren soll

Ab diesem Schritt existieren alle Flächen vollständig in der Flächenliste. Das unendliche bzw. äußere Gebiet lässt sich

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 sollen, sind wie folgt festgelegt:

- Die Einzelteile sollen möglichst simpel gestaltet sein, um unnötig komplizierten Konflikten
- 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 des Einblickes willen eine möglichst stabile Einheit bilden

Aus diesen Kriterien resultiert die Verwendung von herausnehmbaren Wandstücken, welche nicht zu fest im restlichen Modell verankert sind, so dass sie sehr leicht herausgenommen und auch wieder hineingesetzt 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.

3.3.1 Zuweisen von Berechnungskonstanten

Funktionsweise 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 Instanzierung der `Params`-Klasse deren Parameter gesetzt oder auf bereits vor-

handene 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, double
            CornerRadius, double PinMinLength, double PinPWidth,
            double PinPRadius, double PinDistance, double Height,
             double PinHeight, double BasePlateHeight, double
            BasePlatePinCircleHeight){
2              e = E;
3              cornerRadius = CornerRadius;
4              pinMinLength = PinMinLength+CornerRadius;
5              pinDistance = PinDistance;
6              height = Height;
7              pinHeight = PinHeight;
8              pinPRadius = PinPRadius;
9              pinPWidth = PinPWidth;
10             basePlateHeight = BasePlateHeight;
11             basePlatePinCircleHeight = BasePlatePinCircleHeight
                   ;
12      }
```

Codeauschnitt 8: 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 9: 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 (ϵ/Epsilon)

Der Parameter „E“ entspricht der Konstante ϵ (Epsilon), welcher aus Gründen der vorteilhaften Kürze der Parameternamen hier verwendet wurde. ϵ 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.

3.3.2 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 Länge, Höhe und Breite erstellen

der dann wie folgt mit `Cube.toString()` in einen String konvertiert wird:

```
cube([Länge, Breite, Höhe]);
```

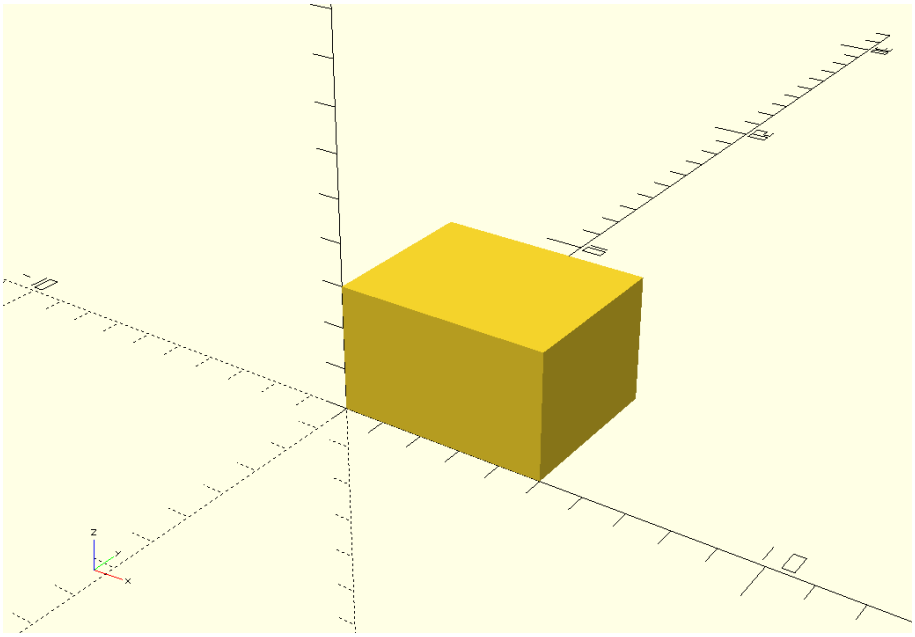

Abbildung 4: Ergebnis von `new Cube(3, 4, 5).toString()` (Screenshot der Verfasser)

3.3.3 Corner

Das Corner Element bezeichnet die Eckpfeiler des 3D-Modells. Es besteht aus zwei zusammengefügten Teilen, dem CornerCylinder und dem CornerPin. CornerCylinder stellt den oberen Teil einer Ecke dar, der ein rundes Grundbauteil mit Einkerbungen für Wände bereitstellt. 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 vollzogen.

CornerPin

3.3.4 Wall

3.3.5 BasePlate

3.4 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.

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

1 Ausgangszustand der GUI (Screenshot der Verfasser) 13
2 Dateiöffnungsdialog (Screenshot der Verfasser) 15
3 GUI mit aktiviertem `StartButton` (Screenshot der Verfasser) 15
4 Ergebnis von `new Cube(3, 4, 5).toString()` (Screenshot der Verfasser) 24

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.

Ort, Datum, Unterschrift

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.

Ort, Datum, Unterschrift