Erstellung einer Webanwendung zur Bearbeitung von 3D-Objekten und Zuweisung von Bearbeitungsschritten

Im Kontext der modularen robotergestützten Prozessführung für den Innenbereich im Baugewerbe („Bau-Robo“)

Master Projektarbeit

Studiengang Geoinformatik und Vermessung
Hochschule Mainz
Fachbereich Technik
Lehreinheit Geoinformatik und Vermessung

Studierende: Rebekka Lange, Lisa Mosis
Betreuer: Prof. Dr.-Ing. Frank Boochs,
Prof. Dr.-Ing. Anita Sellent,
Dr. Jean-Jacques Ponciano,
Jonas Veller M.Sc.

Mainz
Februar 2021

Kurzzusammenfassung

Gegenstand dieses Masterprojektes ist es, eine Webanwendung zu entwickeln, die es dem Benutzer erlaubt, eine Szene aus 3D-Objekten in einem Viewer zu betrachten und zu bearbeiten. Hierbei wird mit dem Framework „Spring“ eine REST-API erstellt, die eine serverseitige Datenhaltung für „.ply-files“ in Java mit einer browserseitigen HTML-Seite verbindet, in der die 3D-Objekte in einem „Three.js“ Viewer angezeigt und bearbeitet werden können.

Schlagwörter: GUI, Benutzeroberfläche, Java, HTML, Webapp, Three.js, Spring-Framework, 3D-Viewer, Softwareentwicklung

Abstract Summary

Subject of this master’s project is developing a web application that allows for a user to view and alter a scene of 3D models. In this process a REST-API is built with the “Spring” framework, that connects server-side data management in Java with a browser-side HTML site that contains a “Three.js” viewer, where 3D objects can be viewed and altered.

Keywords: GUI, graphical user interface, Java, HTML, webapp, Three.js, Spring framework, 3D viewer, software development

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der industriellen Fertigung ist es üblich, das Sprühen von Lacken und Farben, sowie das Schleifen von Objekten, von Robotern durchführen zu lassen, da diese Schritte häufig eine gesundheitliche Belastung für Menschen darstellen können. Jedoch werden eben diese Schritte in der Malerbranche häufig noch von Hand ausgeführt. Gerade da in dieser Branche der Nachwuchs zurück geht, existiert hier ein Bedarf an robotergestützten Systemen für die genannten Aufgaben.

1.1 Hintergrund

Dieses Masterprojekt ist eingebettet in ein größeres Projekt des i3mainz (i3mainz) zur „Entwicklung einer modularen robotergestützten Prozessführung für den Innenbereich im Baugewerbe („Bau-Robo“)“ (Projektbeschreibung 2020). Im Zuge dieses Projektes sollen Malerbetriebe und Stuckateure durch den Einsatz von Robotern beim Schleifen von Wänden und Sprühen von Farben und Lacken unterstützt werden. Hierfür wird eine Benutzeroberfläche (GUI) benötigt, in der ein vorhandenes Modell der Gegebenheiten vor Ort überprüft und verbessert werden kann. Ebenso müssen die weiteren Arbeitsschritte, also das Auftragen von Farben oder Einstellungen für einen Schliff, festgelegt werden. Die GUI wird innerhalb dieser Arbeit erstellt und erklärt und bildet die Grundlage, auf der das gesamte System im weiteren Verlauf aufbauen wird. Aufgrund der Pionierstellung dieses Teilprojektes sind dabei einige Rahmenbedingungen, wie die Auswahlmöglichkeiten bestimmter Arbeitsschritte, noch unklar. Deshalb wird in diesem Teilprojekt zunächst eine Grundlage geschaffen, die leicht anpassbar ist und zunächst die Grundstruktur aus einer Web-Anwendung mit 3D-Viewer zum Bearbeiten einer 3D-Ansicht erstellt. Hierbei werden bisher undefinierte Elemente beispielhaft erstellt und leicht bearbeitbar und ergänzbar gehalten.

1.2 Zielsetzung

Ziel dieser Projektarbeit soll es sein, eine grafische Benutzeroberfläche (GUI) aufzubauen. Diese soll die Möglichkeit zur Kontrolle und Korrektur von automatisch generierten Objekten aus einer Punktwolke beinhalten. Dazu sollen eine Punktwolke und daraus generierte Objekte in einer GUI angezeigt und bearbeitet werden können. Ebenso soll es möglich sein, für einzelne Objekte Bearbeitungsschritte auszuwählen, welche später von einem Roboter umgesetzt werden können. Die GUI soll von einem Tablet bedienbar sein.

2 Anforderungen

Die Aufgabenstellung ist der Projektbeschreibung des Gesamtprojektes „Entwicklung einer modularen robotergestützten Prozessführung für den Innenbereich im Baugewerbe („Bau-Robo“)“ (Projektbeschreibung 2020) entnommen. Ausgehend von dieser wurde zu Beginn des Projektes, in Zusammenarbeit mit den Betreuern, eine Anforderungsliste für das Projekt erstellt. Diese wurde während des Projektes als Ausgangspunkt verwendet und nach Rücksprache mit den Betreuern in manchen Fällen angepasst.

Die gesamte Liste der Anforderungen lässt sich in Frontend, Backend und allgemeine Anforderungen unterscheiden. Ausgehend davon lassen sich weitere Arbeitsfelder ableiten. Zunächst die notwendigen Funktionen des Backends:

- Datenhaltung auf einem Server
- Framework zur Nutzung einer Webseite
- Objekte und Punktwolke nach Anfrage an Frontend übergeben
- Objekte annehmen und speichern, bei Bedarf bereits bestehende Daten ändern

Ebenso die Arbeitsfelder für das Frontend:

- HTML-Gerüst zur Darstellung und Navigation einer Webseite
- Viewer zur Darstellung von Daten
- Editor zur Bearbeitung von Daten

Beim Frontend wurde sich mit den Betreuern darauf geeinigt, dass es zwei Viewer geben soll. Einen zur Überprüfung und Verbesserung der aus einer Punktwolke generierten Objekte und einen zur Festlegung von Arbeitsschritten für den Roboter. Dies war das Ergebnis einer Untersuchung der verschiedenen Anforderungen an die Aufgaben. Es wurde sich für eine Lösung entschieden, bei der in einem HTML-Skript zwischen zwei Reitern ausgewählt werden kann, welche Bearbeitung vorgenommen werden soll. So bleiben die Funktionen in einer Anwendung und trotzdem funktionell getrennt. Der erste Editor, genannt Viewer1, soll die Korrektur der generierten Objekte beinhalten. Im zweiten Editor, genannt Viewer2, soll die Auswahl der Bearbeitungsschritte für den Maler-Roboter stattfinden. Die Unterschiede sind mit dem entsprechenden Editor-Namen gekennzeichnet. Für das Frontend folgende Anforderungen gelten:

- Erkannte Objekte laden und farblich klassifiziert darstellen
- Viewer1: Punktwolke laden
- Navigation
    - In der Szene bewegen, zoomen, drehen
    - Objekt durch Anklicken / Antippen auswählen und farblich hervorheben
- In Viewer1: Punktwolke nicht auswählbar
- Objekteigenschaften anzeigen
- Bearbeitungsoptionen anzeigen
- Bearbeitung Viewer1
    - Neue Objekte (mit wählbaren Eigenschaften und Position) der Szene hinzufügen

        - Objekte löschen
        - Geometrie oder Position eines ganzen oder Teil-Objektes verändern
        - Klassifikation oder Attribut eines Objektes ändern
    - Bearbeitung Viewer2
        - Bearbeitungsschritte für Objekt auswählen (z.B. „Schleifen“, „Streichen in Rot“)
        - Beliebig viele Bearbeitungsschritte möglich machen (Liste)
        - Widersprüchliche Arbeitsschritte klären (z.B. „Streichen in Rot“ und „Streichen in Blau“)
    - Veränderungen speichern
    - Zwischenschritte speichern und „Rückgängig“-Funktion einfügen

Zu den allgemeinen Anforderungen schließlich zählen folgende Punkte:

- Anwendung soll auf Tablet ohne Performance- oder Bedienungseinschränkungen laufen
- Die gesamte Darstellung und Bedienung soll benutzerfreundlich sein
- Die abgespeicherten und verwendeten Daten sollen ein einheitliches Datenformat besitzen und die Schnittstellen sollen klar definiert sein
- Für alle Änderungen sollen Sicherungen eingebunden werden, um bei Programmabsturz Daten zu behalten und die Möglichkeit zur Revidierung des letzten Arbeitsschrittes zu bieten

3 Wahl der Software

Zu Beginn des Projektes wird zunächst festgelegt welche externen Libraries eingebunden werden können, um den Programmieraufwand auf die spezifischeren Aufgaben zu beschränken. Dazu werden verschiedene 3D-Viewer zur Darstellung und Änderung der räumlichen Gegebenheiten und auch mehrere Frameworks zur Nutzung eines Servers betrachtet. Im Folgenden werden die verschiedenen Alternativen genauer betrachtet.

3.1 3D-Viewer

Der 3D-Viewer übernimmt die Aufgabe die Objekte in 3D anzuzeigen und zu bearbeiten. Hierbei existieren mehrere Möglichkeiten, die verschiedene Vor- und Nachteile haben. Dabei wird vor allem darauf geachtet, ob die Möglichkeit besteht Objekte und Punktwolken anzuzeigen und ob Objekte bearbeitet werden können.

3.1.1 Xeokit-Viewer

Der Viewer Xeolabs von Xeokit ist eine Lösung für das Darstellen von BIM (Building Information Modelling) Modellen. Diese können aus verschiedenen Dateiformaten geladen und angezeigt werden. Für diese Aufgaben ist der Viewer gut ausgestattet und bietet eine saubere Darstellung. Für dieses Projektes ist der Viewer jedoch ungeeignet, da er keine Punktwolken anzeigen kann und die Modelle statisch lädt, sodass keine Veränderungen vorgenommen werden können. Ebenfalls ist die Software zwar für akademische Zwecke kostenlos, jedoch nicht für kommerzielle Zwecke. (Xeolabs 2020)

3.1.2 Potree-Viewer

Potree ist ein kostenloser open-source Viewer zum Anzeigen großer Punktwolken im Browser. Er ist basierend auf WebGL und Three.js programmiert und bietet eine flüssige Darstellung. Der Viewer ist nicht darauf ausgelegt Linien, Flächen oder Objekte anzuzeigen, wofür aber eventuell auf die zugrunde liegenden Three.js Library zugegriffen werden kann. Aufgrund anfänglicher Schwierigkeiten bei einigen Versuchen die Software einzubinden und zu nutzen, und da zu Beginn davon ausgegangen wurde, dass die Punktwolken nicht sehr groß sein werden, wird dieser Viewer jedoch nicht genutzt. (Schütz 2020)

3.1.3 GIScene-Viewer

GIScene ist ein sehr einfacher Viewer, der unter der MIT-Lizenz kostenlos nutzbar ist. Er wird allerdings seit 5 Jahren nicht mehr weiterentwickelt und stellt nicht alle nötigen Funktionen bereit, weshalb er hier nicht weiter betrachtet wurde. (Auer 2015)

3.1.4 Three.js

Three.js ist eine JavaScript Bibliothek zur Erstellung und Darstellung von animierten 3D-Grafiken. Sie bietet umfassende Funktionen zur webintegrierten Darstellung und dynamischen Bearbeitung von geometrischen Objekten und deren Eigenschaften. Dabei sind

die Funktionen effizient konzipiert und verursachen somit keine Performance-Einschränkungen bei großen Datenmengen oder Arbeitsschritten.

Die Bibliothek steht über GitHub zum kostenlosen Download und dynamischem Import zur Verfügung und darf für den privaten und kommerziellen Gebrauch verwendet werden. (Three.js 2021)

Es bestanden bereits Vorkenntnisse und Three.js erfüllt voraussichtlich alle Anforderungen. Deshalb fiel die Entscheidung auf diese Bibliothek.

3.2 Frameworks

Um Daten und das Programm auf einem Server zu halten und von einem beliebigen Gerät darauf zugreifen zu können, wird ein Framework zum Ausführen von eingebundenen HTML-Skripten und der Kommunikation zwischen Server und Endgerät verwendet. Es bietet die Möglichkeit Webseiten zu laden und Daten- und Informationsaustausch zwischen Frontend und Backend über eine REST-Schnittstelle zu ermöglichen.

3.2.1 Spring-Framework

Spring ist ein sehr häufig verwendetes Framework, das mit einer großen online-community punktet. Es ist mithilfe von Springboot einfach aufzusetzen und bietet mehrere Möglichkeiten mit Webanfragen umzugehen. Da schon einige Vorkenntnisse zu diesem Framework vorliegen und es die Anforderungen erfüllt, wird es für dieses Projekt genutzt. (VMware 2021)

3.2.2 Play-Framework

Das Play-Framework ist ein einfaches und anfängerfreundliches Framework, das es erlaubt einen Server mit Webseiten zu versorgen und Webanfragen verarbeitet. Es ist unter anderem für Java konzipiert und bietet Möglichkeiten zur Integration in IDEs wie IntelliJ IDEA und Eclipse. Da im Vorfeld keine Vorteile gegenüber Spring deutlich werden, wird weiterhin dieses bevorzugt. (Lightbend 2018)

3.2.3 Dropwizard-Framework

Dropwizard ist ein Java-Framework und basiert auf Open Source Software. Da im Vorfeld keine Vorteile gegenüber Spring deutlich werden, wird dieses stattdessen verwendet. (Dropwizard 2020)

4 Oberfläche

In diesem Kapitel wird der Aufbau der grafischen Benutzeroberfläche (GUI) beschrieben. Die Oberfläche ist in vier einzelne aber miteinander verbundene Webseiten aufgeteilt. Die Seiten sind ein Startmenü (Home), ein Viewer für das Prüfen und Korrigieren der generierten Objekte (Viewer1), ein Viewer zum Festlegen von Arbeitsschritten einzelner Objekte (Viewer2) und ein Bereich, welcher Informationen zum Projekt und damit verbundenen Ressourcen enthält (Informationssymbol/Impressum).

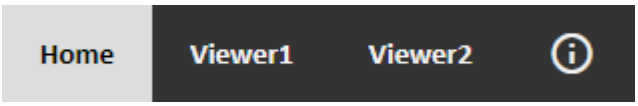

Abbildung 1: Auswahl Webseite

In Abbildung 1 ist die Navigation mit den vier Auswahlmöglichkeiten Home, Viewer1, Viewer2 und Informationen zu sehen, welche in allen Webseiten gleich ist. Ein Klick auf eines der Felder führt auf die jeweilige Seite. Die graue Hinterlegung zeigt dabei an, welche Seite aktuell aufgerufen ist.

4.1 Home

Der erste Bereich ist das Startmenü oder auch Home. Dieser dient zur Dateiauswahl für die Bearbeitungen in den beiden Viewern. Abbildung 2 zeigt den Aufbau der Seite. Da nur der linke Bereich gefüllt ist, zeigt Abbildung 3 den relevanten Teil vergrößert.

Abbildung 2: Home

Auf der Home-Seite wird die Datengrundlage für die weiteren Seiten festgelegt. Ohne eine Dateiauswahl in diesem Bereich kann in den Viewern keine Bearbeitung stattfinden. Zur Dateiauswahl gehören zwei wichtige Komponenten. Dies sind eine Punktwolke und die zuvor daraus generierten Objekte. Beide werden auf der Home-Seite ausgewählt. Um die Objekte anzeigen zu können, muss auf dem Server eine bestimmte Datenstruktur eingehalten werden. So müssen sich alle Daten im Programmverzeichnis

„GUI/data“ befinden. Für jedes Projekt muss ein eigener Ordner angelegt sein, welcher eine Datei mit der Punktwolke und einen Unterordner mit allen Objekten enthält. Ist diese Struktur eingehalten, kann ein Projekt zur Bearbeitung auf der Home-Seite ausgewählt werden (vgl. 5.4 Backend). Alle wichtigen Ebenen sind in Abbildung 3 farblich markiert und jeweils in der gleichen Farbe beschriftet.

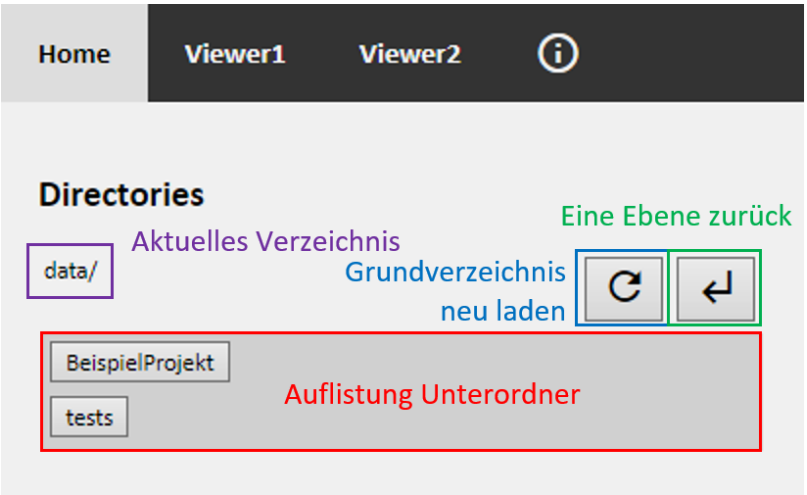

Abbildung 3: Home - Auswahlmöglichkeiten

Wenn die Seite aufgerufen wird, wird das aktuelle Verzeichnis zunächst auf „data“ gesetzt und wird unter „Directories“ angezeigt. In der Darstellung ist dieser Bereich lila markiert. Darunter befindet sich eine Auflistung der Projektordner (rot umrandet) in diesem Verzeichnis. Indem auf eines geklickt wird, werden stattdessen die Unterordner davon angezeigt. Dies sollten ein Unterordner „objects“ und eine Punktwolkendatei „pointcloud.ply“ sein. Der in Abbildung 3 grün markierte Pfeil führt eine Ebene zurück. Mit dem blau markierten Kreispfeil kann das Verzeichnis aktualisiert werden.

4.2 Viewer1

Viewer1 dient zur Kontrolle und Korrektur von Objekten, welche aus einer Punktwolke generiert wurden. Diese sind in Home angegeben. Im Reiter Viewer1 werden die Punktwolke und die Objekte in einem Viewer dargestellt. Möglichkeiten zur Änderung befinden sich links vom Viewer. Am unteren Rand befindet sich eine Statusleiste. Diese zeigt den Status des letzten Bearbeitungsschrittes und auftretende Exceptions an. Rechts oben in der gleichen Zeile wie die Seiten-Auswahl befindet sich ein weißer Pfeil nach links. Dieser dient dazu, bereits gespeicherte Änderungen wieder rückgängig zu machen. Abbildung 4 zeigt den Aufbau von Viewer1.

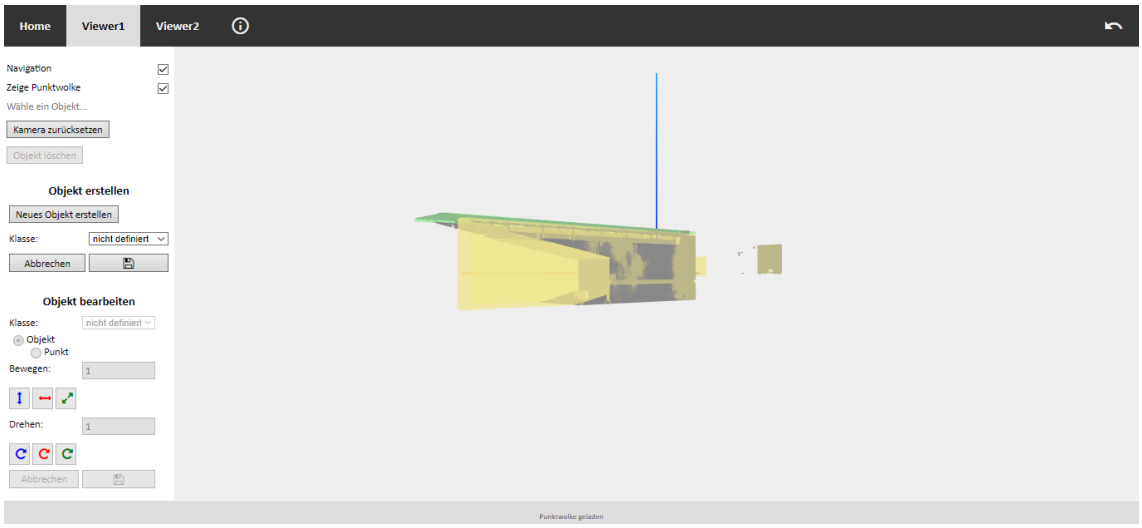

Abbildung 4: Viewer1 Überblick

Im linken Bildbereich befinden sich die restlichen Bearbeitungstools. Diese sind in drei Unterkategorien eingeteilt: allgemeine Funktionen, Optionen zur Objekterstellung und Optionen zur Objektbearbeitung. Alle Tools sind in Abbildung 5, Abbildung 6 und Abbildung 9 zu sehen.

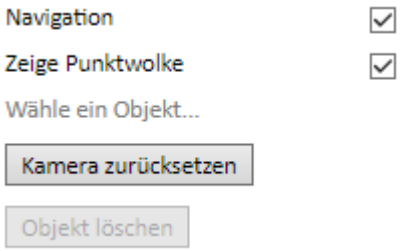

Abbildung 5: Viewer1 - allgemeine Funktionen

Die allgemeinen Funktionen beginnen mit der Auswahloption, die Navigation im Viewer ein- oder auszuschalten.

Darunter befindet sich die Option „Zeige Punktwolke“. Diese bietet die Option die Punktwolke ein- oder auszublenden.

Die nächste Zeile ist in Abbildung 5 mit „Wähle ein Objekt…“ beschriftet. Dies ist lediglich ein Informationsfeld. Wird ein Objekt ausgewählt, werden in diesem Feld die ID und die Klasse des Objektes angezeigt. Die ID entspricht dem Dateinamen dieses Objektes.

Die Funktion „Kamera zurücksetzen“ folgt als nächstes. Wenn dieser Button angeklickt wird, wird die Kamera so rotiert, dass die seitliche Verkippung der z-Achse wieder zurückgesetzt wird.

Als letzte allgemeine Funktion kommt „Objekt löschen“. Ist kein Objekt ausgewählt, ist dieser Button deaktiviert. Mit diesem kann ein ausgewähltes Objekt aus dem Viewer

und dem Server gelöscht werden. Mit dem „Rückgängig-Button“ oben rechts in Abbildung 4 kann diese Aktion rückgängig gemacht und das Objekt und dessen Datei wieder hergestellt werden.

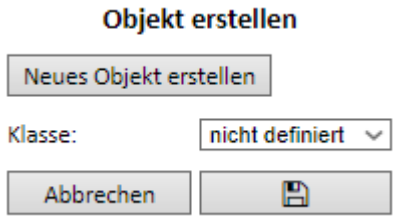

Abbildung 6: Viewer1 - Neues Objekt erstellen

Nach den allgemeinen Funktionen folgen die Tools zum Erstellen eines neuen Objektes. Abbildung 6 zeigt die Funktionen. Falls ein Objekt in der Punktwolke erkannt wird, welches nicht bei den Objekten dabei ist, kann ein Neues erstellt werden. Zunächst ist dies auf zweidimensionale Objekte beschränkt. Mit dem Button „Neues Objekt erstellen“ wird der Editiermodus gestartet. Nun können in der Punktwolke mit Doppelklick die neuen Eckpunkte für das Objekt ausgewählt werden, die dann rot dargestellt werden. Zwischen den Punkten werden rote Linien gezeigt, um die Objektumrandung sichtbar zu machen (Abbildung 7).

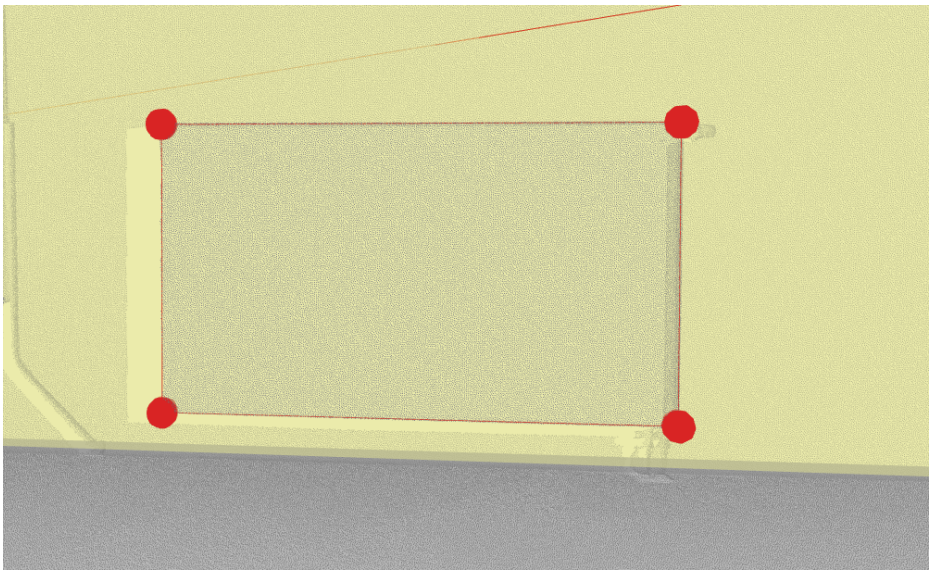

Abbildung 7: Erstellen eines neuen Objektes

Um das Objekt zu speichern, kann noch eine Klasse für das neue Objekt ausgewählt werden. Hinter der Beschriftung „Klasse:“ befindet sich eine Liste mit verfügbaren Klassen, aus denen die passende ausgewählt werden kann (Abbildung 8).

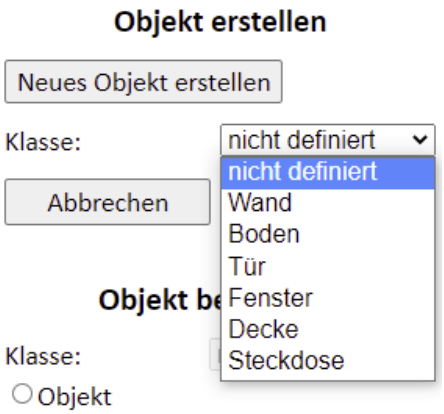

Abbildung 8: Auswahl der Klasse

Der Button „Speichern“, dargestellt durch ein Disketten-Symbol, erlaubt es, das erstellte Objekt zu speichern. Diese Aktion lässt sich ebenfalls durch den „Rückgängig-Button“ rückgängig machen. Wurde ein Objekt editiert, soll aber nicht gespeichert werden, kann der Vorgang mit der Funktion „Abbrechen“ abgebrochen werden.

Abbildung 9: Viewer1 - Objekt bearbeiten

Der Bereich „Objekte bearbeiten“ enthält verschiedene Funktionen der Bearbeitung:

Zunächst kann die semantische Eigenschaft der Klasse eines Objektes geändert werden, wie im Bereich „Objekt erstellen“. Diese Funktion ist nur möglich, wenn ein Objekt ausgewählt ist. Darunter befindet sich eine Auswahl zwischen den Möglichkeiten „Objekt“ und „Punkt“. Diese erlauben die Bearbeitung eines ganzen Objektes oder einem der Punkte.

Neben der semantischen Bearbeitung kann auch die Geometrie eines Objektes bearbeitet werden. Dabei kann eine Rotation (Drehung) und eine Translation (Bewegung) durchgeführt werden. Die Translation verschiebt ein Objekt um einen gesetzten Wert auf einer ausgewählten Achse. Neben „Bewegen“ kann eine Schrittweite eingegeben werden, um welche das Objekt bewegt werden soll. Darunter befinden sich drei Buttons

mit Pfeilen in verschiedenen Farben. Diese repräsentieren die in Abbildung 4 dargestellten Koordinatenachsen x (Rot), y (Grün) und z (Blau). Ein Objekt kann bisher nur entlang der Achsen bewegt werden. Nachdem eine Schrittweite ausgewählt wurde, kann durch Klicken auf den Button mit der ausgewählten Achsen-Farbe das Objekt entsprechend verschoben werden. Einmal Klicken bedeutet eine Bewegung um die Schrittweite.

Ein Objekt kann mit „Drehen“ um die Schrittweite gedreht werden. Die Schrittweite wird in diesem Fall als Winkel in Grad angegeben. Die darunter befindlichen Kreispfeile stellen eine Drehung um eine der drei Koordinatenachsen dar.

Alle Funktionen unter „Objekte bearbeiten“ werden zunächst nur im Frontend gespeichert. Sollen diese dauerhaft gespeichert werden, geht dies mit einem Klick auf den „Speichern“-Button mit dem Diskettensymbol. Alle Änderungen werden daraufhin auch im Backend gespeichert. Sollen diese danach wieder rückgängig gemacht werden, geht dies über den „Rückgängig-Button“. Sollen die Änderungen aus dem Frontend nicht gespeichert, sondern verworfen werden, geht dies äquivalent zu dem Bereich „Objekt erstellen“ über „Abbrechen“. Wird das Programm beendet, ohne zu speichern, werden diese Änderungen nicht beibehalten.

4.3 Viewer2

Viewer2 dient zur Auswahl der Bearbeitungsschritte durch den Roboter. In diesem Viewer können Bearbeitungsschritte wie Streichfarbe oder Schleifgrad einer Wand festgelegt werden. Hier sind keine geometrischen Änderungen möglich, diese sollen in Viewer1 durchgeführt werden. Abbildung 10 zeigt den Aufbau von Viewer2.

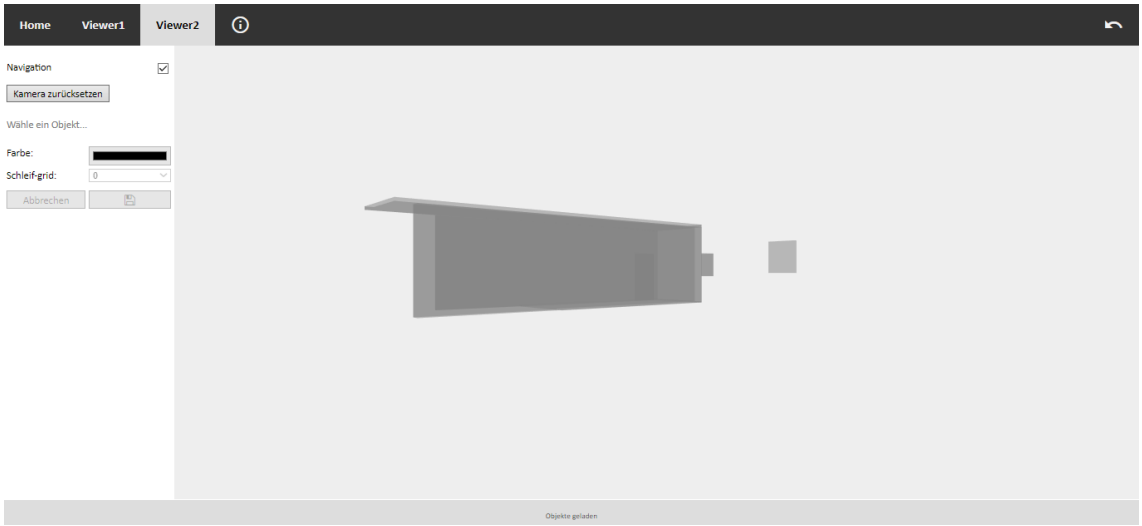

Abbildung 10: Viewer2 Überblick

Der allgemeine Aufbau ist ähnlich wie bei Viewer1. Im linken Bereich befinden sich die Bearbeitungsmöglichkeiten. Im großen Feld in der Mitte befindet sich der Viewer, in welchem die Objekte dargestellt werden. Oben rechts befindet sich der „Rückgängig-Button“ mit welchem bereits gespeicherte Aktionen rückgängig gemacht werden kön-

nen. Am unteren Bildrand befindet sich eine Statusleiste. Diese zeigt die letzte ausgeführte Aktion an und Exceptions, falls welche auftreten. Die Bearbeitungsmöglichkeiten sind in Abbildung 11 vergrößert dargestellt.

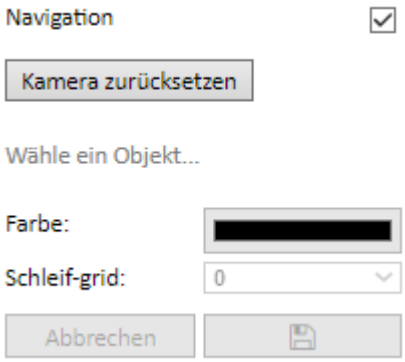

Abbildung 11: Viewer2 – Bearbeitungsmöglichkeiten

Wie in Viewer1 kann auch hier die Navigation an- und ausgeschaltet werden. Ebenfalls gibt es hier die Option „Kamera zurücksetzen“, um die Szene richtig gedreht zu sehen. Das Feld „Wähle ein Objekt…“ zeigt wie bei Viewer1 die Informationen eines Objektes an, wenn es angeklickt wurde. Neben ID und Klasse werden hier auch die festgelegten Bearbeitungsschritte angezeigt.

Die eigentlichen Bearbeitungen in Viewer2 sind darunter zu sehen. Hinter „Farbe“ befindet sich ein Feld aus welchem der Nutzer frei eine Farbe auswählen kann. Abbildung 12 zeigt die Farbauswahl mit dem entsprechend dargestellten Objekt.

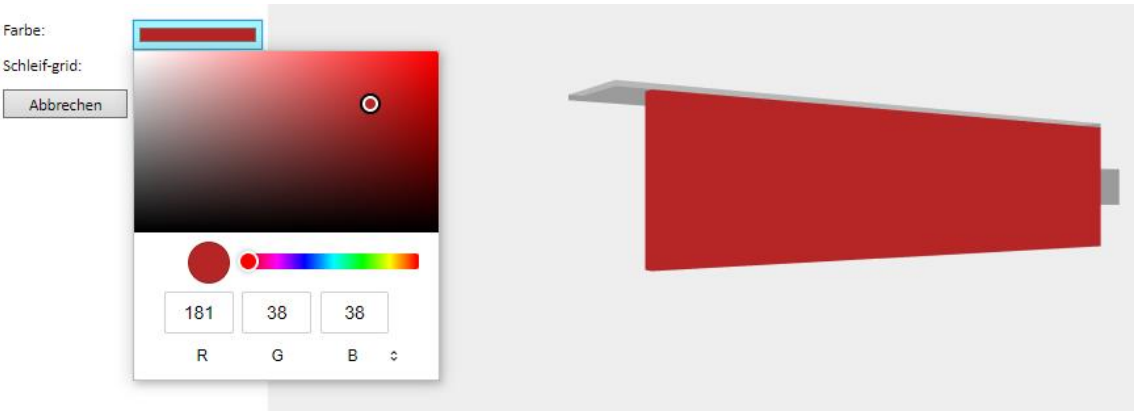

Abbildung 12: Viewer2 – Farbauswahl

Neben der Farbauswahl kann auch eine Schleifoption für ein Objekt gewählt werden. Neben „Schleif-Grid“ kann aus einer Liste ein Schleifgrad gewählt werden. Diese Information wird aktuell nicht im Viewer dargestellt, sondern nur im Informationsfeld angezeigt, da für diesen Bearbeitungsschritt noch keine genauen Vorgaben existieren.

Alle Aktionen im Viewer2 sind zunächst nur im Frontend temporär gespeichert. Um die Änderungen an Farbe und Schleif-Grid dauerhaft zu sichern, kann der „Speichern“-But-

ton mit dem Diskettensymbol angeklickt werden. Alle Änderungen im Frontend in diesem Viewer werden dann ans Backend übersandt und dauerhaft gespeichert. Wenn dies geschehen ist, können Änderungen über den „Rückgängig-Button“ aus Abbildung 10 rückgängig gemacht werden. Wenn die Änderungen im Frontend verworfen werden sollen kann dies mit dem Button „Abbrechen“ geschehen. Alle, bis dahin nicht dauerhaft gespeicherten Änderungen werden dann verworfen.

4.4 Informationen

Der vierte Reiter bietet Informationen rund um das Projekt. Dort befinden sich die Kontaktadressen zu den beteiligten Personen. Zudem sind dort Links zur Hochschule und verwendeter Software aufgeführt. Um die Bedienung des Programmes zu erleichtern, ist dort auch eine Bedienungsanleitung verlinkt.

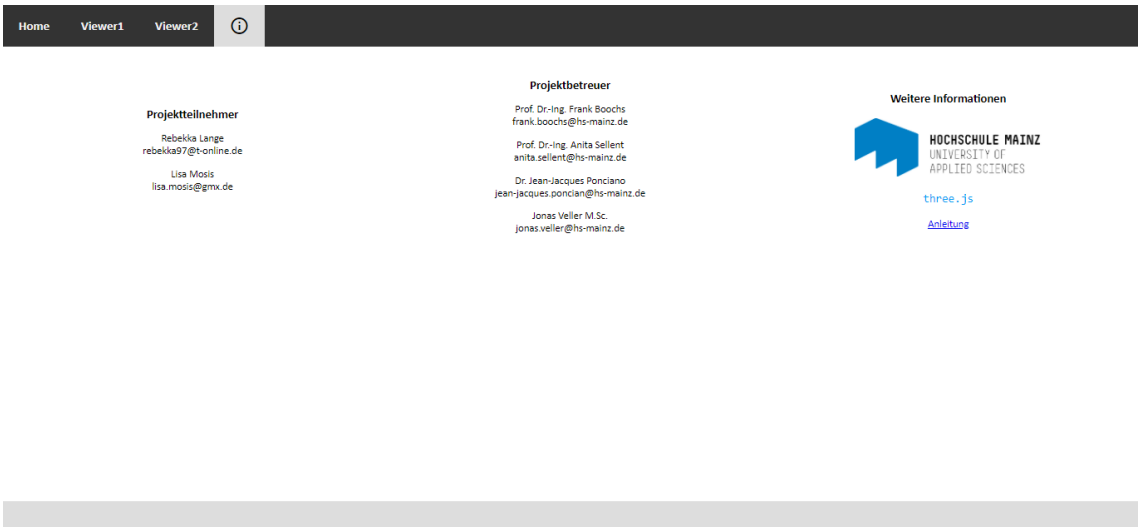

Abbildung 13: Impressum/Hilfe

5 Implementation

Dieses Kapitel beschreibt, wie die Anforderungen implementiert wurden. Zunächst wird der grobe Aufbau der Software erläutert, anschließend die Schnittstellen und die Implementierung der Programm-Komponenten.

5.1 Programmaufbau

Das Programm ist unterteilt in Frontend und Backend. Im Frontend werden mittels HTML und JavaScript die Oberfläche und deren Aktionen definiert und implementiert. Im Backend werden die Daten verwaltet und alle Aktionen protokolliert. Alle Bereiche kommunizieren über Schnittstellen. Abbildung 14 zeigt die Programmteile und deren Interaktion.

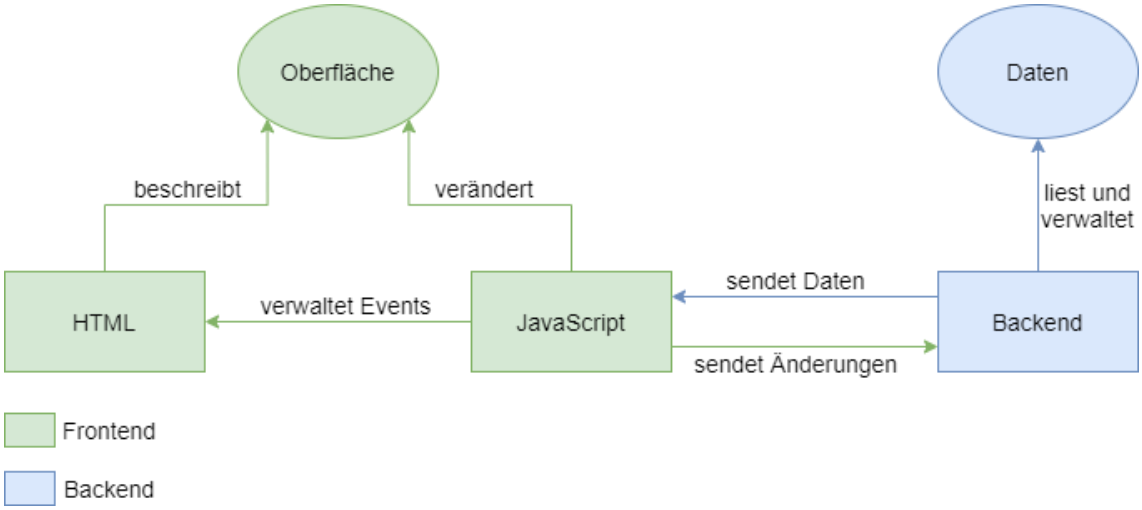

Abbildung 14: Programmaufbau

Im HTML sind alle Bestandteile der Oberfläche beschrieben und sie enthalten Links zu den anderen HTML-Skripten. Für jeden Viewer gibt es eine separate HTML-Datei für seinen spezifischen Aufbau. In JavaScript werden alle Nutzer-Änderungen von HTML-Elementen verwaltet und die entsprechenden Funktionen implementiert. Ebenso ist dort die Steuerung für den Viewer implementiert. Interaktionen mit HTML-Komponenten werden im JavaScript entsprechend ausgeführt und verändern direkt die Oberfläche im Viewer. Die Kommunikation zwischen Frontend und Backend findet zwischen JavaScript und der serverseitigen Java-Klasse GuiApplication.java im Backend statt. Das Backend liest Daten vom Server und sendet diese ans Frontend. Ebenso kann es Änderungen empfangen und speichern. Zudem wird im Backend Protokoll über alle Speicheränderungen geführt.

Diese Softwarearchitektur sorgt dafür, dass alle Dateien zwar auf dem Server liegen, aber sowohl das Rendering des Viewers als auch die Berechnungen der Objektbearbeitungen auf dem Client ausgeführt werden. Ebenso werden die Objekte und die Punktwolke in den Arbeitsspeicher des Clients geladen. Dies sorgt dafür, dass der Arbeitsspeicher des Clients zum Bottleneck der Performance werden kann. Alternativ könnte die Architektur auch so konzipiert werden, dass die Rechenlast hauptsächlich vom Server

getragen wird. Dabei würde nur die so berechnete Grafik an den Client gesendet. Der Nachteil hierbei ist, dass dabei sehr viele Anfragen zwischen Client und Server notwendig sind, wodurch bei Problemen mit der Verbindung schnell die Funktion der gesamten Software zusammenbricht. Ebenso kann die Latenz beim Senden der Benutzerinteraktion den Bearbeitungsfluss stören. Um dies zu verhindern wird hier die Softwarearchitektur mit dem clientseitigen Rechnen gewählt.

5.2 Schnittstellen

Wie jedes Softwarepaket hat auch dieses, wie bereits in 5.1 Programmaufbau erwähnt, mehrere Schnittstellen. Diese sind aufgeteilt in externe und interne Schnittstellen. Externe liegen zwischen dem Benutzer und der Software, beispielweise beim Bedienen der GUI oder beim Einspeisen der Daten (Näheres siehe Kapitel 4 Oberfläche und 5.4 Backend). Eine interne Schnittstelle befindet sich an der REST-API, zwischen dem Backend auf dem Server und dem Frontend, welches die Webseiten sowie Objektdarstellung und -bearbeitung im Browser beinhaltet. Zusätzlich existiert noch eine Schnittstelle zwischen dem Aufbau der Browserseite mit HTML, CSS und eingebettetem JavaScript und dem Three.js Viewer, der komplett in JavaScript implementiert ist.

5.2.1 HTML – JavaScript

Jedes HTML ist in verschiedene Bereiche unterteilt. Im HTML ist das jeweilige JavaScript als Script-Source angegeben. Dadurch kann im JavaScript jedes Element aus dem HTML mittels dessen ID angesprochen werden. Über diese Schnittstelle wird eine Szene des Viewers (vgl. 5.3 Frontend) an einen HTML-Bereich gebunden. Per ID kann auch ein EventListener mit einem HTML-Element verbunden werden, welcher, abhängig vom Event und dem verbundenen HTML-Element, eine Funktion im JavaScript ausführt. EventListener können auf verschiedene Events reagieren. Die hier genutzten Events sind dclick (Doppelklick), click (Klick), input (Auswahl aus Liste oder Eingabe) und change (Farbauswahl in Color Picker). Es gibt zudem passive Elemente, welche keine Events starten, jedoch vom JavaScript aus verändert werden können. Diese sind in diesem Fall Label- oder Paragraph-Elemente, welche Textfelder zur Information enthalten.

5.2.2 Frontend – Backend

Da das Backend auf dem Spring Framework aufbaut, ist die Schnittstelle vom Server zum Internet über eine REST-API im Controller geregelt. Hier wird angegeben welche REST-Anfragen an den Server gestellt werden können. Diese folgen den Prinzipien des http-Protokolls und sind entsprechend ihrer Funktion aufgeteilt in DELETE-, PUT-, POST- und GET-Anfragen. Besonderes Augenmerk liegt hierbei darauf, dass die Funktionen, die durch die Anfragen ausgelöst werden, dem Prinzip der Sicherheit und Idempotenz folgen. So wird sichergestellt, dass die Schnittstelle den Standards entspricht und leicht verstanden, genutzt oder weiterentwickelt werden kann, ohne die genaue Implementation zu kennen (Geheimnisprinzip).

Im Folgenden werden die möglichen Anfragen der Schnittstelle und ihre Nutzung erklärt (genauer in 5.4.2 Funktionen):

/delete

Funktion: Löscht ein Objekt auf dem Server.

Methode: DELETE

Request-Body: String ObjectID

Rückgabe: Boolean mit dem Erfolg der Anfrage

Kann durch die /undo Funktion rückgängig gemacht werden.

/changeObject

Funktion: Übernimmt die übergebenen Änderungen in die Datei des bereits bestehenden Objektes.

Methode: PUT

Request-Body: Objekt (entsprechend der Klasse ObjectDTO)

Rückgabe: Boolean mit dem Erfolg der Anfrage

Kann durch die /undo Funktion rückgängig gemacht werden.

/newObject

Funktion: Erstellt ein neues Objekt auf dem Server. Sollte eine noch nicht vergebene ID übergeben werden, wird diese genutzt, ansonsten wird eine neue generiert.

Methode: POST

Request-Body: Objekt (entsprechend der Klasse ObjectDTO)

Rückgabe: String der übernommenen / generierten ID des Objektes.

Kann durch die /undo Funktion rückgängig gemacht werden.

/changeFilepath

Funktion: Ändert den Projektpfad, aus dem die Objekte geladen werden.

Methode: PUT

Request-Body: String Dateipfad nach dem System data/<Dateipfad>/

(z.B.: data/2021/Projekt1/ → Übergabe: „2021/Projekt1“)

Rückgabe: Boolean mit dem Erfolg der Anfrage

Kann durch die /undo Funktion rückgängig gemacht werden.

/objectInventory

Funktion: Gibt den Inhalt des aktuellen Ordners zurück. Die Einträge werden mit „File“ oder „Directory“ betitelt.

Methode: GET

Request-Body: Leer

Rückgabe: String Array mit den Elementen

/allObjects

Funktion: Übergibt alle Objekte des aktuellen Projektes, bis zu einer Anzahl von 1000. Die maximale Anzahl (maxSending) kann bei Bedarf am Anfang des Controllers angepasst werden.

Methode: GET

Request-Body: Leer

Rückgabe: Array der Objekte (ObjectDTO)

/objectById/{id}

Funktion: Übergibt ein bestimmtes Objekt, dessen ID übergeben wurde.

Methode: GET

Path-Variable: String ID des gesuchten Objektes

Request-Body: Leer

Rückgabe: Objekt (ObjectDTO) mit dem Dateinamen <ID>.ply (z.B.: „2.ply“)

/getPointcloud

Funktion: Übergibt die Punktwolke des Projektes, die als „pointcloud.ply“ im Projektordner vorliegen muss.

Methode: GET

Request-Body: Leer

Rückgabe: Objekt der Punktwolke (PointcloudDTO)

/undo

Funktion: Macht die letzte Änderung auf dem Server rückgängig.

Methode: GET

Request-Body: Leer

Rückgabe: Die zum Umkehren ausgeführte Methode mit ihren Übergabeparametern. Bei einer internen Exception oder dem Fehlen von rückgängig machbaren Schritten wird null übergeben. (z.B.: zum Umkehren von „delete(<object>)“ → „undo_delete(<object>)“)

5.2.3 JSON

Um Objekte über die Schnittstelle zu verschicken, können keine Java-Objekte verwendet werden. Deshalb werden diese zu JSON-Objekten geparst. Für ein ObjectDTO sieht ein solches JSON beispielweise wie folgt aus (JSON):

```
{"objID":"",
"objClass":"Wall",
"color":" #f50000",
"sanding":"100",
"vertex_in-
dices":[[0,2,3,1],[2,6,7,3],[6,4,5,7],[4,0,1,5],[1,3,7,5],[0,4,6,2]],
"x":[-8.219, -8.219, -8.183, -8.183, -8.219, -8.219, -8.183, -8.183],
"y":[0.498, 3.35, 0.498, 3.35, 0.498, 3.35, 0.498, 3.35],
"z":[-0.005, -0.005, -0.005, -0.005, 4.515, 4.515, 4.515, 4.515]
}
```

5.3 Frontend

Es gibt vier Webseiten: index.html, Viewer1.html, Viewer2.html und Impressum.html. Alle sind miteinander verlinkt und können von jeder Seite alle anderen aufrufen. Jedes HTML-Element besitzt eine ID, damit es von einem JavaScript zugreifbar ist. Das Erscheinungsbild der HTML-Elemente ist in der CSS-Datei viewer_stylesheet.css festgelegt. Dort sind Position, Größe, Schriftart, Farbe und Ähnliches für verschiedene Elemente (z.B. Buttons) festgelegt. Die implementierten Funktionen befinden sich in ViewerControls.js und Viewer2Controls.js, aufgeteilt auf die Funktionalitäten der beiden Viewer.

Die Webseite index.html dient zur Dateiauswahl über eine REST-API per GET- und PUT-Aufrufe, welche in der Backend-Klasse GuiApplication.java implementiert sind. Mit GET werden verfügbare Dateipfade vom Backend erfragt und mit PUT wird der Pfad zum ausgewählten Projekt ans Backend gesendet.

In Viewer1.html sind HTML-Elemente für die Ansicht und Kontrolle von einer Punktwolke und den daraus generierten Objekten, sowie Bearbeitungsmöglichkeiten von Ob-

jekten und deren Speicherung auf dem Server, definiert. Die Funktionen zu den Elementen sind in ViewerControls.js implementiert, welche über die HTML-JavaScript-Schnittstelle mit dem HTML verbunden ist und alle Events der Viewer1.html-Elemente verwaltet.

In Viewer2.html sind Elemente zur Ansicht der korrigierten Objekte, sowie die Möglichkeit zur Festlegung von Streichfarbe und Schleif-Grid für Objekte, definiert. Die Implementierung der Funktionen befindet sich in Viewer2Controls.js, welche über die HTML-JavaScript-Schnittstelle mit dem HTML verbunden ist und alle Events der Viewer2.html-Elemente verwaltet.

In Impressum.html befinden sich Informationen zu Projektteilnehmern und -betreuern, sowie Links zu projektrelevanten Webseiten und ein Download-Link für den User Guide.

5.3.1 Funktionen

Alle wichtigen Funktionen im Frontend sind in folgender Liste erläutert. Falls nicht anders angegeben, ist die jeweilige Funktion für beide Viewer identisch.

Objekte laden

HTML-Element: -

JavaScript-Funktion: loadObjects()

Funktion: Sendet eine GET-Anfrage ans Backend und lädt alle erhaltenen Objekte in die Szene, wobei die Farbe entsprechend der Klasse (Viewer1) oder der Streichfarbe (Viewer2) gesetzt wird, und richtet die Kamera auf alle Objekte. Diese Funktion wird beim Aufruf der Webseite ausgeführt.

Punktwolke laden (Viewer1)

HTML-Element: -

JavaScript-Funktion: loadPointcloud()

Funktion: Sendet eine GET-Anfrage ans Backend und lädt die erhaltenen Punkte als Punktwolke in die Szene, wobei die Farbe auf Grau gesetzt wird. Diese Funktion wird beim Aufruf der Webseite ausgeführt.

Navigation an-/ausschalten

HTML-Element: #navigation

JavaScript-Funktion: changeNavigation(boolean)

Funktion: Aktiviert oder deaktiviert die Navigation in der Szene, wenn im HTML-Element ein click-Event aufgerufen wird.

Punktwolke zeigen (Viewer1)

HTML-Element: #showPointcloud

JavaScript-Funktion: showPointcloud(boolean)

Funktion: Rendert die Punktwolke nur, wenn aktiviert.

Kamera zurücksetzen

HTML-Element: #setCam

JavaScript-Funktion: resetCamera()

Funktion: Wenn das HTML-Element ein click-Event aufruft, wird die Kamera so gedreht, dass die z-Achse vertikal nach oben zeigt.

Aktion rückgängig machen

HTML-Element: #undoButton

JavaScript-Funktion: undo()

Funktion: Sobald ein click-Event vom HTML-Element ausgeht, wird eine GET-Anfrage ans Backend gesendet, um die zuletzt protokollierte Aktion zu erhalten und macht diese im Frontend rückgängig. War die letzte Aktion ein neues Objekt, wird dieses wieder gelöscht, wurde ein Objekt gelöscht, wird dieses wieder hinzugefügt, wurde ein Objektattribut verändert, wird das Objekt aus der Szene entfernt und aus der JSON-Antwort vom Backend neu geladen.

Objektauswahl

HTML-Element: #changeObjectMode_Object

JavaScript-Funktion: setSelectMode(Objekt), click(event), objectSelection()

Funktion: Setzt den Bearbeitungsmodus auf „Objekt“, wenn vom HTML-Element ein click-Event ausgeht, welches dazu führt, dass ein dclick-Event auf ein Objekt dieses auswählt, bearbeitbar macht und alle Eckpunkte als rote Kugeln lädt.

Punktauswahl (Viewer1)

HTML-Element: #changeObjectMode_Punkt

JavaScript-Funktion: setSelectMode(Punkt), click(event), pointSelection()

Funktion: Setzt den Bearbeitungsmodus auf „Punkt“ “, wenn vom HTML-Element ein click-Event ausgeht, welches dazu führt, dass ein dclick-Event auf einen Punkt diesen auswählt und bearbeitbar macht. Zuvor muss ein Objekt ausgewählt worden sein, damit dessen Punkte angezeigt werden.

Objektinformationen anzeigen

HTML-Element: #objInfo

JavaScript-Funktion: -

Funktion: Zeigt die Objektinformationen zum aktuell ausgewählten Objekt an, welche die Objekt-ID und dessen Klasse sind und im Viewer2 zusätzlich die Streichfarbe und Schleifqualität. Dieses HTML-Element wird nach dem Aus- oder Abwählen eines Objektes entsprechend vom JavaScript aus aktualisiert.

Objekt löschen (Viewer1)

HTML-Element: #deleteChosenObject

JavaScript-Funktion: deleteObject()

Funktion: Entfernt das ausgewählte Objekt aus der Szene, wenn ein click-Event vom HTML-Element ausgelöst wird, und sendet eine DELETE-Anfrage mit der Objekt-ID ans Backend. Bei Misserfolg beim Löschen im Backend wird das Objekt wieder der Szene hinzugefügt.

Neues Objekt erstellen (Viewer1)

HTML-Element: #makeNewObject

JavaScript-Funktion: setSelectMode(Neu)

Funktion: Setzt den Bearbeitungsmodus auf „Neu“, wenn vom HTML-Element ein click-Event ausgeht, sodass Doppelklicken (dclick-Event) in der Szene Punkte für ein neues Objekt setzt. Alle Objektbearbeitungs-Funktionen sind während diesem Bearbeitungsmodus deaktiviert.

Klasse für neues Objekt setzen (Viewer1)

HTML-Element: #newObjClass

JavaScript-Funktion: setNewObjClass(class)

Funktion: Wenn im HTML-Element ein click-Event auftritt, wird die Klasse für ein neues Objekt gespeichert. Sobald das Objekt erstellt wird, wird die Klasse zugeordnet und das Objekt entsprechend eingefärbt.

Neues Objekt speichern (Viewer1)

HTML-Element: #saveNewObject

JavaScript-Funktion: saveNewObject()

Funktion: Ein click-Event auf das HTML-Element sorgt dafür, dass ein neues Mesh-Objekt aus gesetzten Punkten erstellt wird, falls mindestens drei gesetzt wurden. Das Objekt wird per POST-Aufruf ans Backend gesendet. Bei erfolgreicher Speicherung im Backend wird die neu generierte Objekt-ID in den Objektinformationen gespeichert. Bei Misserfolg wird das Objekt im Frontend gelöscht.

Neues Objekt verwerfen (Viewer1)

HTML-Element: #cancelNewObject

JavaScript-Funktion: cancelNewObject()

Funktion: Die Erstellung eines neuen Objektes wird abgebrochen und alle gesetzten Punkte gelöscht, wenn ein click-Event vom HTML-Element ausgeht.

Änderungen in Frontend zwischenspeichern

HTML-Element: -

JavaScript-Funktion: log(entry)

Funktion: Protokolliert alle Objektbearbeitungen, die zunächst nur im Frontend vorgenommen werden in einem Array. Ein entry hat die Form [Funktion, Objekt-ID, Änderung / alter Wert], also z.B. [„translation“, „1“, „x“, 1], bei einer Klassenänderung wird die vorherige Klasse gespeichert.

Klasse ändern (Viewer1)

HTML-Element: #changeObjClass

JavaScript-Funktion: changeClass(class)

Funktion: Speichert die neue Klasse für ein Objekt, wenn ein input-Event auf dem HTML-Element ausgelöst wird, und färbt das Objekt entsprechend ein.

Translationsweite setzen (Viewer1)

HTML-Element: #translationSteps

JavaScript-Funktion: -

Funktion: In diesem Feld wird die Schrittweite in Projekteinheiten angegeben, in welcher ein Objekt bewegt werden soll. Die Schrittweite wird bei einem entsprechenden input-Event nur im HTML geändert und bei Translation als Variable übergeben.

Translation (Viewer1)

HTML-Element: #translX, #translY, #translZ

JavaScript-Funktion: translation(object, axis, stepsize)

Funktion: Das ausgewählte Objekt wird nach einem click-Event auf einem der HTML-Elemente in angegebener Schrittweite entlang der ausgewählten Achse verschoben, indem die Schrittweite auf die jeweilige Koordinate jedes Punktes addiert wird.

Rotationswinkel setzen (Viewer1)

HTML-Element: #rotationSteps

JavaScript-Funktion: -

Funktion: In diesem Feld wird der Rotationswinkel in Grad angegeben. Der Rotationswinkel wird bei einem entsprechenden input-Event nur im HTML geändert und bei Rotation als Variable übergeben.

Rotation (Viewer1)

HTML-Element: #rotationX, #rotationY, #rotationZ

JavaScript-Funktion: rotation(object, axis, angle)

Funktion: Führt bei einem click-Event von einem der HTML-Elemente eine Achsenrotation des ausgewählten Objektes um seinen Mittelpunkt durch, indem eine Translation des Objekt-Mittelpunktes in den Ursprung, eine Rotation um die gewählte Achse und eine Rücktranslation vorgenommen werden.

Farbe ändern (Viewer2)

HTML-Element: #myColor

JavaScript-Funktion: setColor(color)

Funktion: Speichert die neue Farbe für ein Objekt, wenn ein change-Event vom HTML-Element ausgeht, und färbt das Objekt entsprechend ein.

Schleif-Grid ändern (Viewer2)

HTML-Element: sanding

JavaScript-Funktion: setSanding(sanding)

Funktion: Speichert den neuen Schleif-Grid-Wert für ein Objekt, wenn ein input-Event vom HTML-Element ausgeht.

Änderungen speichern

HTML-Element: #saveChangedObject

JavaScript-Funktion: makePersistent()

Funktion: Wenn auf dem HTML-Element ein click-Event auftritt, wird das log-array eintragsweise abgearbeitet, alle Änderungen mit einer PUT-Anfrage und dem Objekt als JSON ans Backend gesendet und das log-array anschließend geleert. Sollte eine Änderung im Backend nicht gespeichert werden können, wird sie auch im Frontend wieder rückgängig gemacht.

Änderungen verwerfen

HTML-Element: #cancelChangedObject

JavaScript-Funktion: cancelChanges()

Funktion: Wenn auf dem HTML-Element ein click-Event auftritt, wird das log-array eintragsweise abgearbeitet, alle Änderungen umgekehrt und das log-array anschließend geleert.

Status anzeigen

HTML-Element: #status

JavaScript-Funktion: -

Funktion: Gibt den Status der zuletzt ausgeführten Funktion an, indem der Wert des HTML-Elementes vom JavaScript entsprechend geändert wird. Dieser gibt die zuletzt ausgeführte Funktion an und eine entsprechende Fehlermeldung, falls eine Exception aufgrund einer Internetverbindung bei REST-Anfragen aufgetreten ist.

JSON erstellen

HTML-Element: -

JavaScript-Funktion: createJSON(object)

Funktion: Erstellt ein JSON im vorgegebenen Format (vgl. 5.2.3 JSON) aus einem Objekt.

5.3.2 Three.js-Klassen

Zum Darstellen von Objekten, Punktwolke und der Möglichkeit zur Objektbearbeitung wurden Klassen der Bibliothek Three.js (Three.js 2021) verwendet. Folgende Übersicht erläutert die verwendeten Klassen von Three.js:

Scene.js

Erstellt eine Szene und ermöglicht das Rendering hinzugefügter Objekte mittels einer Kamera und einem Renderer.

PerspectiveCamera.js

Mit dieser Klasse erhält der Nutzer eine perspektivische Sicht auf die Szene, ähnlich der natürlichen Wahrnehmung des Menschen.

WebGLRenderer.js

Mit dem WebGLRenderer wird die Szene mittels WebGL regelmäßig gerendert (WebGL 2021).

TrackballControls.js

Diese Klasse ermöglicht die Navigation durch die Szene mit einer Maus oder per Wischen und Tippen auf einem Touchscreen.

Raycaster.js

Diese Klasse dient dazu, eine 3D-Koordinate an dem Punkt zu erhalten, auf den eine Maus auf dem 2D-Bildschirm geklickt hat und kann Überschneidungen zwischen der Mausposition und Objekten in der Szene bestimmen. Ein Raycaster gibt ein Array aus Objekten zurück, auf welche die Maus geklickt hat und es kann der genaue Schnittpunkt zwischen der Maus und dem Objekt abgefragt werden. Diese Funktion wird zur Objekt- und Punktauswahl verwendet, sowie zum Setzen von Punkten für ein neues Objekt. Im letzteren Fall werden Überschneidungen zwischen der Maus und der Punktwolke bestimmt und ein Threshold von zehn Zentimetern gesetzt, sodass ein Punkt als überschnitten gilt, wenn er sich in diesem Threshold zur Maus befindet.

Color.js

Die Klasse Color.js dient dazu, einem Objekt eine Farbe zuzuweisen, welche in diesem Projekt klassenabhängig (Viewer1) oder streichfarben-abhängig (Viewer2) ist. Die Farbe wird als Hexadezimal-Zahl angegeben.

Box3.js

Um die Kamera zu Beginn auf alle Objekte zu richten, wird eine Bounding Box um alle Objekte gelegt und die Kamera in relativer Entfernung auf das Zentrum der Box gerichtet.

Vector3.js

Ein dreidimensionaler Vektor dient der Speicherung von Koordinaten, zum Beispiel eines Objektmittelpunktes oder einer angeklickten Koordinate.

Vector2.js

Diese Klasse dient der Darstellung der Mauskoordinaten auf dem Bildschirm, welche bei der Überschneidungsberechnung vom Raycaster benötigt werden.

AxesHelper.js

Zur Orientierung wird der Szene in Viewer1 ein Koordinatensystem hinzugefügt, welches die Achsen (x: Rot, y: Grün, z: Blau) in jeweils zehn Metern Länge darstellt.

Float32BufferAttribute.js

Um ein Positionsarray zu einer Geometrie hinzuzufügen, wird ein Float32BufferAttribute benötigt, welches mit dem Positionsarray und der Anzahl enthaltener Dimensionen erstellt und anschließend der Geometrie hinzugefügt werden kann.

BufferGeometry.js

Um ein Objekt aus mehreren Punkten in Three.js darzustellen wird eine BufferGeometry erstellt. Dieser wird als Float32BufferAttribute ein Positionsarray hinzugefügt, welches der Form [x1, y1, z1, x2, y2, z2, …] entsprechen muss. Alle Objekte in Three.js werden als eine Zusammensetzung von Dreiecken gerendert. Dazu werden immer drei aufeinander folgende Koordinaten in dem Positionsarray zu einem Dreieck zusammengesetzt und kein Punkt doppelt vergeben. Sollen die Dreiecke in einer anderen Reihenfolge gerendert werden und Punkte doppelt verwendet werden, kann eine Index-Liste der Geometrie hinzugefügt werden. Da nicht alle Flächen automatisch aus Dreiecken, sondern wie in den Beispieldateien auch aus Vierecken, bestehen, wird beim Laden jedes Objektes auf die Anzahl geachtet. Flächen aus Vierecken werden in zwei Dreiecke aufgeteilt, sodass z.B. [1 2 3 4] zu [1 2 3], [3 4 1] wird. Mehreckige Flächen werden zurzeit nicht berücksichtigt.

Earcut.js

Wird ein neues Objekt erstellt, müssen die Indizes für die Dreiecke zum Rendern erst berechnet werden. Dazu verwendet Three.js die Klasse Earcut.js, welche ein übergebenes Polygon trianguliert und eine Index-Liste zurückgibt. Da dieser Algorithmus nur für den zweidimensionalen Raum ausgelegt ist, wird die Fläche näherungsweise in einen zweidimensionalen Raum transformiert. Dadurch wird diejenige Koordinate ignoriert, auf welcher die geringste Strecke zwischen den Punkten vorhanden ist.

SphereGeometry.js

Dies ist eine Erweiterung der BufferGeometry zum Rendern einer dreidimensionalen Kugel, welche die Position, den Radius und die Anzahl der Höhen- und Breitensegmente enthält.

LineBasicMaterial.js

Diese Klasse definiert die Darstellung von Linien und erhält in diesem Fall nur die Farbe Rot.

MeshBasicMaterial.js

Diese Klasse definiert die Darstellung von Objekten und wird mit einer Farbe (Color.js) und einer Deckkraft angelegt. Die Deckkraft wird auf 50 Prozent gesetzt, damit eventuell übereinanderliegende Objekte gesehen werden.

PointsMaterial.js

Diese Klasse definiert die Darstellung von Punkten und erhält in diesem Fall nur die Farbe Rot für Eckpunkte und neue Punkte und grau für die Punktwolke.

Mesh.js

Ein Mesh wird aus einer Geometrie und einem Material erzeugt und bildet ein fertiges Objekt, welches zur Szene hinzugefügt werden kann.

Points.js

Die Punktwolke in Viewer1 wird als ein Objekt gerendert, welches eine Liste von nicht verbundenen Punkten ist.

Line.js

Eine oder mehrere fertige Linien aus Positionsattributen und einem Linienmaterial, welche als Objektkanten bei der Objekterstellung gerendert werden.

5.4 Backend

Die Daten der Anwendung werden nicht in einer Datenbank, sondern als Dateien gespeichert. Diese liegen im Format „.ply“ vor und werden im „data“ Ordner im Root der Software abgelegt. Die Ordnerstruktur sollte wie folgt aussehen:

```
/GUI
   /data
      …
         /Projekt1
              /objects
                   1.ply
                   2.ply
              pointcloud.ply
         /Projekt2
              /objects
                   1.ply
                   2.ply
              pointcloud.ply
      …
```

Abbildung 15: Beispiel Ordnerstruktur

Solange die Projekte innerhalb des Projektordners aufgebaut sind wie in Abbildung 15 und im „data“-Ordner abgelegt sind, kann die Ordnerstruktur auch beliebig tiefer sein, sodass auch Ordner für Jahre oder ähnliches angelegt werden können.

Die ply-files müssen der Struktur aus Abbildung 16 oder Abbildung 17 entsprechen, um für die Software maschinenlesbar zu sein.

```
001 ply
002 format ascii 1.0
003 comment made by JavaPointCloud library
004 comment id=2 class=Wall
005 element vertex 8
006 property double x
007 property double y
008 property double z
009 element face 6
010 property list uchar int vertex_index
011 end_header
012 -8.219 0.498 -0.005
013 -8.219 3.35 -0.005
014 -8.183 0.498 -0.005
015 -8.183 3.35 -0.005
016 -8.219 0.498 4.515
017 -8.219 3.35 4.515
018 -8.183 0.498 4.515
019 -8.183 3.35 4.515
020 4 0 2 3 1
021 4 2 6 7 3
022 4 6 4 5 7
023 4 4 0 1 5
024 4 1 3 7 5
025 4 0 4 6 2
026
```

Abbildung 16: Beispiel 2.ply ohne Bearbeitungsschritte

```
001 ply
002 format ascii 1.0
003 comment made by JavaPointCloud library
004 comment id=2 class=Wall color=#ff0000 sanding=
005 element vertex 8
006 property double x
007 property double y
008 property double z
009 element face 6
010 property list uchar int vertex_index
011 end_header
012 -8.219 0.498 -0.005
013 -8.219 3.35 -0.005
014 -8.183 0.498 -0.005
015 -8.183 3.35 -0.005
016 -8.219 0.498 4.515
017 -8.219 3.35 4.515
018 -8.183 0.498 4.515
019 -8.183 3.35 4.515
020 4 0 2 3 1
021 4 2 6 7 3
022 4 6 4 5 7
023 4 4 0 1 5
024 4 1 3 7 5
025 4 0 4 6 2
026
```

Abbildung 17 Beispiel 2.ply mit Bearbeitungsschritten

Die Files enthalten dabei Kommentare (vgl. Abbildung 16 und Abbildung 17 Zeile 4), die zusätzliche Informationen enthalten, die für dieses Projekt wichtig sind. Hier werden ID, Objektklasse und optional auch die Farbe und der Schleif-Vorgang, mit dem das Objekt später bearbeitet werden soll, gespeichert.

Außerdem enthält ein ply-file immer Informationen zum eigenen Inhalt, wie die Anzahl der Elemente (z.B.: „element vertex 8“ bedeutet 8-mal das Element „vertex“ = Punkt) und deren Zusammensetzung (ein Element „vertex“ besteht aus „double x“, „double y“ und „double z“). Diese Elemente werden dann nach dem „header“ aufgelistet (vgl. Abbildung 16 und Abbildung 17 nach Zeile 11).

Die Flächen, die auf den Punkten aufbauen, werden immer im gleichen Format angegeben. Die erste Zahl (Bsp. **4** 0 2 3 1) gibt an, aus wie vielen Punkten die Fläche besteht. Die übrigen Zahlen (4 **0 2 3 1**) geben an, aus welchen Punkten in welcher Reihenfolge die Fläche besteht.

Zum Schreiben und Lesen der Dateien werden Klassen aus dem sjmply-Projekt von Dirk Toewe verwendet (Toewe 2017), in Verbindung mit der Klasse PlyManager von Dr. Jean-Jacques Ponciano. Diese Klassen erlauben es leicht auf die ply-files zuzugreifen.

Die Punktwolke ist ebenfalls ein ply-file, jedoch vereinfacht aufgebaut, da keine Flächen abgebildet werden müssen. Im header befinden sich Informationen zur Anzahl der Punkte und die Reihenfolge der Koordinaten. Zusätzlich besitzt jeder Punkt eine Intensität. Darunter sind zeilenweise die Punkte aufgelistet. (Vgl. Abbildung 18)

```
001 ply
002 format ascii 1.0
003 element vertex 17
004 property double x
005 property double y
006 property double z
007 property double intensity
008 end_header
009 0.00122 1.02563 -1.32056 0.30353
010 0.00122 1.03735 -1.32153 0.31598
011 0.00122 1.05347 -1.32056 0.29255
012 0.00122 1.14282 -1.31274 0.29914
013 0.00122 1.22241 -1.31860 0.29596
014 0.00122 1.26440 -1.31763 0.26764
015 0.00122 1.28198 -1.31714 0.27448
016 0.00122 1.28735 -1.31665 0.29718
017 0.00122 1.30640 -1.31714 0.27936
018 0.00122 1.32104 -1.31274 0.27961
019 0.00122 1.37427 -1.31763 0.25835
020 0.00122 1.39478 -1.31226 0.25664
021 0.00122 1.42651 -1.31665 0.25298
022 0.00122 1.42798 -1.31128 0.25982
023 0.00122 1.47681 -1.31665 0.26177
024 0.00122 1.50610 -1.31274 0.37777
025 0.00122 1.51978 -0.93188 0.81172
026
```

Abbildung 18: Beispiel pointcloud.ply

5.4.1 Log und undo

Um die undo-Funktion (vgl. Kapitel 5.4.2) möglich zu machen, werden alle aufgerufenen Funktionen, bei denen etwas auf dem Server verändert wird, in der Datei „log.txt“ gespeichert. Diese log Datei liegt im Root des Projektes. Da ein undo nur temporär relevant ist, wird diese Datei bei jedem Serverneustart überschrieben und ist auf 50 Zeilen begrenzt. Dieser Wert kann bei Bedarf oben im Controller angepasst werden. Diese Begrenzung wird erreicht, indem nach dem Schreiben in die Datei von vorne her Zeilen gelöscht werden, wenn sie zu lang geworden ist.

Funktionen, die keine Änderungen auf dem Server verursachen, müssen nicht protokolliert werden. Aufzunehmen ist also nur das Erstellen, Ändern und Löschen eines Objektes, außerdem das Ändern des Dateispeicherpfades.

Da beim Ändern der Daten auf dem Server der vorherige Zustand verloren geht, muss dieser mit in der log Datei gespeichert werden. So werden immer die Variablen mitgespeichert, die zum Umkehren der Änderungen nötig sind.

Jeder Eintrag im log-file setzt sich aus einem Zeitstempel (Format: ddMMyyyy_hhmmss), einem Kürzel für die Art des Eintrags (f = Funktion, e = Exception), der ausgeführten Funktion (function(parameters)) und der Umkehrfunktion dazu (undo_function(parameters)) zusammen. Bei einer Exception werden die letzten beiden Teile dabei durch die Beschreibung dieser ersetzt.

```
001 timestamp type change undo
002 11022021_022311 f changeFilepath(BeispielProjekt) undo_change-
    Filepath("")
003
```

Abbildung 19: Beispiel für einen Eintrag im log-file

5.4.2 Funktionen

Alle Anfragen, die an den Server gestellt werden können, werden in GuiApplication.java implementiert. Die einzelnen Funktionen im Backend werden im Nachfolgenden genauer erklärt:

/delete

Durch den Aufruf „delete“ aus dem Browser kann ein bestimmtes Objekt auf dem Server gelöscht werden. Dabei wird die ID des Objektes übergeben. Hierbei wird im aktuellen Ordner das ply-file mit dem Namen gesucht, der der übergebenen ID entspricht. Wird die Datei gefunden und erfolgreich gelöscht, wird „true“ als Antwort zurückgegeben und der Vorgang wird im log-file (5.4.1 Log und undo) vermerkt. Sollte jedoch eine Exception auftreten oder die Datei nicht gefunden werden, wird „false“ zurückgegeben.
Die Änderung der Daten auf dem Server kann durch den Aufruf der undo-Funktion rückgängig gemacht werden.

/changeObject

Diese Funktion erlaubt es, ein bereits bestehendes Objekt auf dem Server zu ändern. Dabei werden alle Daten des Objektes an das Backend übergeben. Sollte ein Parameter dabei „null“ sein, so wird dieser nicht überschrieben, ebenfalls kann die ID eines Objektes nicht auf diese Weise geändert werden. Alle anderen Daten überschreiben die des bereits bestehenden Objektes. Wenn das Objekt erfolgreich überschrieben wurde, wird „true“ als Antwort zurückgegeben und der Vorgang wird im log-file vermerkt. Bei einer Exception oder einem noch nicht bestehendem Objekt wird „false“ zurückgegeben.

Die Änderung der Daten auf dem Server kann durch den Aufruf der undo-Funktion rückgängig gemacht werden.

/newObject

Durch die Funktion „newObject“ wird ein mit dem Aufruf übergebenes Objekt auf dem Server als ply-file angelegt. Dabei ist es möglich eine ID mit zu übergeben. Sollte also die ID nicht null oder ein leerer String sein, so wird überprüft, ob die vorgeschlagene ID bereits vorhanden ist. Falls die ID noch nicht vergeben ist, wird dann diese übernommen, ansonsten wird eine neue generiert, indem die erste freie ID gesucht wird. Da so der Sender der Anfrage nicht weiß unter welcher ‚ID das neue Objekt zu finden ist, wird diese als Antwort zurückgegeben. Sollte während dem Ausführen der Funktion eine Exception auftreten, wird als Antwort ein leerer String zurückgegeben. Wenn die Funktion fehlerfrei abläuft, wird diese im log-file eingetragen.

Die Änderung der Daten auf dem Server kann durch den Aufruf der undo-Funktion rückgängig gemacht werden.

/changeFilepath

Diese Funktion ändert den Dateipfad, in dem nach der Punktwolke und dem Objekte-Ordner gesucht werden. Hierfür wird der Dateipfad, ab „data/“ ohne einleitenden und abschließenden „/“ übergeben (z.B.: data/2021/Projekt1/ → Übergabe: „2021/Projekt1“). So kann dann der gesamte Dateipfad ab dem Root des Programmes aus „data/“ + übergebener Dateipfad oder Ordner + „/“ generiert werden. Hierbei können auch längere Pfade übergeben werden. Dabei wird getestet, ob der Dateipfad existiert. Sollte der angegebene Pfad existieren und keine Exception aufgetaucht sein, wird „true“ als Antwort zurückgegeben und ein Vermerk im log-file gemacht. Ansonsten wird „false“ zurückgegeben.

Um zum Standardpfad „data/“ zurückzukehren, kann einfach ein leerer String oder ein Leerzeichen übergeben werden. Um ein Projekt anzeigen zu können, sollte der Ordner ausgewählt sein, in dem die „pointcloud.ply“ und der Ordner „objects“ mit den Objekten liegen (vgl. 4.1 Home).

Die Änderung der Daten auf dem Server kann durch den Aufruf der undo-Funktion rückgängig gemacht werden.

/objectInventory

„ObjectInventory“ gibt nach dem Aufrufen den Inhalt des aktuellen Ordners zurück. Dabei werden erst alle Dateien und Ordner im aktuellen Ordner abgefragt und in zwei Gruppen aufgeteilt. Diese werden dann jeweils mit der passenden Bezeichnung „File:“ oder „Directory:“ versehen und als Array von Strings zurückgegeben. Sollte der Ordner leer sein, so wird null zurückgegeben.

/allObjects

Durch den Aufruf „allObjects“ aus dem Browser können alle Objekte aus dem „objects“ Ordner des aktuellen Dateipfades abgefragt werden. Die Objekte werden als Array von Objekten zurückgegeben. Dabei gibt es eine Variable, die die Höchstanzahl der gesendeten Objekte angibt (maxSending). Diese kann oben im Controller angepasst werden. Da noch kein größeres Beispielprojekt vorliegt, konnte diese Anzahl noch nicht ausführlich getestet werden. Sollten keine Objekte im „objects“ Ordner sein, wird null zurückgegeben. Bei einer Exception beim Lesen eines einzelnen Objektes wird dieses einfach übersprungen.

/objectById/{id}

Mit dieser Funktion kann ein einzelnes Objekt mit einer bestimmten ID abgefragt werden. Die ID wird dabei als Path-Variable mit übergeben. Wenn das Objekt (<ID>.ply) im aktuellen Ordner nicht gefunden wird, wird null, sonst einfach das Objekt zurückgegeben.

/getPointcloud

Diese Funktion gibt die Punktwolke an den Browser zurück. Dafür wird aus dem aktuellen Ordner die Datei „pointcloud.ply“ als PointcloudDTO gelesen und an den Browser zurückgegeben. Hier kann aufgrund der unterschiedlichen Informationen weder ObjectDTO noch PlyManager verwendet werden, auch wenn die genutzten Methoden zum Teil sehr ähnlich sind. Sollte keine Punktwolke gefunden werden, wird stattdessen null zurückgegeben.

/undo

Die Funktion „undo“ macht die letzte Änderung auf dem Server rückgängig. Die Funktionen, die Änderungen auf dem Server vornehmen können (außer „undo“), sind „delete“, „changeObject“, „newObject“ und „changeFilepath“. Für diese Funktionen ist jeweils eine Methode implementiert, die die Auswirkungen dieser rückgängig macht. Beispielweise ist für „delete“ die Umkehrfunktion „undo_delete“ implementiert, die ein neues Objekt erstellt. Hierbei werden die Informationen benötigt, die verändert wurden. Um diese nutzen zu können, werden alle Daten, die zum Rückgängigmachen der Funktion nötig sind, während den jeweiligen Funktionen im log-file temporär gespeichert (5.4.1 Log und undo). Das log-file enthält dabei die Umkehrfunktion der jeweiligen

Funktion mit ihren Übergabeparametern. Diese wird durch die undo-Methode dynamisch aufgerufen, wodurch die Änderungen rückgängig gemacht werden. Da im log-file auch Exceptions gespeichert werden und zum Rückgängig machen immer der letzte Eintrag angeschaut wird, kann so immer nur bis zur letzten Exception rückgängig gemacht werden. Nach dem Umkehren der letzten Änderung wird diese aus dem Log gelöscht, sodass der Schritt davor auch rückgängig gemacht werden kann.

Da beim Aufruf der Funktion noch nicht klar ist, welcher Vorgang passiert, wird die Umkehrfunktion mitsamt ihren Übergabeparametern als Antwort übergeben. So sind im Frontend alle Informationen gegeben. Sollte eine Exception aufgetreten sein, beispielweise dass eine Funktion nicht dynamisch aufgerufen werden kann, wird stattdessen null geantwortet.

Um unvorhergesehene Fehler zu vermeiden, schreiben die Teile der log- und undo-Funktionen nicht in das log-file.

5.4.3 Klassen

Hier wird kurz auf die Funktion und den Inhalt jeder Klasse, die im Rahmen dieses Projektes im Backend erstellt wurde, eingegangen.

PlyManager

Diese Klasse wurde von Dr. Jean-Jacques Ponciano als Grundstein zur Verfügung gestellt. Sie nutzt die Methoden der eingebundenen sjmply Bibliothek (Toewe 2017) um das Bearbeiten von einfachen ply-files zu ermöglichen. Hierbei wird nur die Geometrie der Objekte beachtet.

(Tests in PlyManagerTest)

ObjectDTO

Diese Klasse erbt von PlyManager und erweitert sie um die zusätzlichen Informationen, die für dieses Projekt nötig sind. Jede Instanz repräsentiert ein ply-file mit einem 3D-Objekt, das auf dem Server liegt. Hierdurch ermöglicht sie das Lesen, Schreiben, Bearbeiten und Verschicken der 3D-Objekte.

(Tests in ObjectDTOTest)

PointcloudDTO

Eine vereinfachte Version von PlyManager, weshalb sie nicht davon erben kann. Sie ist darauf zugeschnitten eine Punktwolke vom Server zu laden. Die Punktwolke ist dabei ebenfalls ein ply-file, welches jedoch anders aufgebaut ist als die 3D-Objekte. Dadurch dass die Punktwolke nur der Orientierung und als Referenz angezeigt werden soll, sind nur die nötigen Funktionen zum Lesen und Versenden der Informationen implementiert. So wird die Datei „pointcloud.ply“ über die Schnittstelle schreibgeschützt.

(Tests in PointcloudDTOTest)

GuiApplication

GuiApplication ist der Controller, der das Spring-Framework über http ansprechbar macht. Bindet hauptsächlich Methoden aus ObjectDTO ein, um die Funktionen (5.4.2 Funktionen) zu implementieren. Alle wichtigen Variablen, die in dieser Software statisch festgelegt sind, können am Anfang der Klasse angepasst werden.

(Tests in GuiApplicationTest)

MyLogManager

Diese Klasse enthält alle nötigen Funktionen, um das log-file zu schreiben, zu bearbeiten und zu lesen.

6 Testen der Software

Das Testen der entwickelten Software teilt sich in zwei Teile auf: automatisierte Tests für die serverseitige Datenhaltung und Tests durch Benutzer für das browserseitige Frontend und die Performance der gesamten Vorgänge.

Da dieser Teil des BauRobo-Projektes vor dem davor angesiedelten Schritt (Ponciano, Trémeau, Boochs 2019) umgesetzt wird, sind keine aussagekräftigen Beispieldaten für das Testen der Anwendung vorhanden, sodass alle Tests auf einem Beispieldatensatz aus Punktwolke und einigen wenigen stark vereinfachten Beispiel-Objekten stattfinden müssen. Die Objekte sind dabei nicht repräsentativ für die späteren Anwendungen der Software. Da die Punktwolke für ein anderes Projekt, und so auch für einen anderen Zweck, aufgenommen wurde, ist auch diese eventuell in Größe und Dichte nicht gut mit späteren Anwendungen vergleichbar.

6.1 Frontend

Die Tests im Frontend fanden erst gegen Ende der Implementierung statt, da erst spät eine repräsentative Beispielpunktwolke zur Verfügung stand. Die Anwendung wurde von mehreren Testpersonen auf verschiedenen Geräten ausgeführt. Dabei wurden zwei unterschiedliche Szenarien getestet:

Einmal sind Frontend und Backend über den Localhost verbunden und werden auf dem gleichen Gerät ausgeführt. In diesem Fall gab es keine Performance-Einschränkungen.

Im zweiten Fall wird eine Client-Server-Beziehung über zwei verschiedene Geräte mithilfe der Software „ngrok“ aufgebaut (NGROK 2021). Die dabei entstandenen Probleme wurden notiert und anschließend in der Implementierung korrigiert. Auffallend war, dass die Punktwolke in allen Fällen zu unterschiedlich starken Performance-Einschränkungen geführt hat. Auf einem Smartphone[1] und einem Tablet[2] wurde die Punktwolke gar nicht geladen. Auf einem Laptop[3] hat das Laden der Punktwolke eine Minute gedauert. Die Navigation war stockend und beim Erstellen eines neuen Objektes hat sich das Programm aufgehängt. Auf einem Rechner[4] waren Laden und Navigation stockend. Es konnte aber ein Objekt erstellt werden.

[1] CPU: Kirin 658, RAM 4GB (Huawei P10 lite)

[2] Chipsatz: Qualcomm Snapdragon 855 Octa Core 1.78-2.84GHz (Samsung Galaxy Tab S6)

[3] CPU: Intel® Core™ i5-6200U CPU @ 2.30GHz, GPU: NVIDIA GeForce 940MX, RAM: 8GB

[4] CPU: AMD Ryzen 5 2600 Six-Core Processor, GPU: NVIDIA GeForce GTX 1070, RAM: 16GB

6.2 Backend

Zum Testen des Backends wurden automatisierte Java Unit Tests für die einzelnen Methoden geschrieben. Diese befinden sich im Test-Bereich des Projektes und sind entsprechend der Klassen, die darin getestet werden, benannt. Die JUnit Tests prüfen meist eine Funktion oder Methode. Um die Daten, mit denen getestet wird, nicht zu verändern, werden die Vorgänge nach dem Überprüfen der Richtigkeit wieder rückgängig gemacht. So kann bei den Funktionen, die durch undo rückgängig gemacht werden können, dieser Vorgang ebenfalls getestet werden. Da bisher nicht die Möglichkeit besteht, auf Objekte außerhalb des „data“-Ordners zuzugreifen, liegen die Test-Dateien dort in einem eigenen Projektordner.

7 Zusammenfassung

Im Kapitel 2 Anforderungen wurden die Vorgaben für dieses Projekt beschrieben. Im Folgenden zunächst alle Punkte, welche umgesetzt wurden:

Im Backend:

- Framework zur Nutzung einer Webseite
- Daten konvertieren und nach Anfrage an Frontend übergeben
- Daten annehmen und korrekt speichern, eventuell bestehende Daten ändern
- Sicherung vornehmen

Im Frontend:

- HTML-Gerüst zur Darstellung und Navigation einer Webseite
- Viewer zur Darstellung von Daten
- Editor zur Bearbeitung von Daten
- Erkannte Objekte laden und farblich klassifiziert darstellen
- Viewer1: Punktwolke laden
- Navigation
    - In der Szene bewegen, zoomen, drehen
    - Objekt durch Anklicken/drauf tippen auswählen und farblich hervorheben
    - In Viewer1: Punktwolke nicht auswählbar
    - Objekteigenschaften anzeigen
- Bearbeitungsoptionen anzeigen
- Bearbeitung Viewer1
    - Neue Objekte mit wählbaren Eigenschaften und Position der Szene hinzufügen
    - Objekte löschen
    - Geometrie oder Position eines ganzen oder Teil-Objektes verändern
    - Klassifikation oder Attribut eines Objektes ändern
- Bearbeitung Viewer2
    - Bearbeitungsschritte für Objekt auswählen (z.B. „Schleifen“, „Streichen in Rot“)
    - Widersprüchliche Arbeitsschritte klären (z.B. „Streichen in Rot“ und „Streichen in Blau“)
- Veränderungen speichern
- Zwischenschritte speichern und „Rückgängig“-Funktion einfügen

Allgemeine Anforderungen:

- Die gesamte Darstellung und Bedienung soll benutzerfreundlich sein
- Die abgespeicherten und verwendeten Daten sollten ein einheitliches Datenformat besitzen und die Schnittstellen sollen klar definiert sein

- Für alle Änderungen sollen Sicherungen eingebunden werden, um bei Programmabsturz Daten zu behalten und die Möglichkeit zur Revidierung des letzten Arbeitsschrittes zu bieten

Also wurden die folgenden Anforderungen nicht verwirklicht:

Im Backend:

- Datenhaltung auf einem Server

Die Datenhaltung ist bereits implementiert, aber das Programm läuft aktuell nur über den Localhost und kann über die Software „ngrok“ von einem anderen Endgerät genutzt werden, es ist aber noch kein Server eingerichtet.

Im Frontend:

- Beliebig viele Bearbeitungsschritte möglich machen (Liste)

Die Bearbeitungsschritte sind nicht als freie Liste implementiert. Weitere Bearbeitungsmöglichkeiten müssen erst implementiert werden, bevor sie auswählbar sind. Durch die jetzige feste Implementierung sind widersprüchliche Bearbeitungsschritte ausgeschlossen. Eine klar vorgegebene Liste an Möglichkeiten verhindert ebenfalls häufige Probleme von String-Eingaben, die zu Problemen bei der Umsetzung von Seiten des Roboters führen.

Allgemein:

- Anwendung soll auf Tablet ohne Performance- oder Bedienungseinschränkungen laufen

Das Programm ist auf einem Tablet ausführbar, jedoch verursacht die Größe der Punktwolke erhebliche Performance-Einschränkungen. Auch ist zurzeit keine Möglichkeit zum Verschieben des sichtbaren Ausschnitts der Szene auf einem Tablet gegeben, da es kein Äquivalent zur rechten Maustaste gibt.

8 Ausblick

Aufgabe dieses Projektes war ein Grundgerüst zur Überprüfung und Korrektur von Objekten sowie Festlegung von Bearbeitungsschritten. So konnten hier nicht alle wünschenswerten Funktionen implementiert und optimiert werden. Im Folgenden sind deshalb weitere Schritte, die sich als nächstes anbieten, aufgelistet:

- Die Aufgabenformulierung beinhaltet keine klaren Vorgaben der möglichen Klassen der 3D-Objekte und möglicher Bearbeitungsschritte. Diese festzulegen wäre ein wichtiger Schritt für weitere Bearbeitungen. Daraufhin könnten weitere Bearbeitungsschritte implementiert werden. Im Besonderen wäre es hilfreich sich hierbei mit den betreffenden Malerbetrieben abzusprechen.

- Da die Punktwolke momentan zu Performance-Einschränkungen führt, wäre es erforderlich die Implementierung dort zu verbessern. Wie in 3.1.2 Potree-Viewer erwähnt, bietet die Software „Potree“ performancefreundliche Funktionen zum Einbetten einer Punktwolke und ist mit Three.js kompatibel.

- Es wäre vorteilhaft, eine tiefergehende Objektbearbeitung zu ermöglichen. Zum Beispiel das Hinzufügen und Entfernen von Punkten zu bestehenden Objekten oder die Möglichkeit Objekte zu einem zusammenzufügen oder aufzuteilen.

- Ebenso wäre eine benutzerfreundlichere geometrische Bearbeitung wünschenswert, da die dreidimensionale Bearbeitung aktuell nur in Achsenrichtung implementiert ist.

- Für das Programm kann ein Server eingerichtet werden, wodurch es über eine feste URL zu erreichen wäre. Dabei ist zu überlegen, ob eine andere Softwarearchitektur sinnvoller wäre, bei welcher die Berechnungen nicht client- sondern serverseitig durchgeführt und nur die Grafik „gestreamt“ wird.

- Die parallele Nutzung der Software durch mehrere Nutzer sollte ermöglicht werden, wofür einige Punkte in der Programm-Logik geändert werden müssten.

- Durch die Möglichkeit Volumenobjekte zu erzeugen, könnten manche 3D-Objekte (z.B. Steckdosen / Türen) besser dargestellt werden. In manchen Fällen sollten Objekte auseinander ausgeschnitten werden können, damit es keine Überschneidungen gibt (z.B. Tür aus Wand).

- Da Three.js Objekte generell nur aus Dreiecken zusammensetzt und der Earcut-Algorithmus nur zwei Dimensionen berücksichtigt, wäre es empfehlenswert einen stabileren Algorithmus zur Aufteilung von Vielecken zu Dreiecken zu verwenden.

- Weiterhin sollte vor der kommerziellen Nutzung und Weitergabe die sjmply-Bibliothek ersetzt werden, da diese unter Copyleft steht.

- Im Log-Vorgang sind momentan ein paar Fälle noch nicht abgefangen. Im log-file werden sowohl Exceptions als auch undo-Schritte protokolliert, was zu Problemen bei der undo-Funktion führen könnte. Hier wäre es sinnvoll, die Protokollierung in zwei log-files aufzuteilen.

- Im Frontend wird zurzeit bei der Rückgängig-Funktion nicht abgefangen, ob noch ungespeicherte Änderungen bestehen. Dieser Fall könnte in einer erweiterten Version abgefangen werden, indem der Nutzer zum Beispiel eine Warnung erhält und die ungespeicherten Änderungen erst gespeichert oder verworfen werden müssen, bevor eine Rückgängig-Aktion möglich ist.

9 Literaturverzeichnis

Auer, Michael (2015): https://giscience.github.io/GIScene.js/index.html (abgerufen am 21.02.2021)

Dropwizard (2020): https://www.dropwizard.io/en/latest/ (abgerufen am 21.02.2021)

i3mainz: https://i3mainz.hs-mainz.de/ (abgerufen am 21.02.2021)

JSON: https://www.json.org/json-de.html (abgerufen am 21.02.2021)

NGROK (2021): https://ngrok.com/ (abgerufen am 21.02.2021)

Lightbend (2018): https://www.playframework.com/ (abgerufen am 21.02.2021)

Ponciano, Trémeau, Boochs (2019): Automatic Detection of Objects in 3D Point Clouds Based on Exclusively Semantic Guided Processes https://www.youtube.com/watch?v=5Rqfra2dSl0&feature=youtu.be (zusätzl. Erläuterungen)

Projektbeschreibung (2020): Entwicklung einer modularen robotergestützten Prozessführung für den Innenbereich im Baugewerbe („Bau-Robo“) (auch zu finden im Projektverzeichnis)

Schütz, Markus (2020): https://github.com/potree/potree http://potree.org/potree/examples/clipping_volume.html (abgerufen am 21.02.2021)

Three.js (2021): https://threejs.org/ https://github.com/mrdoob/three.js/ (abgerufen am 21.02.2021)

WebGL (2021): https://get.webgl.org (abgerufen am 22.02.2021)

Toewe, Dirk (2017): https://github.com/DirkToewe/sjmply (abgerufen am 21.02.2021)

VMware (2021): https://spring.io/ (abgerufen am 21.02.2021)

Xeolabs (2020): https://xeolabs.com/ https://xeokit.github.io/xeokit-sdk/examples/#BIMOffline_XKT_HolterTower (abgerufen am 21.02.2021)

Anhang A: Aufteilung der Projektdokumentation

1	Einleitung:	Lisa Mosis
1.1	Hintergrund:	Lisa Mosis
1.2	Zielsetzung:	Rebekka Lange
2	Anforderungen:	Rebekka Lange
3	Wahl der Software:	Lisa Mosis
3.1	3D-Viewer:	Lisa Mosis
3.1.1	Xeokit-Viewer:	Lisa Mosis
3.1.2	Potree-Viewer:	Rebekka Lange, Lisa Mosis
3.1.3	GIScene-Viewer:	Lisa Mosis
3.1.4	Three.js:	Rebekka Lange
3.2	Frameworks:	Rebekka Lange
4	Oberfläche:	Rebekka Lange
5	Implementation:	Rebekka Lange
5.1	Programmaufbau:	Rebekka Lange, Lisa Mosis
5.2	Schnittstellen:	Lisa Mosis
5.2.1	Html – JavaScript:	Rebekka Lange
5.2.2	Frontend – Backend:	Lisa Mosis
5.2.3	JSON:	Lisa Mosis
5.3	Frontend:	Rebekka Lange
5.4	Backend:	Lisa Mosis
6	Testen der Software:	Lisa Mosis
6.1	Frontend:	Rebekka Lange
6.2	Backend:	Lisa Mosis
7	Zusammenfassung:	Rebekka Lange
8	Ausblick:	Rebekka Lange, Lisa Mosis

Anhang B: Aufteilung Programmcode

Lisa Mosis:

ObjectDTO.java

PointcloudDTO.java

GuiApplication.java

MyLogManager.java

GuiApplicationTest.java

ObjectDTOTest.java

PointcloudDTOTest.java

index.html

Viewer1.html

Viewer2.html

Impressum.html

viewer_stylesheet.css

Rebekka Lange:

viewerControls.js

viewer2Controls.js

JavaScript-Teile in Viewer1.html und Viewer2.html