Abstract

Ziel dieser Bachelorarbeit ist es ein Programm zu entwickeln, mit dem man eine Visualisierung von medizinischer Daten in einer virtuellen Realität ausgeben und auswerten kann. Zuerst wurde der aktuelle Stand der Kunst untersucht und in wie weit für diese Aufgabe schon Tools vorhanden sind. Da diese durchaus Verbesserungsmöglichkeiten haben, wurde festgelegt, eine ähnliche Funktion bezüglich Schichten wie es zum Beispiel ParaView besitzt, in VR zu realisieren. Hierfür werden spezifische Anforderungen definiert, die sowohl mit VR als auch ohne funktionieren müssen. Primär muss eine Methode definiert werden, womit man einzelne Schichten aus einer Meta-Image Datei herauslesen und als Textur wiedergeben kann. Diese Implementierung für die Schichten wurde zufriedenstellend erledigt. Aufgrund von Komplikationen bei der Programmierung der Steuerung konnte hier kein zufriedenstellendes Ergebnis erzielt werden, was in der Bewertung des Projekts auch so formuliert wurde.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Durch die immer stetigere Entwicklung der Computer-Technik wird der Einsatz dieser Maschinen auch für viele weitere Sektoren genutzt, die ursprünglich dafür nicht gedacht waren. So wird mittlerweile an den Schulen Themen behandelt, die früher keine Relevanz gespielt haben, aber aufgrund der fortschreitenden Technik immer wichtiger werden, wie zum Beispiel Medienkompetenz. Doch während an Schulen - trotz vorgegebenem Lehrplan hier noch wenig Fortschritte gemacht werden[RCI] sind diese Themen in andere Gebiete schon deutlich weiter.

So soll zum Beispiel in der Medizin eine künstliche Intelligenz dafür sorgen, dass die Fehlerquote bei Diagnosen deutlich reduziert werden. Auch wurde eine künstliche Nase entwickelt, welche besser imstande ist einen Geruchssinn wahrzunehmen als das menschliche Gegenstück. [ACH] Und obwohl die Computertechnik schon in der Praxis, direkt bei den Patienten angewendet wird, ist die moderne Technologie selbst bei der Ausbildung in der Medizin schon fest verankert.

So arbeitet Oculus, die Firma hinter dem VR-Headset Oculus Rift seit 2017 mit einem Krankenhaus in Los Angeles zusammen, in dem es für die Auszubildenden VR-Simulationen bereitstellt. Dabei war diese Methode so erfolgreich, dass man diese Programme auf 11 weitere medizinische Einrichtungen ausgeweitet hat.[TGR]

Diese Entwicklung zeigt klar auf, dass der Fortschritt in der medizinischen Branche nicht nur durch die Computer-Technik verbessert wurde, sondern neuartige Technologien, wie eben die virtuellen Möglichkeiten gänzlich neue Methoden und Übungsarten bieten.

1.1 Aktuelle Stand der Kunst

Obwohl der aktuelle VR-Markt durch die zwei führenden VR-Headsets Oculus Rift und HTC Vive relativ neu erscheinen, ist die Entwicklung in der Medizin schon in allen Bereichen ziemlich fortgeschritten. So wird nicht nur die virtuelle Technologie in der Chirurgie eingesetzt, sondern werden damit auch erfolgreich Phobien bekämpft. Hier wird zum Beispiel eine für den Patienten angstauslösende Umgebung erschaffen, in der er sich seine Ängste in einer dennoch sicheren Umgebung stellen kann.[TMO]

Auch gibt es neben den rein für den medizinischen Gebrauch an Krankenhäuser etliche VR-Software, die man diesem Spektrum zuordnen kann und frei zugänglich sind. Mittlerweile sind etliche VR-relevante Spiele, die dieses Thema als Lernsimulation abdecken, erschienen.[STE]

Der aktuelle Stand der Kunst zeigt klar, dass die Nutzung von virtuellen Geräten in der Medizin immer größer wird. Dabei werden auch alle relevante Gebiete ausgefüllt, sei es durch passives Lernen oder aktives teilnehmen. Doch diese positive Entwicklung in diesem Bereich ist noch nicht abgeschlossen. Es wird angenommen, dass bis 2020 dieser und der Bereich der Augmentation einen weltweiten Markt von bis zu 2,54 Milliarden Dollar erzeugen kann. [API]

1.2 Hinführung zum Thema

Eine Recherche der aktuellen Stand der medizinischen Entwicklung hat ergeben, dass zwar fast jeder Bereich abgedeckt wird. Aber die meisten der Programme sind in sich geschlossen, von Firmen entwickelt worden oder speziell für einen Einsatzgebiet gedacht. So fehlt zum Beispiel die Möglichkeit, eine vorhandene Software überarbeiten oder anpassen zu können. So gibt es zwar schon Programme, mit denen man die Anatomie eines Menschen von Innen begutachten kann [STE], jedoch ist man hier an ein einziges Modell - eben den Menschen - gebunden.

Möchte man zum Beispiel die Gehirne von 2 Menschen untersuchen, dessen Daten in einem .mhd-/.raw-Format vorliegt, so bietet diese Software keine Möglichkeit, eine externe Quelle zu begutachten. Auch gibt es im aktuellen Entwicklungsstand keine frei zugängliches Programme, womit man eigene Daten in der VR auslesen und anschauen kann. Dabei hätte dieses Dateiformat den Vorteil, dass man jede einzelne Schicht - mit den passenden Programmen, wie zum Beispiel in ParaView - untersuchen kann. Dennoch fehlt hierfür eine geeignete Software, mit der man dies in einer virtuellen Realität, ähnlich wie die in den medizinischen Spiele, anschauen kann.

Im Rahmen dieser Arbeit soll nun untersucht werden, ob man eine virtuelle Umgebung erstellen, in der man eben solche medizinische Daten untersuchen und begutachten kann. Dabei soll der Faktor hervorgehoben werden, dass man ausschließlich frei zugängliche Software verwendet, so dass eine Weiterentwicklung jederzeit möglich ist.

1.3 Ziele und Aufbau der Arbeit

Anhand den vorhandenen Daten (mhd. -und .raw-Dateien, Code-Ausschnitte) sollen mehrere Entwicklungsziele für die Software definiert werden. Je nach Problemstellung sollen Work-Arounds gefunden werden, wodurch sämtliche Anforderungen an dem Programm erfüllt werden.

1.3.1 Das optimale Ergebnis

Das Ziel der Arbeit ist es, eine zumindest rudimentäre Funktionalität der Aufgabenstellung zu gewährleisten. Dies soll damit erfüllt werden, in dem eine Liste an Anforderungen aufgestellt wird, mit der am Ende das Programm laufen soll, unter Berücksichtigung dass es auf Hardware-Ebene nicht zu stark ausgelastet wird und gleichzeitig schnell genug läuft. Des weiteren soll es Basisfunktionen bieten, worauf bei einer eventuellen Weiterentwicklung des Programms aufgebaut werden kann.

Folgende Ergebnisse soll die Software schlussendlich erreicht haben:

- Es soll eine virtuelle Umgebung erschaffen werden, die mit einem gängigen VR-Headset (Oculus Rift) funktioniert
- Die mhd. bzw. .raw-Datei soll eingelesen werden, so dass man diese bearbeiten kann
- Man soll - möglichst ohne Verzögerung - die einzelnen Schichten der medizinischen Datei anzeigen lassen können
- Eine rudimentäre Steuerung per Oculus Touch-Controller soll gewährleistet sein

1.3.2 Vorgehensweise

Vor der Implementierung der eigentlichen Arbeit muss erst mal geklärt welche Tools in Frage kommen. Anhand den Grundlagen im 2. Kapitel wird geklärt ob diese in Einklang mit den verwendeten Hardware in Kapitel 3 gebracht werden können. Ausgehend von diesem Ergebnis wird im 4. Kapitel ein Lösungsansatz erarbeitet, womit einerseits

die Arbeit realisiert werden kann, andererseits auch die geforderten Funktionalitäten, wie z.B. die Schnelligkeit oder auch die Hardware-Anforderungen erfüllt werden. Anschließend folgt eine Beurteilung der Arbeit und einen Ausblick darauf, in weit man das Programm noch verbessern kann bzw. ob man auf diesem Programm aufbauen kann für eine Weiterentwicklung.

Kapitel 2

Grundlagen und Auswahl der Software

Um sinnvoll an ein Konzept für so eine Arbeit anfangen zu können muss erst mal geklärt werden, welche relevanten Tools und Programme in Frage kommen. Einen Stand der aktuellen Recherche ergeben, dass die folgenden 3, frei zugängliche Programme grundsätzlich in der Lage wären, eine Entwicklung in der Virtuellen Umgebung umzusetzen und die Aufgabenforderungen zu erfüllen:

- Unreal Engine 4
  - Die Unreal Engine 4 ist eine leistungsstarke, von Epic Games entwickelte Computer-Spiel-Engine. Da die primäre Programmiersprache C++ ist, welche nicht im Laufe des Hochschuls-Studium bearbeitert wurde, ist diese trotz ihrer sehr guten Performance weniger geeignet für dieses Projekt. Durch ihre Leistungsstärke wäre es aber für einen alternativen Ansatz interessant, da diese die Unterstützung für alle gängigen VR-Headsets bietet[UEN]
- Unity-Engine
  - Die Unity-Engine, welche entwickelt wird von Unity Technologies ist ebenfalls wie die Unreal-Engine eine Entwicklungsumgebung primär für Video-Spiele. Diese zeichnet sich vor allem durch ihre Benutzerfreundlichkeit aus. Als Hauptprogrammiersprache unterstützt die Engine C. Auch Python, für Skriptsequenzen, wird unterstützt.
- Vizard VR
  - Vizard VR ist eine für virtueller Realitäten spezialisierte Entwicklungsumgebung, entwickelt von WorldViz. Die Plattform, welche mit Python programmiert wird, besitzt mehrere beeindruckende VR-Demos, wodurch man

die Leistungsstärke der Umgebung erahnen kann. Da sowohl Oculus Rift als auch die HTC Vive vollständig unterstützt wird, wurde als Hauptprogramm für die Entwicklung dieser Arbeit auf dieses Programm gesetzt.

2.1 Vizard VR

Vizard VR ist eine frei zugängliche auf Python basierende Entwicklungsplattform, spezialisiert auf eine Programmierung in der virtuellen Realität. Neben der Verfügbarkeit der aktuellen Main-Stream Head-Mounted Displays (HMD), wie zum Beispiel der Oculus Rift als auch der HTC Vive, hat diese Umgebung ebenfalls den Vorteil, dass es darin - quasi als Demos - vorgefertigte Umgebungen liefert, wodurch in dieser Arbeit das Erzeugen von zum Beispiel eines virtuellen Umgebung (Räume, Texturen, usw.) wegfällt, und man sich auf andere Teile der Arbeit spezialisieren kann. Des weiteren bietet die Umgebung mehrere Programme, wie zum Beispiel Vizconnect, wodurch man einfach die Bewegungssteuerung der VR-Hardware - in diesem Fall die Oculus Touch - mit dem Hauptprogramm verbinden kann. Auch das Tutorial und die Dokumentation von Vizard ist für Anfänger geeignet. [VIZ] Daneben bietet es noch etliche Demos, womit die Fähigkeiten der Plattform demonstriert werden. [VIZ2] Neben der freien Version gibt es noch die Möglichkeit, eine kostenpflichtige Lizenz für das Programm beantragen zu können. Die Software in dieser Arbeit wurde primär mit einer kostenpflichtigen Version (Vizard 5) und der freien Version (Vizard 6) erstellt.

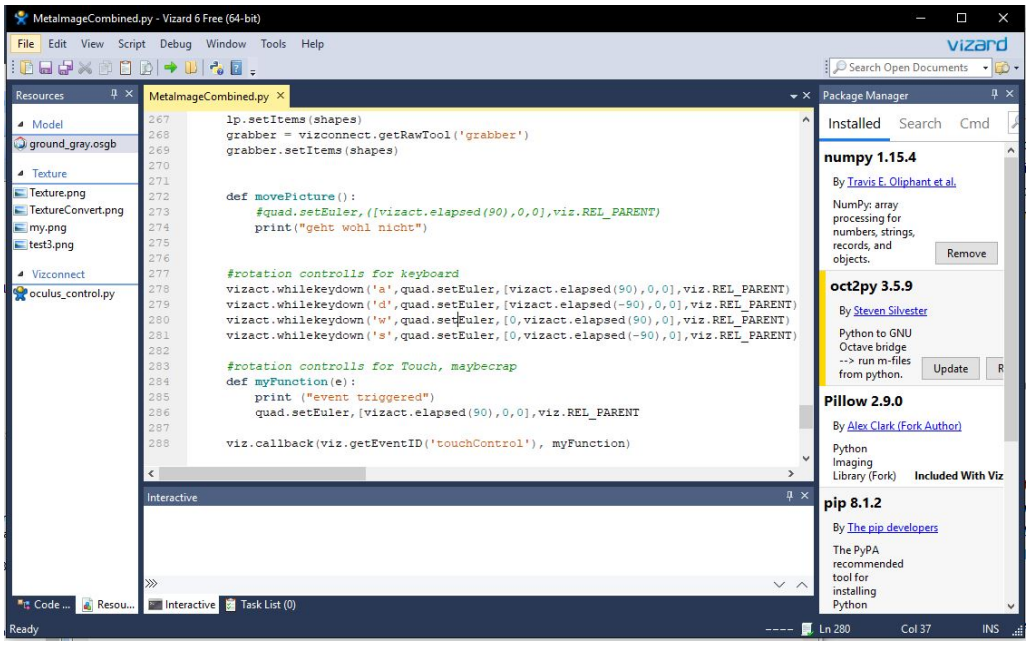

Abbildung 2.1: Benutzeroberfläche von Vizard 6

2.2 Python

Dadurch dass Vizard VR Python als Programmiersprache voraussetzt wird die ausgewählte Programmiersprache automatisch damit entschieden. Dennoch hat die Programmiersprache einige Vorteile gegenüber zum Beispiel Java oder C++ [PYT]:

- Die Programmiersprache ist Open Source. Das heißt, sie ist vollständig frei zugänglich, ohne Mehrkosten, auf jeder Plattform und ohne Einschränkungen.
- Das „Python Package Index“ bietet eine sehr große Auswahl an Module an, die in Python integriert werden können. Numpy, eines solcher Module wird zwingend für diese Arbeit verwendet, um große Arrays mit Daten manipulieren zu können.
- Ebenfalls wie Vizard bietet Python eine große Auswahl an Tutorials und Anleitungen.
- Dadurch, dass Python als höhere Programmiersprache eingestuft wird, muss man im Verhältnis zu zum Beispiel Java deutlich weniger LoC (Lines of Code) schreiben.
- Durch den kürzeren Schreibaufwand und der Eigenart weniger Schlüsselwörter zu besitzen wird eine deutlich übersichtlichere Darstellung erzeugt.

2.3 mhd.- und .raw-Dateien

Als Ausgangs-Dateien, mit denen das vorhandene Programm entwickelt werden soll, wurden mehrere Dateien - sogenannte .mhd und .raw Dateien zur Verfügung gestellt. In dieser Arbeit wird primär mit den medizinischen Bildern eines Karpfen (carp.mhd / carp.raw) gearbeitet. Diese Meta-Bilder wurde unter anderem für den medizinischen Bereich entwickelt, um zum Beispiel die Möglichkeit zu haben, Blutgefäße von Innen zu beobachten oder diese Daten auswerten zu können[ITK].

Kapitel 3

Anforderungen an die Hardware

Dadurch dass nun die zu verwendete Software für die Arbeit bestimmt wurde, müssen die Kriterien für die Hardware festgelegt werden, so dass das Programm schlussendlich funktioniert. Auch müssen die aktuelle Firmware der Hardware kompatibel mit der auswählten Software sein.

In diesem Kapitel wird dementsprechend geklärt, welche Aufgaben die verwendete Hardware regeln muss, und einen Überblick darüber gegeben, auf welche Firmware-Versionen die Software schlussendlich angepasst wurde.

3.1 Problem der Rechenleistung

Außerhalb der Verarbeitung in der VR gibt es einige gebräuchliche Tools, mit denen man die einzelnen Schichten einer mhd-/Raw-Datei anzeigen lassen kann. Da die benötigte Zeit um eine einzelne Schicht einer mhd-Datei zu visualisieren mehrere Sekunden dauern kann (zum Beispiel in einem Programm wie ParaView), muss eine Möglichkeit gefunden werden, dies mit so möglich wenig Rechenzeit zu erreichen. Hierfür muss also eine Methode gefunden werden, um das bei der Leistung der heutigen Desktop-PCs erfüllen zu können.

Da die finale Funktionalität sich dadurch auszeichnen soll, dass das Wechseln der Schichten ohne Verzögerung funktioniert, stellt sich hier eine Anforderung an die Software-Implementierung. Dieser Faktor ist vor allem deswegen wichtig, da eine zu starke Verzögerung oder eine zu niedrige Bildwiederholrate Motion-Sickness in der virtuellen Umgebung auslösen kann.[BCA] Ein Upgrade der PC-Hardware wäre in diesem Fall möglich, jedoch sollte das Programm schlussendlich auf möglichst viele Hardware-Kombinationen laufen.

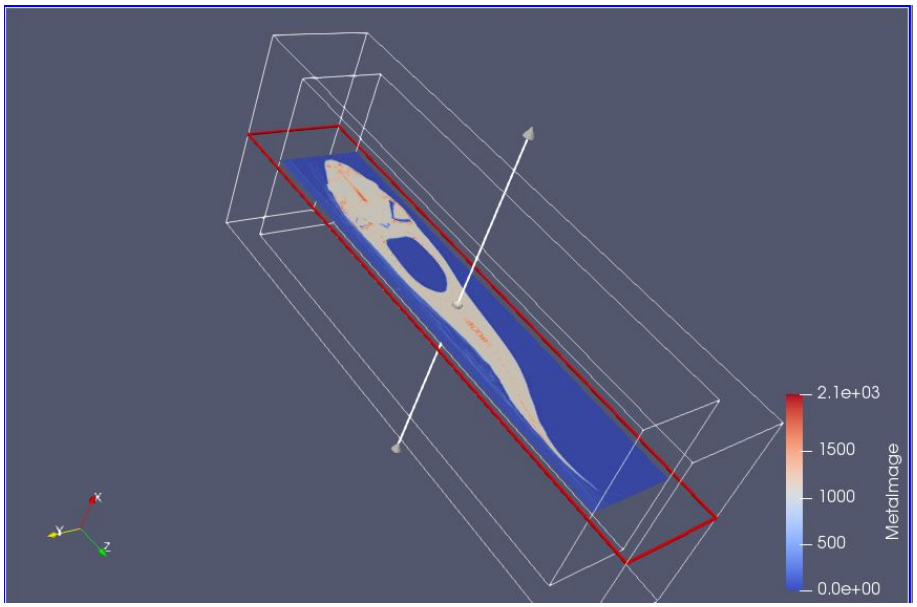

Abbildung 3.1: ParaView: Anzeigen einer einzelnen Schicht

3.2 Verwendung aktueller VR-Hardware

Da die Arbeit ausschließlich auf die Verwendung von Head-Mounted Displays in der VR basiert, muss das Programm zumindest auf eine der großen Hardware-Entwickler spezialisiert werden. Gleichzeitig muss eine Basis geschaffen werden, dass diese Software auch auf alternativen Geräten funktionieren kann.

3.2.1 Oculus Rift

Die in dieser Arbeit hauptsächlich verwendete HMD ist die Oculus Rift Consumer Version 1 (CV1) unter der Firmware-Version 709/b1ae4f61ae. Zur Unterstützung werden 2 nicht stationäre Sensoren verwendet, welche gegenüber aufgestellt werden. Damit soll erreicht werden, dass man sich zentral im Feld befindet, gleichzeitig aber keine Sensor-Abbrüche stattfinden. Da bei der Verwendung von Vizconnect (Kapitel 4.2.1) alle Elemente in Vizard 6 unterstützt werden, wurde hier die Basis für die Hardware-Bedingung geschaffen.

Da die Bildwiederholfrequenz der Oculus Rift bei 90 Hertz liegt, muss es zwingend die Anforderung an der Software sein, diese so nah wie möglich zu erreichen.

3.2.2 Oculus Touch

Ebenfalls wie das HMD muss die Hardware so gewählt werden, dass es möglich ist diese in Vizard VR zu verwenden. Da Oculus Rift standardmäßig auf den XBox One-Controller läuft, bzw. als Alternativ-Steuerung die Touch-Controller für Bewegung in der VR anbietet, ist eine Kompatibilität mit der Entwicklungs-Umgebung erforderlich. Beide

Abbildung 3.2: Oculus Rift CV1 und 2 Sensoren

Eingabe-Geräte werden ebenfalls von Vizconnect unterstützt.

Abbildung 3.3: Bedienungselement: 2 Oculus Touch-Controller

Durch die freie Handbewegungen mit der Touch-Controllern soll hier die Anforderung gestellt werden, dass man damit in der virtuellen Umgebung interagieren und die Schicht bearbeiten kann.

3.2.3 Alternative VR-Hardware

Trotz der explizit ausgewählten Hardware soll es zumindest die Möglichkeit geben, dass das Programm auch von anderer VR-Hardware unterstützt wird. Da die HTC-Vive vom Setup her ähnlich wie die Oculus Rift aufgebaut ist, unterstützt Vizard / Vizconnect

auch diese Hardware. Dennoch muss für jede Hardware ein eigenständige Datei erstellt werden (Kapitel 7.2). Dabei soll allerdings die Grundvoraussetzung geschafft werden, dass im Python-Code in Vizard selbst keine Anpassung für ein anderes HMD notwendig ist. Diese Arbeit konzentriert sich dennoch ausschließlich auf die Implementierung mit der Oculus Rift.

Kapitel 4

Lösungsansätze

Nachdem nun geklärt wurde, mit welcher Hard- und Software das Projekt realisiert wird, müssen nun im Rahmen dieser Forschung festgelegt werden, wie das Programm am Ende schlussendlich funktionieren soll. Dabei fliesen hier auch Überlegungen mit ein, die relativ früh wieder überworfen wurden. Aufgeteilt wird diese Vorgehensweise wesentlich in 2 Teile: Einerseits die Funktionalität im Hintergrund, die Abseits des Nutzers ablaufen sollen und ausschließlich in Vizard implementiert werden, andererseits die Steuerung, mit dem die Funktionen bedient werden können. Bei beiden Fällen muss berücksichtigt werden, dass man immer erst eine Testumgebung, ohne VR bzw. dessen Steuerung hat, um alle Implementierungen testen zu können.

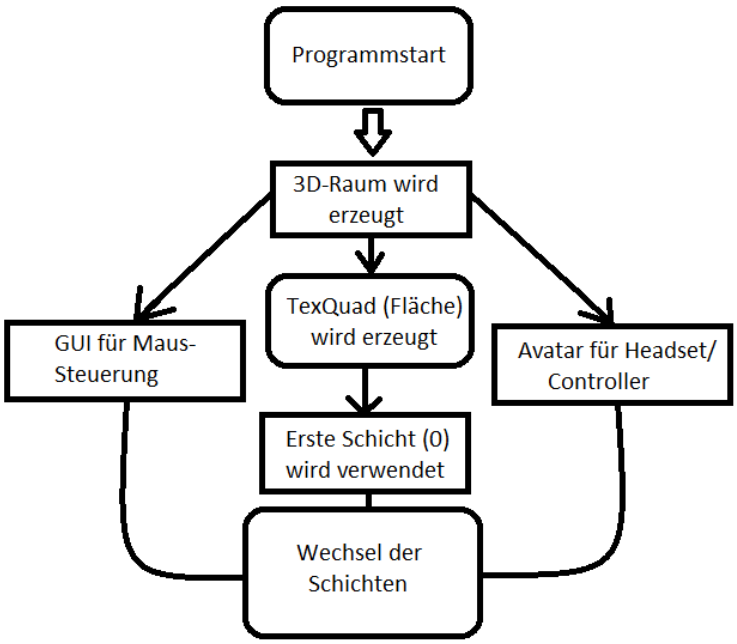

Abbildung 4.1: Schema-Bild der Funktionalität des Programm-Codes

4.1 Funktionalität im Hintergrund

Damit das Projekt am Ende erfolgreich funktioniert, müssen bestimmte Basisfunktionen vorhanden sein. Als Vorbild wurden dabei Programme verwendet, die eine ähnliche Funktion anbietet, allerdings ohne eine virtuelle Umgebung. Primär wurden dabei die beiden Programme ITK-Snap und ParaView verwendet. Beide Tools ermöglichen die Visualisierung einzelner Schichten von mhd.-/.raw-Dateien.

4.1.1 Visualisierung der Schichten

Als Erstes muss eine Möglichkeit gefunden werden, eine einzelne Schicht der Dateien auslesen und anzeigen lassen zu können. In dem vorhandenen Format muss es also möglich sein die Daten so abrufen zu können, dass man einzelne Schichten klar definiert anzeigen lassen kann. Dabei muss eine Umwandlung vorgenommen werden - zum Beispiel zu einem Numpy-Array, womit man exakt die einzelne Werte hernehmen und darstellen lassen kann.

4.1.2 Verschiebung der Ebenen

In ParaView konnte man die Verschiebung der einzelnen Ebenen erreichen, indem man eine Ebene in die Grafik hineinzieht, wodurch nach kurzer Bearbeitungsdauer die angeforderte Grafik angezeigt wird. Eine ähnliche Funktion soll hier auch realisiert werden, jedoch ohne die Bearbeitungszeit. Dabei soll eine Möglichkeit gefunden werden, anhand den vorhandenen Achsen-Werten die Grafik optisch nach vorne oder nach hinten verschieben zu können, wodurch bei jedem Vorgang die Anzeige mit dem neuen Bild aktualisiert wird.

4.1.3 Virtuelle Umgebung

Um das Programm sinnvoll ausführen zu können, muss ein geeigneter, virtueller Raum vorhanden sein. Um sich bei diese Arbeit auf die Implementierung der Schichten zu konzentrieren zu können, soll hier ein vorgefertigte Demo-Umgebung verwendet werden.

4.2 Bedienung

Um eine optimale Steuerung in der VR zu gewährleisten, sollte eine möglichst freundliche Bedienungsart gefunden werden. Da der Benutzer keine Möglichkeit hat, eine Tastatur sinnvoll mit einer VR-Brille zu bedienen, muss eine Bedienbarkeit abseits von der üblichen PC-Eingabegeräte gefunden werden.

Da man bei Tests ohne einer virtuellen Umgebung keine VR-Geräte wie die Touch-Controller bedienen kann, muss dennoch eine Möglichkeit gefunden werden, sowohl Tastatur als auch Maus bedienen zu können. Eine grafische Benutzeroberfläche, die nur auf dem Desktop-Monitor angezeigt wird, wäre eine Lösung.

4.2.1 Steuerung mit Oculus Touch

Da sowohl die linke als auch die rechte Hand in einer virtuellen Umgebung mit den Touch-Controllern simuliert werden kann, muss hierfür eine sinnvolle Bedienung gefunden werden. Als Unterstützungstool soll Vizconnect zum Einsatz kommen. Dieses Programm ist ein Tool für Vizard, dass ohne Programmierkenntnisse eine Verknüpfung von Tracker (in dem Fall der Kopf durch das HMD) mit der linken und rechten Hand ermöglicht. Dabei werden sogenannte Tools angeboten, wodurch man zum Beispiel eine Grabber-Funktion mit der rechten Hand, also dem rechten Touch-Controller verknüpfen kann.

Um für diese Arbeit eine ordentliche Steuerung anbieten zu können soll es möglich sein, über die Laserpointer bzw. Highlighter-Funktion (markieren aus der Entfernung) die Schicht auswählen und damit interagieren zu können. Des weiteren soll es möglich sein, sich ohne echte Bewegung durch die Sticks in der virtuellen Umgebung bewegen zu können. Hierfür ist es erforderlich, dass man alle Bedienungselemente des Controllers mit der zugehörigen Funktion verknüpft.

4.2.2 Alternative Steuerungen

Abseits der oben genannten finalen Steuerung soll auch eine Möglichkeit gefunden werden, das Programm ohne den Einsatz von virtuellen Geräten steuern zu können. Dabei soll es allerdings möglich sein, ohne Veränderung des Quelltextes beide Bedienungsarten verwenden zu können.

4.3 Exkurs: Fehlgeschlagene Ansätze

Durch die Recherche im Internet, wie man das Projekt am besten erfolgreich lösen kann, wurden mehrere Lösungsansätze und Prototypen erstellt, die von dem endgültigen Ergebnis deutlich abweichen. In diesem Punkt sollen kurz 2 Alternativen vorgestellt werden, die sich jedoch als falsch bzw. als nicht realisierbar herausgestellt haben.

4.3.1 Schichten darstellen aus einer .STL-Datei

Zu Beginn der Arbeit wurde ein Code-Ausschnitt bereitgestellt, der eine .mhd-Datei in ein Numpy-Array-Format umwandelt. Da zu Beginn des Projekts den Entschluss gefasst wurde, keinen vorhandenes Gerüst zu verwenden, wurden mehrere Alternativ-Methoden entwickelt, die das ähnlich, oder in einer anderen Art und Weise erledigen.

Eine der ersten Überlegungen war es, die mhd./.raw-Datei in eine .STL-Datei umzuwandeln. Dies ist eine Oberflächensprache, woraus sich die Bilder dann aus Dreiecken zusammensetzen. Erreicht werden sollte damit eine optisch deutlich schönere Darstellung. Da weder die Umwandlung weder visuell funktioniert hat, noch die Performance passend war, wurde dieser Ansatz sehr schnell wieder fallen gelassen.

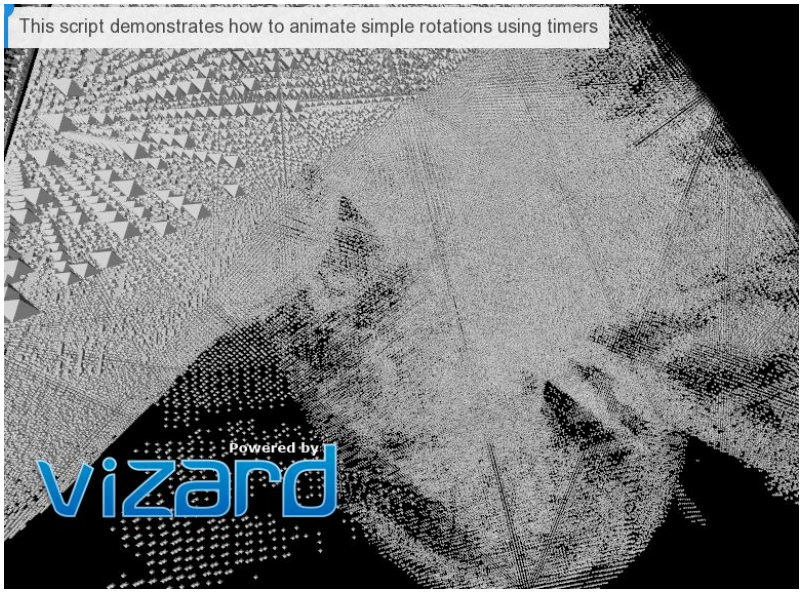

Abbildung 4.2: STL Datei Bild

Da die .STL über 700 Megabyte groß war, wurde die Erkenntnis gewonnen, dass das Laden einer ziemlich großen Datei in Vizard Probleme macht. So hat der Programm-Start beim Laden dieser Datei über 30 Sekunden gedauert. Dieses Problem hat nur unterstrichen, dass dieser Ansatz nicht realisierbar ist.

4.3.2 Visualisierung der Schicht in 3D

Da die finale Implementierung auf einer 2D-Fläche basiert (ein Rechteck), wurde anfangs überlegt, ob es nicht möglich wäre die Darstellung auf einer 3D-Fläche, wie zum Beispiel einen Kreis zu projektieren. Ziel hierbei wäre es gewesen, dass sich anhand der Werte der Schicht die 3D-Fläche sich dieser anpasst, um quasi ein 3-dimensionales Objekt zu haben.

```
#Texture on 3d object
textureExample = viz.add('TextureConventNumber180.png')
object3d = vizshape.addBox(size=(6.0,4.0,4.0), splitFaces=True,pos=(0,1.8,4))
object3d.texture(textureExample,node='back')
```

Abbildung 4.3: Code-Ausschnitt des Versuchs

Zwar konnte man damit eine 3D-Fläche mit einer Textur bedecken, jedoch erlangte man damit nicht den gewünschten Effekt.

Da bei dieser Vorgehensweise in kurzer Zeit kein Fortschritt gemacht werden konnte, wurde diese Überlegung auch wieder fallen gelassen.

Kapitel 5

Imports

Für die Entwicklung des Programms werden mehrere Pakete (Imports) verwendet. Neben Standard-Imports, bei denen es nicht nötig ist ein Paket zu installieren, wurden auch mehrere Pakete über den Vizard Package Manager installiert. Die folgende Liste zählt nur die Wichtigsten auf.

Als Vorbereitung für das Programm muss man folgende Pakete und Imports in Vizard einbinden:

- os
  - Os ist ein Modul, welches System-Operationen bereitstellt. Hiermit wird der Ordner „textures“ erzeugt, in dem die Texturen temporär gespeichert werden.
- shutil
  - Shutil wird benötigt, damit immer nur eine Textur im Ordner „textures“ gespeichert wird. Würde dieser Import fehlen, würden alle Texturen in dem Ordner abgespeichert werden. Da es nicht der Sinn dieses Projekt ist, Texturen zu speichern, ist diese Funktion unumgänglich.
- Pillow Version 2.9.0 (from PIL import Image)
  - Mit diesem Paket kann man Bildmanipulationen vornehmen. Hier wird das Image-Modul verwendet. Dies ist dafür nötig, um aus den Daten eines Array ein Bild zu erzeugen.
- Numpy Version 1.15.4
  - Wird benötigt, um Numpy-Arrays-Aktionen vorzunehmen.
- vizconnect
  - Wird benötigt, um Vizconnect benutzen zu können.

Im folgenden Bild ist die Übersicht aller Imports zu sehen. Diese sind zwingend erforderlich, damit das Programm fehlerfrei bzw. überhaupt starten kann.

```
import sys
import os.path
import numpy
import numpy as np
from PIL import Image
import vizinfo
import os
import shutil
import vizconnect
```

Abbildung 5.1: Alle Imports auf eine Blick

Kapitel 6

Implementierung der Funktionen

Dieses Kapitel erklärt die vollständige Implementierung des Hauptprogramms anhand von Code-Ausschnitten und Interface-Bilder. Es beschränkt sich dabei ausschließlich auf die fertige Datei, jedoch nicht auf die Ansätze die verworfen und nicht verwendet wurden.

Um die Implementierung logisch zu gliedern, werden die einzelne Abschnitte in 3 Bereiche unterteilt und erklärt:

- Als erstes wird kurz auf den Ausgangscode des Programms eingegangen. Dieser wurde vor Beginn des Projekts zur Verfügung gestellt. Zwar wurde wie in Kapitel 4.3 erklärt versucht, einen eigenen, anderen Ansatz zu finden, schlussendlich wurde jedoch der vorgegebene Code verwendet. Dieser Code ist essenziell für das gesamte Projekt, weil man erst dadurch die Meta-Images bearbeiten kann.

- Danach wird näher auf die Implementierung der Schichten-Mechanik eingegangen, die grundsätzlich durchgehend läuft und immer funktionieren muss. Primär geht es hier darum, wie die einzelnen Schichten bzw. dessen Texturen erstellt werden und wie man auf die unterschiedlichen Schichten zugreifen kann.

- Als Beispiel werden noch auf 2 Test-Fälle eingegangen. Diese waren beim Testen die zwei wichtigsten Fälle, die zwar während des eigentlichen Programms nicht laufen, allerdings man jederzeit wieder aktivieren kann. Hierfür ist es jedoch notwendig das man den Quellcode bearbeitet bzw. die richtigen Stellen wieder auskommentiert.

- Als letztes wird noch kurz auf die Befehle eingegangen, womit man einen virtuellen Raum erzeugt. Da dieser Code aber von Vizard vorgegeben wird, ist dieser Abschnitt kurz gehalten.

6.1 Vorhandenes Code-Gerüst

Mit dem vorhandenen Code wurde die Grundfunktionalitäten geschaffen, die Daten aus einer .mhd-Datei lesen zu können. Hierfür wird der Import „Numpy“ benötigt.

```
if __name__ == '__main__':
    filename = "Carp.mhd"
    image = MetaImage(filename, doDataLoad=True)
    print('image DimSize: ', image.DimSize)
    print('image Offset:  ', image.Offset)
    print('image TransformMatrix:  ', image.TransformMatrix)
    print('image ElementSpacing: ', image.ElementSpacing)
    print('average grey value: ', image.dataArray.mean())
    print('grey value standard deviation: ', image.dataArray.std())
    print(image.dataArray)
```

Abbildung 6.1: Vorhandener Code, Teilausschnitt 1

Mit diesem Code lässt sich somit die Daten des Numpy-Arrays auslesen, womit im weiteren Teil gearbeitet werden kann.

```
class MetaImage (object):
    '''Load 3D image characteristics from a mhd file
    '''
    def __init__(self, fileName, doDataLoad=True):
        self.__dataTypeMap = \
                {'MET_UCHAR': numpy.uint8, 'MET_CHAR': numpy.int8,
                 'MET_USHORT': numpy.uint16, 'MET_SHORT': numpy.int16,
                 'MET_UINT': numpy.uint32, 'MET_INT': numpy.int32,
                 'MET_ULONG': numpy.uint64, 'MET_LONG': numpy.int64,
                 'MET_FLOAT': numpy.float32, 'MET_DOUBLE': numpy.float64}
        self.__dic = {}
        self.__loadMHD(fileName)
        self.NDims = 3
        if 'NDims' in self.__dic:
            self.NDims = int(self.__dic['NDims'][0])
        self.BinaryDataByteOrderMSB = []
        if 'BinaryDataByteOrderMSB' in self.__dic:
            self.BinaryDataByteOrderMSB \
                = bool(self.__dic['BinaryDataByteOrderMSB'][0])
        self.TransformMatrix = [1, 0, 0, 0, 1, 0, 0, 0, 1]
        if 'TransformMatrix' in self.__dic:
            self.__readPar('TransformMatrix', self.TransformMatrix,
                           float, '0', 9)
```

Abbildung 6.2: Vorhandener Code, Teilausschnitt 2

Da nun alle Daten der Grafik vorhanden sind, kann man einzelne Elemente herausschneiden, wodurch ein 2D-Bild entsteht.

6.2 Der Code im Hintergrund

Der bedeutsamste Teil dieser Arbeit sind die Funktionen, die man weder direkt bedienen kann, noch sieht.

Der wichtigste Teil der Arbeit besteht darin, dass einerseits die Textur der Schichten richtig angezeigt werden, als auch der Wechsel zu einer neuen Schicht reibungslos funktioniert. In diesem Unterkapitel werden die folgenden 3 Implementierungen näher erklärt, so dass man die Grundfunktionen des Programms versteht:

- Initialisierung und Darstellung der ersten (Dummy-)Schicht
  - Zu Programmstart muss neben dem virtuellen Raum auch die Leinwand (TexQuad) erzeugt werden, worauf die einzelnen Schichten projektiert werden. Dafür muss eine einzelne „Dummy-Schicht“ zum Programmstart erzeugt werden, welche allerdings zu den Original-Texturen gehört. Es handelt sich dabei um die erste Schicht (0), die - ähnlich wie die letzte Schicht - immer schwarz ist.
- Erstellung einer neuen Textur/Schicht während der Ausführung
  - Als zweite wichtige Hauptaufgabe des Programms zählt das Wechseln der Schicht. Hierfür musste eine Lösung gefunden werden, die dafür sorgt das eine neue Textur erstellt wird, während die alte, nicht mehr benötigte Datei überschrieben wird. Der wichtigste Aspekt dabei war es, dass dies wesentlich schneller generiert wird als in ParaView.
- Die aktuelle Textur ändern und anpassen
  - Gleichzeitig muss die neue Textur wieder auf die Leinwand übertragen werden. Dies alles muss fließend übergehen. Der relevante Aspekt dabei ist, dass die Performance in der virtuellen Umgebung nicht merkbar beeinflusst wird.

6.2.1 Initialisierung der ersten Dummy-Schicht

Zum Programmstart ist es wichtig, dass sofort eine Textur - egal ob echt oder nicht - hergestellt wird. Diese braucht man, damit auf der Leinwand, welche erstellt wird, sofort ein Bild dargestellt wird. In dieser Arbeit handelt es sich einfach um die erste Textur, die das Array liefert, also an der Stelle 0.

```
### convert to uint8 / 255 grey
#second number, from 0 to 255, all slices
testSliceConvert = image.dataArray[:, 0,:]
print(type(testSliceConvert))
testSliceConvert = testSliceConvert.astype(np.uint8)
print("Test ConvertImage", testSlice)
testSliceImage = Image.fromarray(testSliceConvert, "L")
testSliceImage.save("TextureConvert.png")
```

Abbildung 6.3: Erzeugung der ersten Textur

Die Zeile

testSliceConvert = image.dataArray[:, 0, :]

Bedeutet, dass die einzige Zahl, die hier verändert wird, die 0 ist. Damit „rutscht“ man quasi immer an der betroffenen Achse entlang, um immer wieder ein neues Bild zu laden.

In der Zeile

testSliceImage = image.fromArray(testSliceConvert, „L“)

wird das Bild aus den Array-Daten erzeugt und später gespeichert. Das „L“ bedeutet, dass es aus 8-bit-Pixel besteht und die Farben schwarz und weiß wiedergibt.

Der gesamte Code wird geladen, bevor Vizard die Anweisung bekommt, eine virtuelle Umgebung zu erzeugen. Dadurch soll erreicht werden, dass man in VR sofort ein Bild sieht, ohne auf das Laden der Textur warten zu müssen.

6.2.2 Eine neue Textur erstellen

Da das Hauptaufgabenziel der Arbeit war, eine Veränderung der Textur während des laufendes Betriebes zu ermöglichen, musste eine Methode gefunden werden die das ermöglicht.

Der nachfolgende Code-Ausschnitt zeigt den Bereich des Programms, in der dies bewerkstelligt wird:

```
#create the necessary texture, but still saved it
def createTextureOnTheFly(textureNumber):

    # create folder "textures"
    newpath = 'textures'
    if os.path.exists(newpath):
        shutil.rmtree(r'textures')
    if not os.path.exists(newpath):
        os.makedirs(newpath)

    #-> second number, from 0 to 255, all slices
    testSliceConvert = image.dataArray[:, textureNumber,:]
    testSliceConvert = testSliceConvert.astype(np.uint8)
    testSliceImage = Image.fromarray(testSliceConvert, "L")
    ImageFileName = "textures/TextureConventNumber" + str(textureNumber) + "
    print(ImageFileName)
    testSliceImage.save(ImageFileName)
    print("Created Slice Number ", textureNumber)
```

Abbildung 6.4: Die benötigte Textur wird erstellt

Als erstes wird ein Ordner erzeugt, in der die Texturen gespeichert werden. Hierfür wird die Importfunktion „os“ verwendet. Sollte der Ordner („textures“) schon erstellt worden sein, wird dieser verwendet. Andernfalls wird eben dieser neue Ordner, an dem Ort wo die Python-Datei ausgeführt wird, erstellt.

Im weiteren Verlauf des Codes wird durch die „textureNumber“ bestimmt, um welche Schicht es sich handelt. Diese liegt zwischen 0 und 255. Bestimmt wird diese durch den Slider, der in Kapitel 7.1 näher erklärt wird. Daraufhin wird dieser Code abgespeichert als

„textures/TextureConvert + die Nummer (0 - 255)“

und in den Ordner „textures“ gespeichert. Diese Methode wird jedes mal aufgerufen, wenn man an den Schieberegler für die Schichten einen neuen Wert einstellt. Dabei war zu beachten, dass es zu keine Systemabstürze bzw. Fehlermeldungen kommt, wenn man

den Regler theoretisch sehr schnell bewegt. Eigene Tests haben ergeben, dass dieses Kriterium erfüllt wurde.

6.2.3 Die Textur der Schicht ändern

Im oberen Abschnitt wurde zwar nun eine neue Textur erstellt und gespeichert, jedoch wurde die Grafik auf dem TexQuad noch nicht geupdatet. Dies geschieht durch die Methode im unteren Bild.

```
#with movement to the back (scripted, not real)
def changeSliceTexture(sliceNumber):
    TextureName = "textures/TextureConventNumber" + str(sliceNumber) + ".png"

    #default number = 3.0
    movementNumber = 3.0
    # default multiplicator
    multiNumber = 0.003921
    newMovementNumber = movementNumber + (multiNumber * sliceNumber)
    pic = viz.addTexture(TextureName)
    quad.setPosition([-.75, 2, newMovementNumber]) #put quad in view
    quad.texture(pic)
```

Abbildung 6.5: Die aktuelle Textur wird ausgegeben

In der Zeile

$$\text{pic} = \text{viz.addTexture(TextureName)}$$

wird die Leinwand mit der neuen Textur belegt. Da „TextureName“ aus exakt dem selben Wert besteht wie die neu abgespeicherte Textur (sliceNumber ist der Wert des Sliders, der gleich ist mit der Nummer der Schicht), wird das richtige Bild nun auf den TexQuad gebracht.

Die Zeile

$$\text{movementNumber} = 3.0$$

liefert dabei eine andere Funktion: Um einen gewissen „3D-Effekt“ bei der Verschiebung zu erzeugen, sorgt dieser Wert dafür, dass sich der TexQuad optisch nach hinten verschiebt. Der Startwert ist dabei die 3.0, was gleichzusetzen ist mit der Schicht 0 der mhd.-Datei. Je nachdem, wie höher die Zahl der Schicht ist, bis zum Maximalwert 255, verschiebt sie sich noch weiter nach hinten. Der Ausgangswert und auch der Multiplikator dafür wurde so festgelegt, dass bei der Darstellung in der virtuellen Umgebung einerseits der Effekt stark genug dargestellt wird, andererseits allerdings nicht zu extrem, so dass

die Leinwand immer noch in Reichweite des Benutzers sich befindet.

Der Startwert als auch der Multiplikator wurden händisch, ohne eine interne Berechnung festgelegt. Je nachdem, wie groß man den TexQuad beziehungsweise die Textur festlegt, müsste man diesen Wert neu anpassen.

Mit diesen 3 Funktionen wurde nun die Grundlage geschaffen, dass sich einerseits die Schicht sehr schnell erneuert, auf die Leinwand projektiert wird und gleichzeitig beim Start einen Standardwert lädt. Das System ist dabei so ausgelegt, dass bei Neustart des Programms der Ursprungswert wieder hergestellt wird. Das heißt, die erste Textur auf dem TexQuad ist immer bei 0, und es befindet sich nur diese Textur abgespeichert im Ordner.

6.3 Tests

Während der Entwicklung des Programms wurden mehrere Tests entwickelt, welche die vorhandene Funktionen überprüfen sollen. Während die meisten Testfälle wieder entfernt wurden, da dessen Funktion nicht mehr verwendet wird oder keinen Nutzen mehr hat, werden die folgenden zwei weiterhin verwendet. Sollte man zum Beispiel eine andere mhd.-Datei testen wollen, lohnt es sich die beiden Testfälle zu verwenden damit man diese auf Fehler überprüfen kann, oder ob es funktioniert.

Diese Testzwecken und alle weiteren, die nicht hier nicht weiter dokumentiert werden, wurden grundsätzlich auskommentiert. Allerdings soll das Programm gewährleisten, dass bei einer Reaktivierung dieser Fälle es zu keine neuen Problemen kommt.

6.3.1 Erzeugung aller Schichten

Während der Benutzung des Programms ist es nicht möglich, alle einzelne Schichten gezielt einzeln durchzugehen. Da keine Anzeige implementiert wurde, bei welcher Schicht man aktuell ist, kann man nicht speziell eine einzelne Schicht überprüfen. Außerdem werden sämtliche Schichten wieder gelöscht, sodass nur immer eine Schicht vorhanden ist.

Diese Funktion soll hierfür Abhilfe schaffen: Ähnlich wie im normalen Programm wird eine Schicht erzeugt. Jedoch werden alle 256 Schichten, sprich von 0 bis 255 nacheinander erzeugt und abgespeichert, in dem man mit einer while-Schleife alle Schichten durch iteriert.

```
### Test: iterate all images from 0 to 255
n = 0
while n != 256:
    testSliceConvert = image.dataArray[:, n,:]
    testSliceConvert = testSliceConvert.astype(np.uint8)
    testSliceImage = Image.fromarray(testSliceConvert, "L")
    ImageFileName = "TextureConventNumber" + str(n) + ".png"
    print(ImageFileName)
    testSliceImage.save(ImageFileName)
    print("Created Slice Number ", n)
    n = n+1
print("done")
```

Abbildung 6.6: Test zur Erzeugung aller Texturen / Schichten

Diese befinden sich nun im Ordner des Programm und können einzeln angesehen werden. Damit kann man schnell alle Texturen durchgehen und jede Einzelne überprüfen.

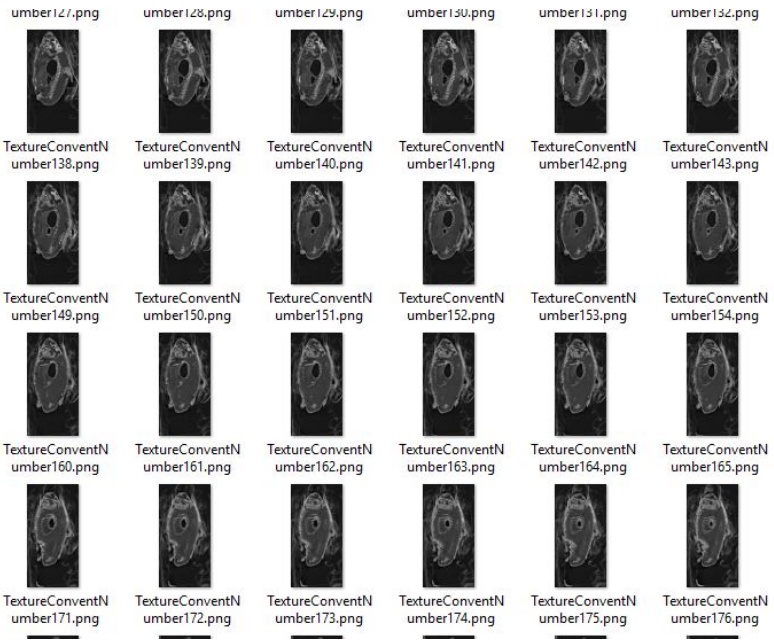

Abbildung 6.7: Erzeugung aller Texturen (Bildausschnitt)

6.3.2 Überprüfung aller Textur-Größe

Da je nachdem welche .mhd-Datei man auswählt, benötigt man dessen Maße um die Leinwand darauf anzupassen. Diese Hilfsmethode gibt die Breite und die Höhe jeder einzelnen Schicht aus.

```
###test: image size
im = Image.open(ImageFileName)
width, height = im.size
print(width)
print(height)
```

Abbildung 6.8: Test zur Überprüfung der Größe der Texturen

Dadurch kann man durch die Konsole schnell feststellen, ob eine einzelne Schicht eventuell ein Problem erzeugt oder ob man das TexQuad neu justieren muss.

6.4 Exkurs: Basic VR-Funktionen

In diesem Exkurs wird kurz dargestellt, wie die VR-Funktionen von Vizard funktionieren. Da diese standardmäßig von dem Programm vorgegeben werden, werden diese extern von der Implementierung behandelt.

Mit den 3 Zeilen

viz.setMultiSample(4)

viz.fov(60)

viz.go()

wird die VR-Einstellung grundsätzlich gestartet. SetMultiSample(4) setzt den Wert der Kantenglättung. Je höher der Wert, desto „runder“ sind die Ecken von geometrischen Figuren in der Umgebung. Jedoch beeinflusst diese Einstellung sehr stark die Performance des Programms. Dementsprechend wird der Wert auf eine niedrige Einstellung gestellt.

Mit viz.fov(60) wird das Field of View (Sichtfeld) eingestellt. Ein höheren Wert erzeugt ein größeres Sichtfeld. Jedoch fließt auch das auf die Performance mit ein.

Mit viz.go() wird das Vizard-Fenster gestartet, worin die Simulation abläuft.

Um eine optische, virtuelle Umgebung zu erzeugen, muss man eines der vorgefertigten Räume laden. Für diese Arbeit wurde folgender Raum verwendet:

```
viz.addChild('ground_gray.osgb')
```

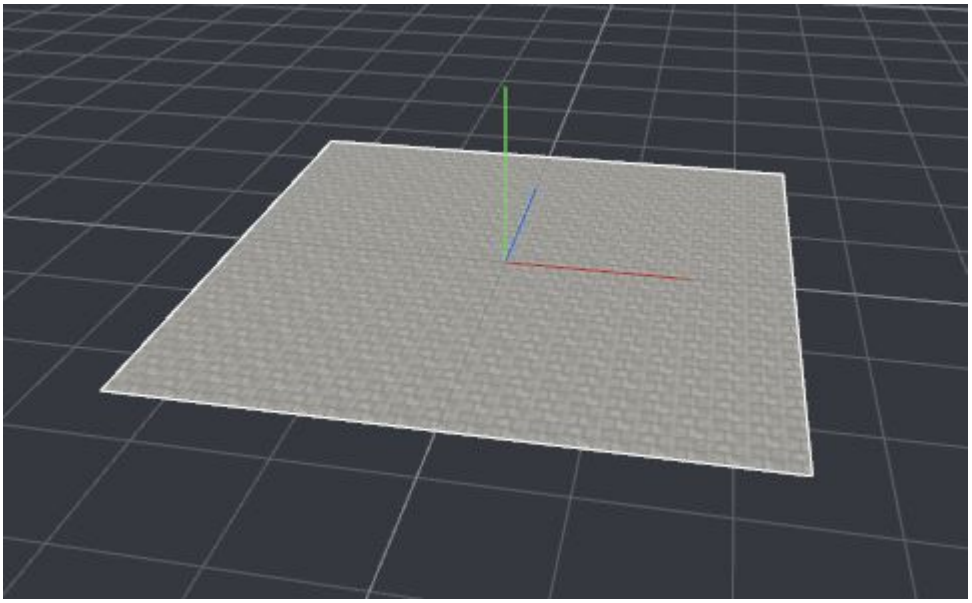

Abbildung 6.9: ground_gray.osgb, die Virtuelle Umgebung

Mit der oben beschriebenen viz.addChild wird je nach Vererbung eine andere Funktion aufgerufen. Da in diesem Fall von viz.World geerbt wurde, bedeutet dieser Aufruf das ein Objekt, hier die Umgebung „ground_gray.osgb“ zu der Szene hinzugefügt wird. Mit diesem Aufruf kann man auch alle weitere Objekte hinzufügen.

Kapitel 7

Implementierung der Steuerung

Um eine problemlose Entwicklung des Programms erfüllen zu können, muss man bedenken, dass man nicht immer jede Funktion sinnvoll in einer virtuellen Umgebung testen kann. Zwar ist das oberste Ziel der Arbeit, eine optimale Steuerung mit Eingabegeräten, die speziell für VR entwickelt wurden zu ermöglichen, dennoch darf man ein alternative Steuerung, vorzugsweise mit Maus und Tastatur, nicht vergessen.

Jedoch muss man beachten, dass beide Eingabemöglichkeiten, am Monitor und mit einem HMD, zwei unterschiedliche Oberflächen benötigen. Während man in der virtuellen Umgebung quasi direkt mit den Objekten interagiert, benötigt man für die Maus zum Beispiel Schaltflächen und Regler. Allerdings sollten beide Bedienungsmöglichkeiten durchgehend verwendbar sein, ohne dass man den Quelltext bearbeiten muss.

Hierbei liefert Vizard jedoch die Möglichkeit, die grafische Benutzeroberfläche (GUI) auszublenden, so dass man diese nur am Monitor, jedoch nicht in der virtuellen Realität sieht. Dementsprechend sollen zwei Steuerungs-Arten programmiert werden: Eine für die Bedienung am Desktop und eine speziell ausgerichtet für die Verwendung eines HMDs.

7.1 Maus-Steuerung mit GUI

Auch wenn das Projekt am Ende ausschließlich mit einem HMD funktionieren soll, ist man dennoch die meiste Zeit dabei beschäftigt das Programm per Maus und Tastatur zu steuern. Hierfür wurde ein einfacher Slider in die Oberfläche eingebaut, welche man mit dem Mauszeiger bedienen kann:

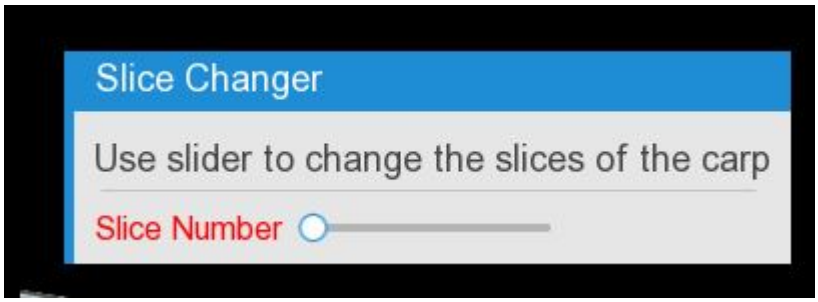

Abbildung 7.1: Regler für die einzelnen Schichten

Der Code ist dabei die Standard-Einstellung für Vizard, um einen Slider zu erstellen:

```
#Add slider in info box
#Initialize info box with some instructions
info = vizinfo.InfoPanel('Use slider to change the slices of the carp', align=viz.ALIGN_RIGHT_TOP, icon=False)
info.setTitle( 'Slice Changer' )
info.addSeparator()
slider = info.addLabelItem('Slice Number', viz.addSlider())
slider.label.color(viz.RED)
```

Abbildung 7.2: Code für den Slider

Die Funktionalität ist dabei identisch mit dem aus dem Vizard-Tutorial. Über den Regler lässt sich nun wie im Kapitel 6.2.3 beschrieben den Wert der neuen Schicht einstellen. Die grafische Benutzeroberfläche wird mit diesem Code in die rechte, obere Ecke der Vizard-Ausgabe platziert. Dabei ist diese jedoch nicht der VR sichtbar.

Da bei auskommentierter VR-Einstellungen keine Bewegung notwendig ist (die Textur wurde so festgelegt, dass diese mittig platziert wird), wurde im Gegensatz zur VR-Steuerung keine Tasten für das Bewegen der Kamera festgelegt.

Als Vorläufer für eine weitere Funktion wurden die Tasten „a, s, d, f“ mit der Rotation des Bildes belegt:

```
#rotation controlls for keyboard
vizact.whilekeydown('a',quad.setEuler,[vizact.elapsed(90),0,0],viz.REL_PARENT)
vizact.whilekeydown('d',quad.setEuler,[vizact.elapsed(-90),0,0],viz.REL_PARENT)
vizact.whilekeydown('w',quad.setEuler,[0,vizact.elapsed(90),0],viz.REL_PARENT)
vizact.whilekeydown('s',quad.setEuler,[0,vizact.elapsed(-90),0],viz.REL_PARENT)
```

Abbildung 7.3: Code für das Rotieren der Grafik

Hiermit lässt sich die Grafik entweder nach links oder rechts (a und d) oder nach oben und unten drehen (w und s). Aufgrund von Zeitmangel konnte die eigentliche Funktion nicht mehr implementiert werden: Anstelle der Grafik sollte sich eigentlich die

Schicht drehen. Da aktuell nur eine Seite der Schicht dargestellt wird sollte damit eine 3-dimensionale Ansicht ermöglicht werden.

7.2 Virtuelle Steuerung mit Vizconnect

Da die finale Version ausschließlich in der virtuellen Realität und dementsprechenden Eingabegeräten funktionieren soll, muss eine Lösung gefunden werden die genau dies schafft. Da Vizard das Tool Vizconnect integriert hat, war es die logische Entscheidung, dieses für diese Arbeit zu verwenden.

Vizconnect selbst ist eine Benutzeroberfläche, mit der man per Maus ein entsprechendes Template zusammenstellen kann. Zwar bietet es die Möglichkeit, ein Standard-Set auszuwählen, jedoch konnte man anfangs damit die Touch-Controller nicht bedienen. Erst nach einem Update von Vizard 5 auf Vizard 6 (mit einem neuen Firmware-Update für die Touch-Controller) konnte man diese Templates verwenden. Für diese Arbeit wurde jedoch ein eigenständiges, modifiziertes Set verwendet.

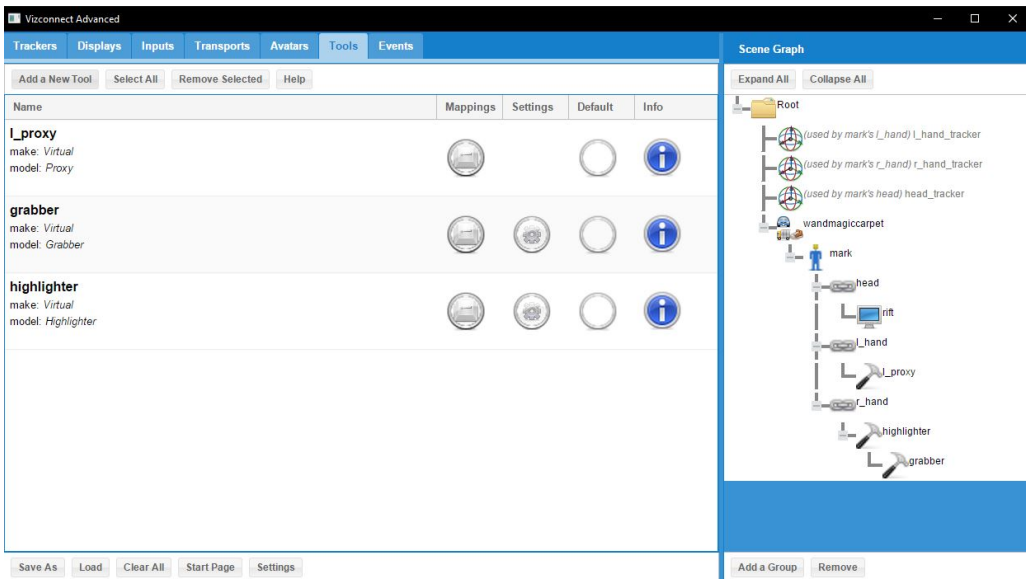

Abbildung 7.4: Vizconnect

Im Bild oben wird das normale Benutzertool von Vizconnect angezeigt. Um in dieser Aufgabe ein funktionierende Bewegung zu erzeugen, musste man folgende Schritte machen:

- Tracker
  - Unter Tracker muss man die Elemente einstellen, die in der virtuellen Realität erkannt werden. In dieser Arbeit war es der linke und rechte Touch-Controller, sowohl das HMD, ausgehend als Kamera. Jedoch werden hier nicht explizit

die Geräte selbst eingestellt, sondern nur die Option, dass man solche erkennt. Im aktuellen Stand würde das Programm erkennen, dass es 2 Hände und einen Kopf gibt. Jedoch muss man diese erst noch mit den passenden Eingabe-Möglichkeiten verknüpfen. Manuell angepasst werden musste man unter anderem die Kopfhöhe, da diese sonst im Boden versunken wäre. Durch eine Veränderung des Offsets kann man das jedoch problemlos vornehmen. Vizconnect selbst bietet noch eine Live-Vorschau. Diese ist zwar unpraktisch, da die Anzeige ein aktiviertes VR-Headset voraussetzt (man muss ein Headset aufhaben), jedoch war es ausreichend genug um kleine Veränderungen sofort zu erkennen.

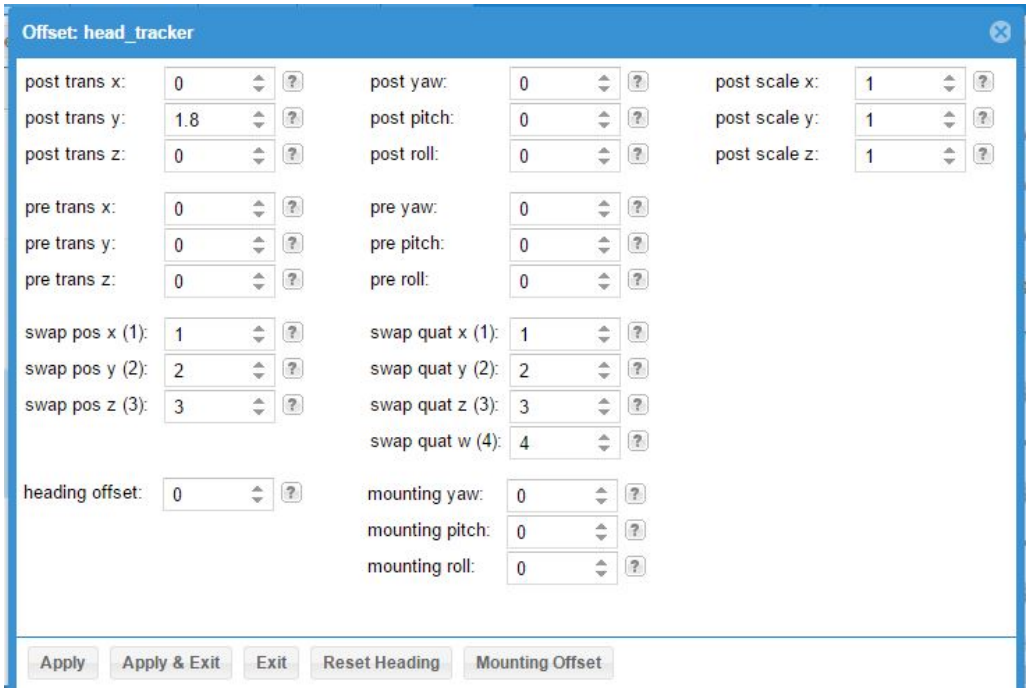

Abbildung 7.5: Einstellungen für den Head-Tracker (HMD)

- Display
  - Während man im Tracker die Geräte registriert, welche erkannt werden sollen, muss man hier festlegen, welches als Display dienen soll. In dieser Arbeit wurde die Oculus Rift als Typ HMD festgelegt. Eine generische Fenster-Anzeige ist ebenfalls möglich.

- Inputs
  - Ebenfalls wie beim Display wählt man hier die zu den beiden Händen verknüpfende Eingabe-Geräte aus. Gewählt wurde hier der linke und rechte Oculus Touch-Controller. Man ist hier jedoch nicht auf Geräte, die ausschließlich in der virtuellen Umgebung funktionieren, beschränkt. Neben Maus und Tastatur kann man auch zum Beispiel einen Controller verwenden.
- Transport
  - In dieser Einstellung war es möglich, unterschiedliche Bewegungsarten auszuwählen, die simuliert werden. Hier wurde die Wand Magic Carpet verwendet, so dass man sich frei per Stick bewegen kann.
- Avatar
  - Um alle Einstellungen, die man bisher vorgenommen hat verwenden zu können, muss man einen Avatar erstellen. Dies kann man quasi vergleichen mit einer Spielfigur. Dessen Hände (der linke und rechte Touch-Controller) und Kopf wurden damit verknüpft. In unserer Einstellung simuliert das jedoch nur einen Körper. Eine Third-Person-Sicht, so dass man das Charakter-Modell sehen kann, wäre möglich, aber nicht sinnvoll für diese Aufgabe.
- Tools
  - Der wichtigste Bereich in Vizconnect ist der Tools-Tab:

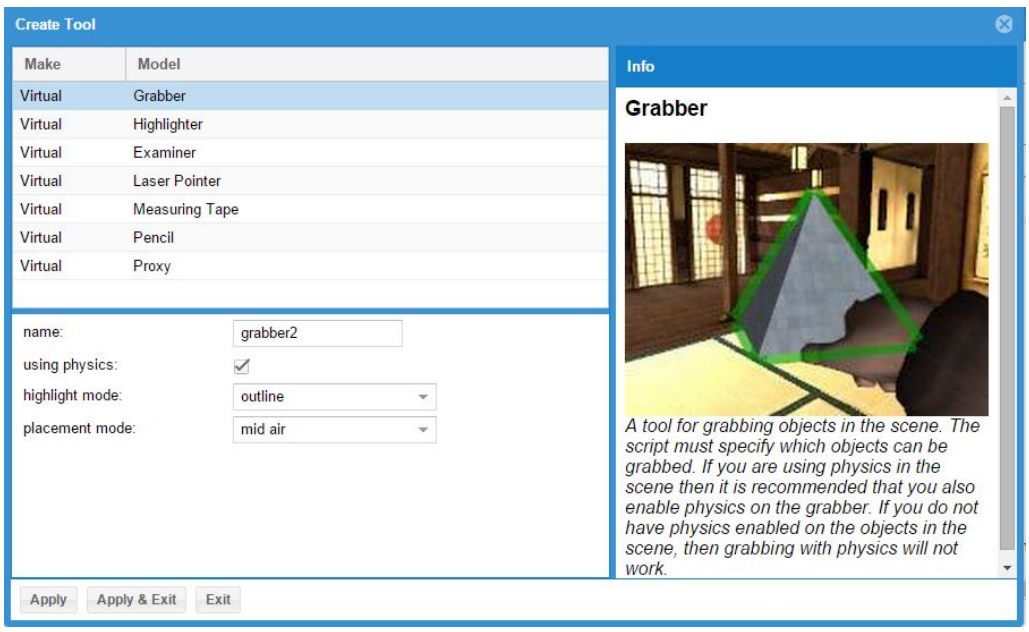

Abbildung 7.6: Funktionen im Tools-Tab

Hier lassen sich die einzelnen Funktionen, die man mit den eingestellten Input-Geräten verwenden kann, festlegen. Die wohl prominenteste Einstellung ist dabei die Grabber-Funktion: Hiermit lässt sich einstellen, dass man Objekte (wie eben zum Beispiel unsere Grafik) greifen oder herumtragen kann. Eine weitere wichtige Funktion ist der Highlighter. Hiermit kann man mit einem Laser-Pointer auf Objekte zeigen, und diese markieren.

- Scene Graph
  - Alle angelegte Optionen lassen sich nun im Scene Graph verbinden: Ausgehend von dem gewählten Transport muss man den Avatar, mit seinen 3 Eigenschaften (linke/rechte Hand, Kopf) verknüpfen. Die angelegten Tools lassen sich hier mit Drag Drop so verbinden, wie man es braucht. Damit sind die Einstellungen in Vizconnect abgeschlossen.

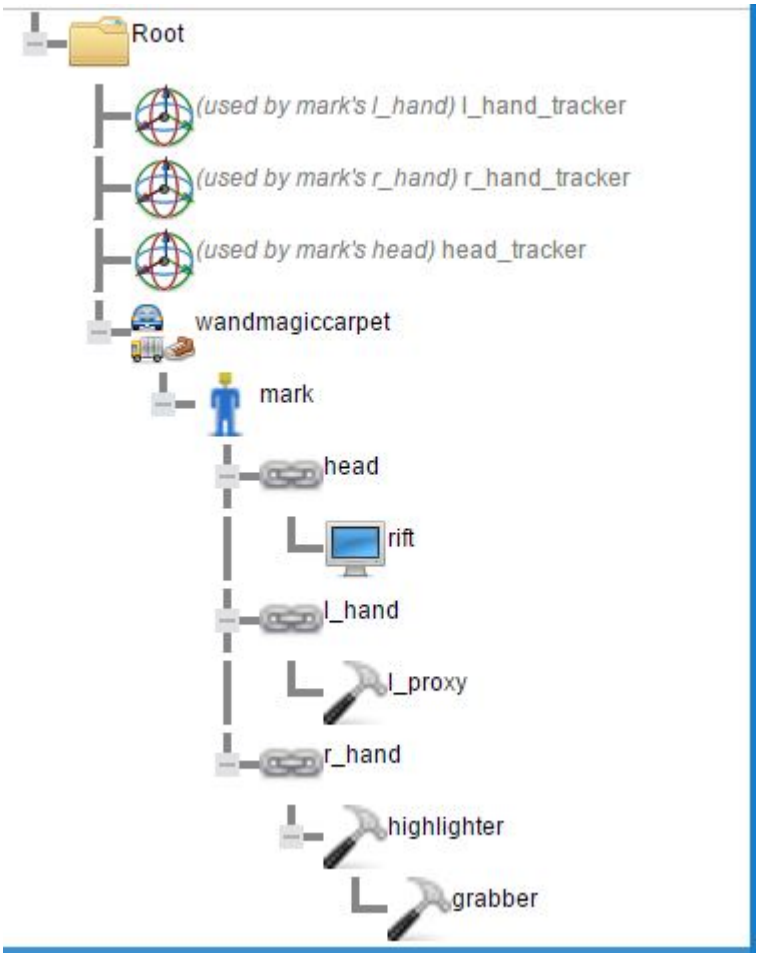

Abbildung 7.7: Fertige Scene Graph

Damit sollte nun in Vizard die Steuerung in der VR geschrieben werden. Die eigentliche Funktionalität sollte darin bestehen, dass man mit dem Highlighter das Bild auswählen kann (aus der Entfernung), und man dieses dann bewegt, wenn man zum Beispiel den Trigger des Controllers hält und damit die Grapper-Funktion auslöst. Auch sollte damit die Tasten Rotation-Funktionen der Tasten a, d, w und s auf den Controller gemappt werden.

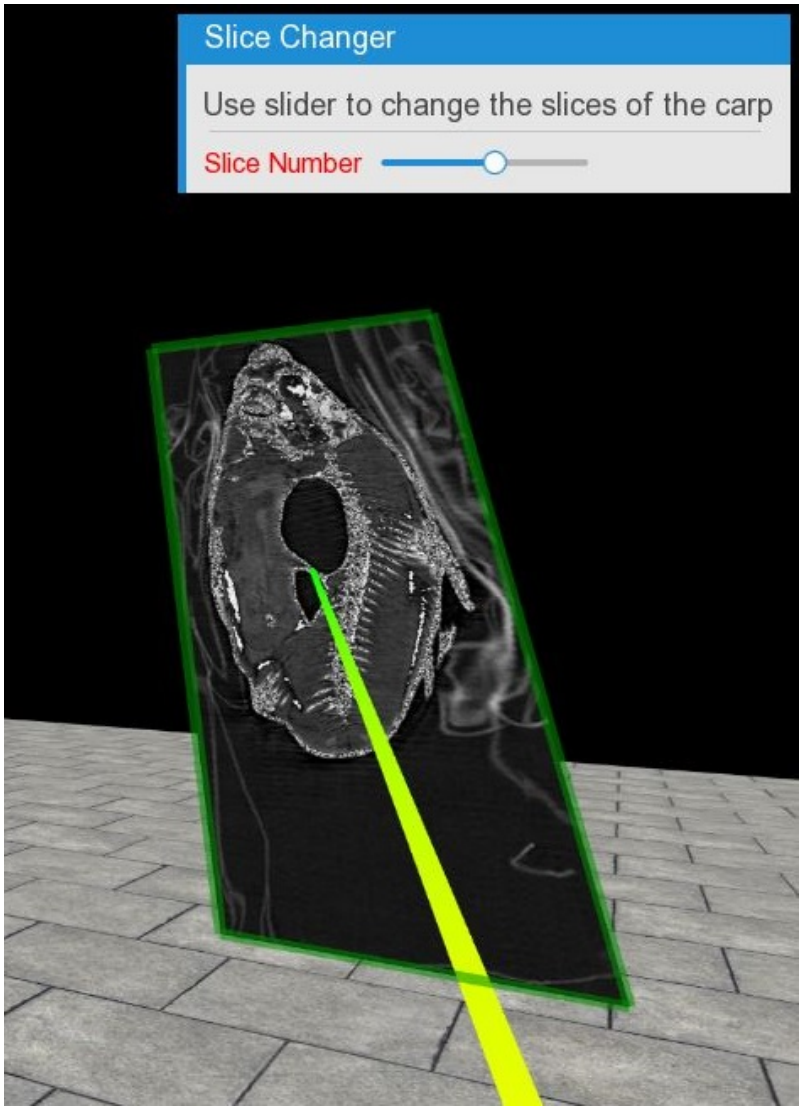

Abbildung 7.8: Highlighter in Vizard VR, ohne Funktion (Bild wurde ausgeschnitten)

Jedoch hat Vizard bzw. Vizconnect nicht von Haus aus die Fähigkeit, mehrere Funktionen miteinander zu kombinieren. Laut Recherchen im Internet [WVU] gibt es zwar die Möglichkeit, die vizconnect.py anzupassen, jedoch konnte weder das Beispiel das gegeben wurde noch ein selber geschriebenes in der virtuellen Realität verwendet, bzw. im Vizconnect angepasst werden. Zwar funktionierte das Beispiel für die Desktop-Variante, jedoch steht hier die Anwendung in der virtuellen Umgebung im Fokus.

Das nächste Problem: Wenn man die vizconnect.py (in diesem Projekt „oculus_control.py“) manuell bearbeitet, kommt eine Fehlermeldung bei Vizconnect (Abbildung 7.9). Zwar lässt sich weiterhin die gewünschten Funktionen bedienen, dennoch konnte man damit keine Verknüpfungen erzeugen. Stellenweise gingen auch die vorher eingestellten Funktionen, wie zum Beispiel der Highlighter, nicht mehr.

Abbildung 7.9: Ansicht, wenn ein Element in Vizconnect modifiziert wurde

Durch die aufgetretene Probleme und fehlender Zeit konnte keine sinnvolle Lösung für dieses Problem gefunden werden. Zwar wurde dennoch eine sinnvolle Steuerung für die Bewegung eingestellt (siehe Kapitel 7.2.1), dennoch wurde hier nicht das Ziel erreicht eine funktionierende Steuerung zu finden.

Im aktuellen Stand des Projekts muss man weiterhin eine Maus und Tastatur verwenden, um die Schichten zu wechseln bzw. um den TexQuad zu drehen. Prinzipiell müsste es möglich sein, über den Reiter Events in Vizconnect Buttons - zum Beispiel von einem X-Box-Controller - mit einer Methode in Vizard zu verknüpfen. Diese Funktion wurde zwar implementiert, jedoch hat diese nicht Funktioniert und konnte aufgrund Zeitmangel nicht weiter entwickelt werden.

7.2.1 Belegte Steuerung

Folgendes Steuerungs-Schema wurden für beide Touch-Controller in Vizconnect festgelegt:

- Linker Touch-Controller
  - Analog-Stick: Mit der Bewegung nach oben bzw. nach unten bewegt man sich im Raum entweder nach oben oder nach unten.
- Rechter Touch-Controller
  - Analog-Stick: Mit diesem bewegt man sich im Raum auf einer Ebene, je nachdem in welche Richtung man lenkt (Stick nach links, Bewegung nach links, usw.)
  - A-Taste: Wenn man die A-Taste gedrückt hält aktiviert man den Highlighter, womit das Bild hervorgehoben werden kann (grüner Rahmen). Abgesehen davon hat es keine Funktion.
  - Unterer Trigger: Mit diesem Taster aktiviert man die Grabber-Funktion, welche keine Funktion zugeordnet wurde.

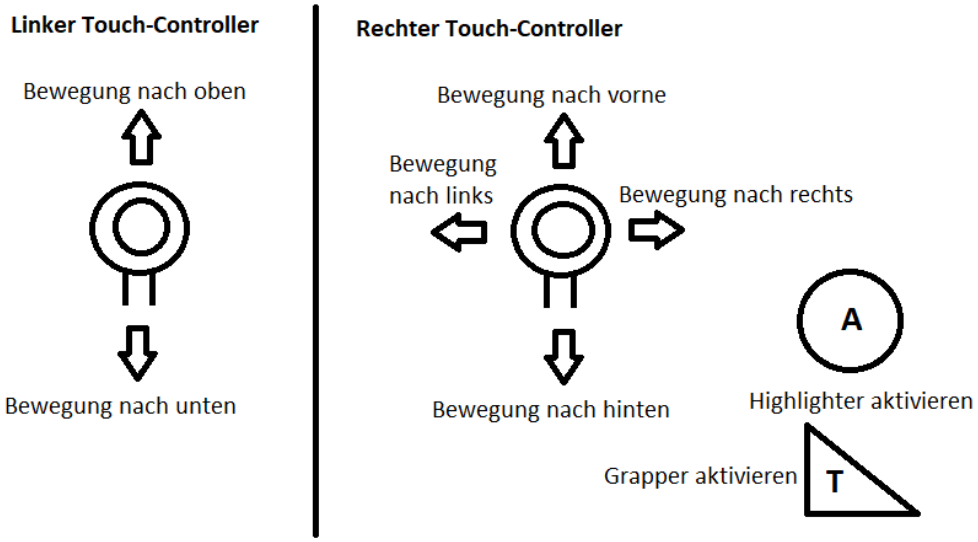

Abbildung 7.10: Steuerungs-Schema für Touch-Controller

Kapitel 8

Beurteilung

In diesem Kapitel wird das Ergebnis vorgestellt und anhand dessen den aktuellen Softwarestand bewertet.

8.1 Ergebnis

Dadurch, dass die Steuerung in der VR mit den Touch-Controllern nicht funktioniert, muss das Gerät mit Tastatur und Maus bedient werden. Dies ist grundsätzlich ein großes Problem, da man keine andere vernünftige Steuerungsmechanik hat, während man die VR-Brille verwendet. Alternativ kann man die VR-Funktion ausstellen, so dass man es komplett am Desktop steuern kann. Das Bedienelement, um die Schichten hin und her zu schieben bleibt immer bestehen, wird jedoch bei der Verwendung eines HMDs ausgeblendet.

Das Verschieben der Schichten funktioniert ohne eine sichtbare Verzögerung. Auch das Starten und Laden der Anwendung funktioniert ohne Komplikationen.

8.2 Bewertung

Durch die Probleme bei der Steuerung in einer virtuellen Umgebung, dass man keinen Grabber mit einem Laserpointer verknüpfen kann, ist die Steuerung in dieser quasi nicht möglich. Die Bearbeitung der Vizconnect-Datei wäre die einzige Möglichkeit gewesen, dieses Problem zu beseitigen. Allerdings führte das zu Probleme bei der Bearbeitung der Steuerung in Vizconnect selber, da diese grundsätzlich Komplikationen bei manipulierten Dateien hat.

Aufgrund von Zeitmangel konnte die .STL-Datei nicht über die Leinwand, wo die Schichten abgebildet werden, gelegt werden. Auch die Möglichkeit, die Schicht (nicht

die Leinwand) zu drehen fehlt. Dadurch ist die Funktionalität des Programms ziemlich beschränkt. Auch muss man den Code selbst jedes Mal ändern, möchte man eine andere .mhd-/.raw-Datei als die Carp.mhd. verwenden. Hier fehlt zum Beispiel eine Funktion, während des Programms über zum Beispiel eine Konsole eine neue Datei auswählen zu können.

Abgesehen von den fehlenden Features funktioniert das Verschieben der einzelnen Schichten deutlich schneller als zum Beispiel in ParaView. Auch die allgemeine Performance in der virtuellen Umgebung läuft problemlos.

Als Gesamtbewertung muss man attestieren, dass das Programm nicht die Funktionen bietet, die es eigentlich benötigt um zumindest Minimal zu funktionieren. Inwieweit da Vizard im Bezug auf Funktionalitäten überschätzt wurde, lässt sich schwer beurteilen, grundsätzlich wäre eine alternative Vorgehensweise auf einer anderen Plattform zumindest zu Beginn sinnvoll gewesen.

Kapitel 9

Ausblick und Verbesserungsmöglichkeiten

Im folgenden Kapitel wird kurz vorgestellt, wie man das Projekt verbessern kann. Auch wird ein kurzer Blick in die Zukunft geworfen, in wie weit das Thema Medizin und Virtueller Realität weiterentwickelt wird.

9.1 Korrekturmöglichkeiten des Projekts

Obwohl das Ergebnis in gewisse Aspekte zu gebrauchen ist, bietet es sehr viele Verbesserungsmöglichkeiten, oder haben zumindest dahingehend Aufschluss gegeben, ob es nicht doch besser wäre eine alternative Plattform zu verwenden.

- Eine aufwändigere Möglichkeit wäre es, speziell für Vizard VR Module zu schreiben, und diese dann zu verwenden. Wie im Kapitel 7.2 angemerkt wurde, ist es grundsätzlich möglich zum Beispiel die Vizconnect-Datei zu manipulieren. Je nach Aufwandsschätzung könnte man hier ansetzten und externe Tools entwickeln, die speziell für diesen Fall hier gedacht sind.

- Ein völlig anderer Ansatz an die Aufgabe könnte auch unter Umständen eine Verbesserung des Programms hervorrufen. So könnte man zum Beispiel das Projekt so umgestalten, dass man anstelle eines Avatars (also eine menschliche Person, in dem Fall uns selber) einen Roboter steuert, der einen Greifarm (zum Beispiel der rechte Touch-Controller) hat. Dies hätte den Vorteil, dass man den Greifarm mit der Grabber-Funktion verknüpfen kann, so dass dieser die einzelnen Schichten bewegt.

Eine alternative Möglichkeit wäre gewesen, wenn man ein Bedienungselement, wie zum Beispiel eine Konsole (physikalisch) in den virtuellen Raum platziert, wodurch man dadurch die Schichten durchgeht.

- Die erste Verbesserungsmöglichkeit, die man untersuchen sollte wäre ob eine alternative Entwicklungsumgebung, wie zum Beispiel die Unity- oder Unreal-Engine doch eine Verbesserung darstellt. Anhand den hier erworbene Kenntnisse sollte man überprüfen, ob die Fehler die hier aufgetreten sind durch andere Programme gelöst werden können.

9.2 Die Zukunft der VR

Da die Computer-Technik immer weiterentwickelt wird und auch die Medizin immer größere Fortschritte macht, wird der Faktor VR in der Zukunft eine noch größere Rolle in der Medizin spielen.

So hat diese Entwicklung den Vorteil, dass man in naher Zukunft möglicherweise auf die Dissektion - also die Zerstückelung - von menschlicher Leichen verzichten kann, da man dies alles auch simulieren könnte [MWA]. Auch erhofft man sich so, dass man neue Erkenntnisse über den Menschen herausfinden kann [JSC]. Dies alles zeigt, dass eine Weiterentwicklung und natürlich auch eine Nutzung der neuen, gegebenen Mittel weitaus hilfreicher sein können, als das man es vermutet.

Aber auch Abseits der chirurgischen Medizin wird man in der Zukunft immer mehr auf die virtueller Realität setzten. Da man damit auch Behandlungen gegen Ängste erfolgreich bekämpfen kann, aber die Mittel für eine Konfrontationstherapie - also der Bekämpfung der betroffenen Angst, vor Ort - begrenzt ist, kann man durch eine simulierte, virtueller Umgebung die Patienten wesentlich kostengünstiger und schneller helfen. [JWE]

Doch auch die Entwickler von solchen Geräten bleiben nicht untätig. So plant Oculus eine neue Generation von HMDs, die eine deutlich höhere Auflösung bieten als die aktuelle Ware. Insgesamt wird davon ausgegangen, dass innerhalb den nächsten 5 Jahre der große Durchbruch und völlig neue Inhalte für VR entwickelt wird.[MBA]

Grundsätzlich kann man festhalten, dass die Zukunft für VR-Anwendungen prächtig aussieht. Man kann zwar noch nicht genau vorhersagen, in welchen Bereichen es deutlich stärker ausgebaut wird, dennoch kann man davon ausgehen, dass uns noch lange Jahre die virtuelle Welten erhalten bleiben.

Kapitel 10

Fazit

Dadurch, dass die Entwicklung in der virtuellen Realität immer interessanter werden war die Verknüpfung dieses Themas mit einer weiteren großen Branche, der Medizin, sehr sinnvoll. Das diese Kombination in naher Zukunft immer relevanter wird, ist unausweichlich und auch sehr positiv zu sehen.

Die Mittel sind da, die Tools sind frei zugänglich, so dass man keine große Einstiegshürde hat, um an dieser Forschung teilzunehmen. Da die meisten Entwicklungsumgebungen noch weiter verbessert werden, und auch speziell für andere Bereiche gedachte Tools wie die Unreal- oder Unity-Engine im VR-Bereich extrem leistungsstark sind, ist die Auswahl an den Möglichkeiten vorhanden. Und auch die Head-Mounted Displays werden weiterentwickelt, so dass man davon ausgehen kann das in Zukunft das Thema VR immer wichtiger werden wird auf dem Weltmarkt und in der Forschung.

Schlussendlich hat diese Arbeit jedoch nicht das gewünschte Ergebnis geliefert. Zu viele Probleme gab es bei der Durchführung. Etliche Ansätze, zu denen Code und auch Testverfahren entwickelt wurden, sind aufgrund Fehlinformationen wieder fallengelassen worden und zum Schluss muss man auch die verwendete Software in Frage stellen, ob diese überhaupt fähig genug ist das erwünschte Ziel zu realisieren. Viele Kriterien, die man an die Software gestellt hat, konnten nicht vollständig oder teilweise gar nicht umgesetzt werden.

Grundsätzlich hätte eine bessere Auflistung der Features und Wünsche, die man am Schluss benötigt, und eine Überprüfung dessen was in Vizard VR möglich ist zu Beginn der Bearbeitung sehr viel Sinn ergeben. Auch eine Kontaktaufnahme mit den Entwicklern von Vizard (WorldViz) wäre hilfreich gewesen, um von vorneweg zu klären, was funktioniert und was nicht. Hier wurden klare Fehler gemacht, die man bei einer Weiter- bzw. Neuentwicklung unbedingt berücksichtigen soll.

Dennoch ist das Endergebnis eine Basis, worauf man aufbauen kann. Auch sind die Erkenntnisse, die diese Arbeit gebracht hat hilfreich, wenn man sich nochmal an diesem Thema versuchen möchte. Mit dieser Arbeit hat man zumindest einen Überblick, in wie weit Vizard VR hilfreich bei der Realisierung von VR-Projekte ist, und auch wo die Probleme bei dieser Software liegen.

Literaturverzeichnis

[ACH] Ann-Christin Herbe: Künstliche Intelligenz in der Medizin: Der Computer weiß, was dir fehlt - https://www.msn.com/de-de/nachrichten/wissenundtechnik/künstliche-intelligenz-in-der-medizin-der-computer-weiß-was-dir-fehlt/ar-BBPwl8u (Stand 19.11.2018)

[API] Alexander Pinker: VR Healthcare: Virtual Reality in der Medizin - https://medialist.info/201 healthcare-virtual-reality-in-der-medizin/ (Stand 19.11.2018)

[BCA] Becca Caddy: Vomit Reality: Why VR makes some of us feel sick and how to make it stop - https://www.wareable.com/vr/vr-headset-motion-sickness-solution-777 (Stand 19.11.2018)

[ITK] ITK.org: ITK/MetaIO/Documentation - https://itk.org/Wiki/ITK/MetaIO/Documentatic (Stand 19.11.2018)

[JSC] Jürgen Schreier: Mit Virtual Reality durch die Blutgefäße wandern - https://www.industry-of-things.de/mit-virtual-reality-durch-die-blutgefaesse-wandern-a-770091/ (Stand 19.11.2018)

[JWE] Jessica Wermke: VR und AR in der Medizin – VR-Reihe - https://bold-ventures.de/magazin/vr-und-ar-in-der-medizin-vr-reihe/(Stand 19.11.2018)

[MBA] Matthias Bastian: Zukunft von VR: So schätzt Oculus die nächsten fünf Jahre ein - https://vrodo.de/zukunft-von-vr-so-schaetzt-oculus-die-naechsten-fuenf-jahre-ein/ (Stand 19.11.2018)

[MWA] Matthias Wallenfels: Virtuelle Realität sorgt für Anatomie im neuen Format - https://www.aerztezeitung.de/praxis_wirtschaft/e-health/article/976048/revolution-medizinstudiu virtuelle-realitaet-sorgt-anatomie-neuen-format.html (Stand 19.11.2018)

[PYT] Python: About - https://www.python.org/about/ (Stand 19.11.2018)

[RCI] Ruth Ciesinger: Wie Schulen sich für die digitale Zukunft ändern müssen -

https://www.tagesspiegel.de/politik/schule-und-digitalisierung-wie-schulen-sich-fuer-die-digitale-zukunft-aendern-muessen/21059542.html (Stand 19.11.2018)

[STE] Steam: 3D Organon VR Anatomy - https://store.steampowered.com/app/548010/3D_Or
(Stand 19.11.2018)

[TGR] Thomas Grohganz: Oculus Education: Oculus erweitert medizinische VR-Ausbildung auf neue Institutionen - https://www.vrnerds.de/oculus-education-oculus-erweitert-medizinische-vr-ausbildung-auf-neue-institutionen/ (Stand 19.11.2018)

[TMO] ThinkMobiles: VR in Medizin: ein Neuer Durchbruch im Gesundheitswesen - https://thinkmobiles.com/de/blog/virtuelle-realitaet-medizin/ (Stand 19.11.2018)
(Stand 19.11.2018)

[UEN] Unreal Engine: Unreal Engine for AR, VR and MR - https://www.unrealengine.com/en-US/vr (Stand 19.11.2018)

[VIZ] Vizard: Getting started - https://docs.worldviz.com/vizard/latest/index.htm
(Stand 19.11.2018)

[VIZ2] Vizard: Demos - https://docs.worldviz.com/vizard/latest/World_demos.htm
(Stand 19.11.2018)

[WVU] WorldViz User Forum: Zspace Stylus beam - https://forum.worldviz.com/showthread.php
(Stand 19.11.2018)

Abbildungsverzeichnis