



Hochschule
Augsburg University of
Applied Sciences

Bachelorarbeit

Fakultät für
Informatik

Studienrichtung
Informatik

Andreas Wundlechner
Interaktive 3D-Visualisierung
medizinischer Daten

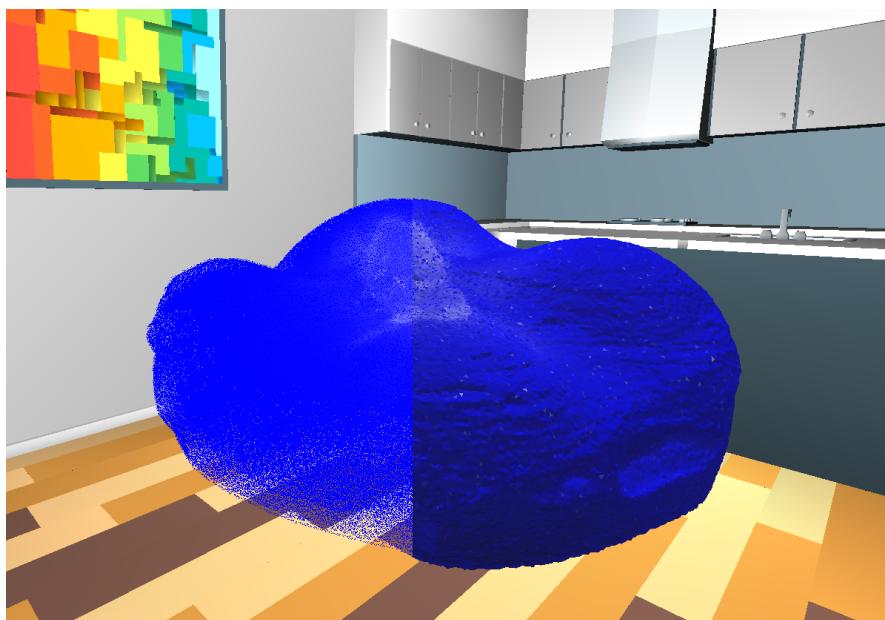
Abgabe der Arbeit am: 13.04.2018

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences
An der Hochschule 1
D-86161 Augsburg
Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Verfasser der Bachelorarbeit:
Andreas Wundlechner
a.wundlechner@gmx.de

Zusammenfassung

Bei der Behandlung von Zähnen mit Hilfe von Verblendungen und Füllungen wird abhängig von der Stärke der Füllung gesunde Zahnsubstanz entfernt. Durch die Entwicklung dünnerer Füllungen kann der Patient eine bessere Behandlung erhalten. Um mögliche Schwachstellen konkreter Füllungen zu identifizieren, werden Simulationen mit variierenden Restaurationsmethoden und Materialien durchgeführt. Die Ergebnisse liegen als volumetrische Daten vor, die bisher über konventionelle Software dargestellt wurden. Die Visualisierung der Zähne wurde hierbei durch einen flachen Monitor in der Räumlichkeit limitiert. Über die Nutzung der virtuellen Realität können die Forscher nun einen besseren Einblick erhalten. Um die VR-Darstellung zu ermöglichen, wurden die Programme ParaView (zum Auslesen der Simulationsergebnisse) und Vizard (zum Darstellen der ausgelesenen Daten) kombiniert. In der erstellten Anwendung wurde neben der konventionellen Darstellung des Zahns als Oberflächenmodell auch eine Ansicht als Punktwolke implementiert. Diese lässt den Nutzer auch Werte in der inneren Struktur erkennen und somit eine bessere Einschätzung erhalten. Durch die intuitive Steuerung können auch kleine Details einfach betrachtet werden. Wegen der zeitlichen Begrenzung wurde nur ein Teil der denkbaren Funktionen implementiert. Die Anwendung könnte in Zukunft unter anderem durch die Integration weiterer ParaView-Funktionen oder die parallele Betrachtung mehrerer Simulationen noch hilfreicher werden.



Kombinierter Screenshot zur Darstellung der Punktwolken- und Oberflächenansicht

Inhaltsverzeichnis

1. Einleitung	5
2. Analyse des Anwendungsfalls	6
2.1. Aktuelle Problemstellung	6
2.2. Anforderungen	7
3. Vergleich verfügbarer Lösungen	8
3.1. Z880	8
3.2. ParaView	9
3.3. ParaView - VR-Version	9
3.4. 3D Slicer	10
3.5. Ergebnisse des Vergleichs	11
4. Konzept - Auswahl des Entwicklungsansatzes	12
5. Umsetzung	13
5.1. Virtuelle Realität	13
5.2. Engine	13
5.3. Programmstruktur	14
5.4. Erste Prototypen mit PyVTK	14
5.5. Implementierung mit ParaView	15
5.6. Auswahl der Materialien	17
5.7. Entwicklung der Interaktion	18
6. Diskussion	20
7. Schlussfolgerung	23
8. Ausblick auf weitere Entwicklungen	24
Anhang	27
A. Details zur Zahnerhaltung	27
A.1. Allgemeine Zahnmedizin	27
A.2. Restaurierungen	28
A.3. Simulationsprozess	28

B. Bedienungsanleitung	30
B.1. Installation	30
B.2. Tastenbelegung	30
C. Beschreibung des Entwicklungsprozesses	32
C.1. Organisation der Arbeit	32
C.2. Entwicklungsumgebung	33
C.2.1. Vizard	33
C.2.2. ParaView	33
C.2.3. Blender	34
C.2.4. Google Blocks	34
C.2.5. Gimp	34
D. Weitere umgesetzte Funktionen	35
D.1. Setzen von Querschnitten	35
D.2. Auslesen einzelner Werte	36
D.3. Anpassung der Punktwolkenansicht	37
D.4. Implementierung der Umgebung	38
D.5. Steuerung mit Vizconnect	39
D.6. Automatische Steuerungserkennung	40
E. Begriffsverzeichnis	41
E.1. Grafik-Engine	41
E.2. Rendering	41
E.3. VR - Virtuelle Realität	41

1. Einleitung

Bei der Nahrungsaufnahme sind vor allem unsere Zähne die wichtigsten Komponenten. Im Alter, bei Karies oder durch Knirschen kann der wichtige Zahnschmelz beschädigt werden. Damit der Zahn möglichst gesund erhalten bleibt, kann die fehlende Zahnsubstanz durch verschiedene Materialien aufgefüllt werden.

Unter der Leitung von Prof. Dr. med. dent. Karl-Heinz Kunzelmann wird an der Poliklinik für Zahnerhaltung und Parodontologie der Ludwig-Maximilians-Universität in München an der Optimierung von Restaurierungen geforscht. Da es sich bei den Restaurierungen meist um sehr dünne Präparate handelt, sollten anwendende Fachärzte die vom Hersteller des Materials angegebene Mindeststärke nicht unterschreiten. Diese Mindeststärke orientiert sich jedoch an älteren Werkstoffen. Beim Einsetzen einer Restaurierung aus modernen Materialien müssen dadurch unnötig große Teile der noch gesunden Zahnsubstanz abgetragen werden. Um die Beschaffenheit der Restaurierungen zu verbessern, wurden Tests durchgeführt. Hierbei wurden verschiedene Restaurierungsformen und -materialien variiert, um Regeln für bessere Restaurierungen zu finden. Diese Tests wurden in Computersimulationen durchgeführt. Dadurch können mechanische Größen auch im Zahnninneren betrachtet werden und die Restaurierung kann ohne die Veränderung des restlichen Zahns variiert werden. (Für detailliertere Informationen zur Zahnmedizin, Restaurierungen und dem Simulationsprozess siehe A. Details zur Zahnerhaltung).

Bei den Simulationen (Finite-Elemente-Analyse, kurz FEA) wurde das Modell eines digitalisierten Zahns in einzelne Tetraeder unterteilt. Für diese Elemente wurden mehrere mechanische Größen berechnet, um eine qualitative Aussage über die Restaurierung treffen zu können. Die wichtigste ist hierbei die Vergleichsspannung. Sie kann mit den Materialeigenschaften verglichen werden und zur Vorhersage von Materialversagen dienen. Neben der Vergleichsspannung kann auch die Verformung der Restaurierung auf mögliche Bruchstellen hinweisen.

Deutschlandweit werden jedes Jahr über 50 Millionen Füllungen eingesetzt (Stand 2013) [1]. Eine Verbesserung der Restaurierungen könnte somit vielen Menschen eine bessere Behandlung ermöglichen.

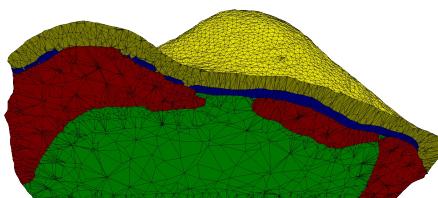


Abbildung 1.1.: Visualisierung der berechneten Tetraeder [2]

2. Analyse des Anwendungsfalls

2.1. Aktuelle Problemstellung

Die Auswahl des richtigen Materials für eine optimale Restauration ist selbst für Experten kein leichtes Unterfangen. Die komplexe Struktur des Zahns, die Kontaktflächen mit dem Antagonisten und die Stärke der Restauration sind wichtige Parameter, die eine erfolgreiche Behandlung beeinflussen.

Durch die Simulation wurden bereits wichtige Parameter berechnet. Diese können zwar durch die gängigen Programme wie ParaView visualisiert werden, jedoch ist es trotzdem schwierig die entscheidenden Details zu erkennen. Die Visualisierung beschränkt sich meist auf die Darstellung der Oberflächen. Über das Setzen von Querschnitten und das Filtern der angezeigten Daten über einen Schwellenwert können zwar bessere Einsichten gewonnen werden, aber ein gutes räumliches Verständnis zu erhalten, ist trotzdem schwierig.

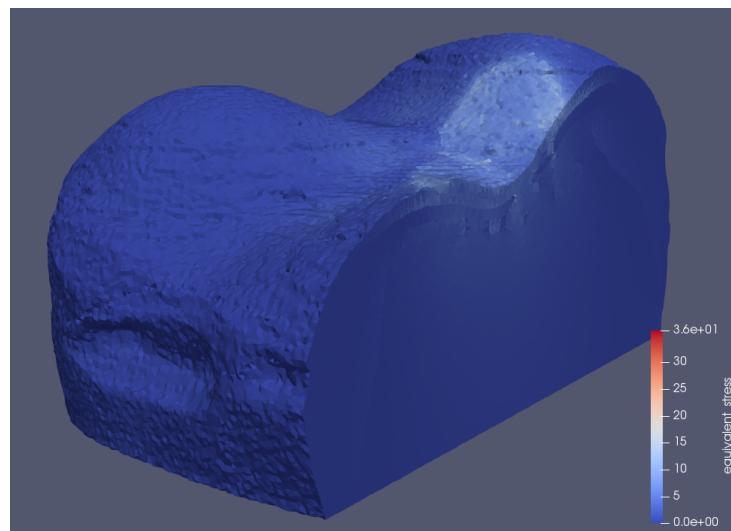


Abbildung 2.1.: Darstellung in ParaView (Eigener Screenshot)

Die Visualisierung von ParaView orientiert sich an ähnlichen Programmen aus dem Maschinenbau wie ANSYS [3] und Abaqus [4] (vgl. Abbildung 2.2). Mit diesen Programmen werden meist Konstruktionen dargestellt, bei denen vergleichsweise regelmäßige Strukturen vorhanden sind. Über das Betrachten der Oberflächen und mit Hilfe von Querschnitten kann hier ein relativ gutes räumliches Verständnis erhalten werden. Bei komplexen und unregelmäßigen Strukturen wie Zähnen ist dies nicht so einfach möglich.



Abbildung 2.2.: Darstellung in ANSYS [3] und Abaqus [4]

Die Visualisierung von dreidimensionalen Daten ist über einen herkömmlichen Monitor problembehaftet, da bei der Darstellung die Räumlichkeit verloren geht. Das Objekt muss von 3 Dimensionen auf 2 reduziert werden. Bei der Anzeige wird somit nur ein Ausschnitt der Informationen dargestellt.

2.2. Anforderungen

Da beim Simulationsprozess die Ergebnisse im Ausgabeformat von Z88OS vorliegen und diese in das VTK-Format umgewandelt werden, muss die angestrebte Lösung mit einem der beiden Formate kompatibel sein.

Die räumliche Darstellung der Simulationsergebnisse steht im Vordergrund dieser Arbeit. Um im Vergleich zu herkömmlichen Programmen das Verständnis für die Ergebnisse zu verbessern, wird die Verwendung der virtuellen Realität angestrebt. (Mehr Informationen zur virtuellen Realität sind unter der Begriffserklärung in E.3. VR - Virtuelle Realität zu finden.)

Dabei soll die Software intuitiv und direkt in der virtuellen Realität bedienbar sein. Dadurch muss die VR-Brille für Eingaben nicht abgesetzt werden. Das soll dem Benutzer helfen, die Bewertung der Ergebnisse schneller durchzuführen.

3. Vergleich verfügbarer Lösungen

Zur Darstellung der Simulationsergebnisse gibt es verschiedene Programme, die das Format von Z88OS oder VTK unterstützen. Hierbei wird das Programm gesucht, mit dem die Anforderungen bestmöglich umgesetzt werden können.

3.1. Z88O

In der FEA-Software Z88OS ist bereits der Postprozessor Z88O enthalten, der die in Textdateien vorliegenden Ergebnisse als dreidimensionale Grafik ausgeben kann. Die Ausgabe lässt jedoch nur rudimentäre Einblicke in komplexe Strukturen zu. So kann lediglich die Oberfläche des Objekts betrachtet werden. Die Darstellung der berechneten Werte erfolgt über die Einfärbung der Oberfläche. Berechnete Werte im Inneren des Objekts sind somit nicht zu erkennen. Eine detaillierte Bewertung der komplexen Struktur von Zähnen ist dadurch nicht möglich.



Abbildung 3.1.: Logo von Z88OS[5]

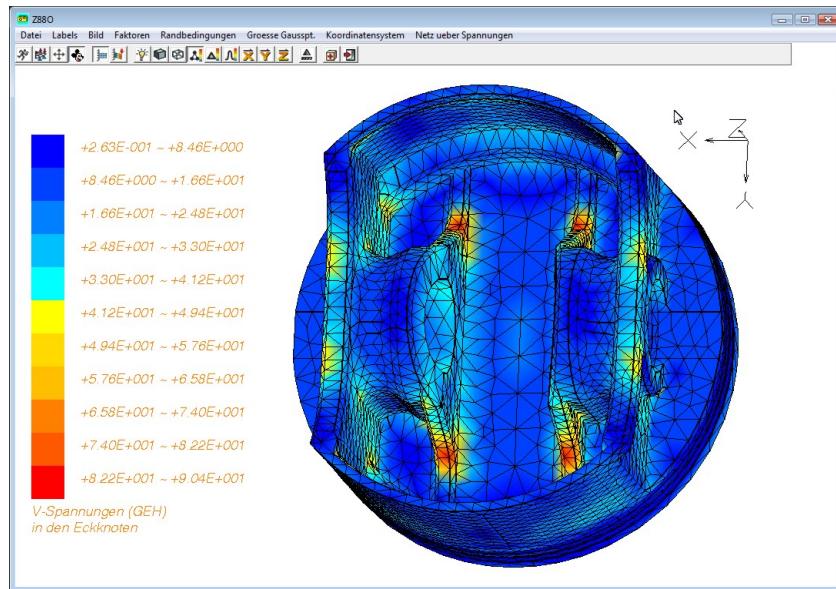


Abbildung 3.2.: Visualisierung von FEA-Ergebnissen über Z88OS [5]

3.2. ParaView

ParaView ist ein Programm zur Datenanalyse und Visualisierung. Dank seiner Skalierbarkeit und Netzwerkfähigkeit kann es auch Supercomputer zur Berechnung nutzen, um besonders große Datensätze darzustellen. Leistungsschwächere Computer können sich hierfür über das Netzwerk mit den Supercomputern verbinden, um die Steuerung zu übernehmen. Der große Funktionsumfang enthält viele verschiedene Filter, die die Datenstrukturen bearbeiten. So können unter anderem Querschnitte gesetzt, Schwellenwerte für verschiedene Parameter festgelegt und Histogramme berechnet werden.

Für die Darstellung bietet ParaView mehrere Renderer (Darstellungsfunktionen). Die meisten basieren auf dem Oberflächenmodell, wodurch der Informationsgehalt bei Zähnen relativ gering ausfällt. Hierzu gehören 3D Glyphs, Points, Surface, Surface With Edges und Wireframe. Der Point Gaussian Renderer stellt jeden Punkt innerhalb des Volumens mit einer kleinen Sphäre dar. Jedoch können hier nur den Punkten zugeordnete Werte angezeigt werden. Die wichtige Vergleichsspannung, die den Zellen zugeordnet ist, kann somit nicht dargestellt werden. Der Volume Renderer kann zwar auch die inneren Werte der Vergleichsspannung anzeigen, jedoch ist die Berechnungsdauer bei Blickwinkeländerungen relativ hoch, wodurch die Interaktivität eingeschränkt wird.

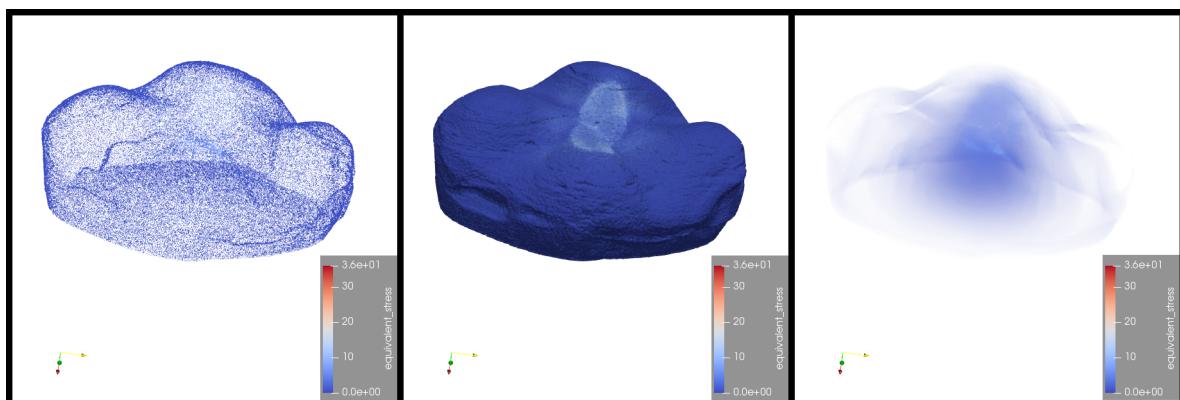


Abbildung 3.4.: Auswahl der verschiedenen Darstellungsarten, von links nach rechts: Points, Surface, Volume (Eigener Screenshot)

Das Open Source Programm wird vor allem durch das Unternehmen Kitware betreut. Der Entwickler betreut ebenfalls das Visualisation Toolkit (VTK), das von ParaView als Grafikbibliothek verwendet wird.

3.3. ParaView - VR-Version

Von ParaView befindet sich eine VR-Version in Entwicklung (aktuelle Version vom 31.10.2017; Stand 05.03.2018). Die Bedienung erfolgt - wie bei der normalen Version



Abbildung 3.3.: Logo von ParaView [6]

- über die Benutzeroberfläche am Monitor. Die anzuzeigenden Daten können über einen Knopf zur Darstellung an die VR-Brille geschickt werden [7, Abschnitt „The VR version of ParaView“]. Bei der VR-Darstellung können nur noch geringe Eingriffe in die Daten vorgenommen werden. Die Darstellungsarten entsprechen der normalen Version. Durch die höhere Leistung, die für die flüssige Darstellung innerhalb der virtuellen Realität benötigt wird, können die hilfreichen Renderer Point Gaussian und Volume nur schlecht verwendet werden.

Da es sich hierbei um eine experimentelle Version handelt, sind mehrere Fehler bei längerer Benutzung zu beobachten. Im Vergleich zu anderer VR-Software nutzt ParaView eine relativ ungewöhnliche Steuerung. So erfolgt z.B. die Auswahl eines Menüpunkts über das Neigen des Controllers.

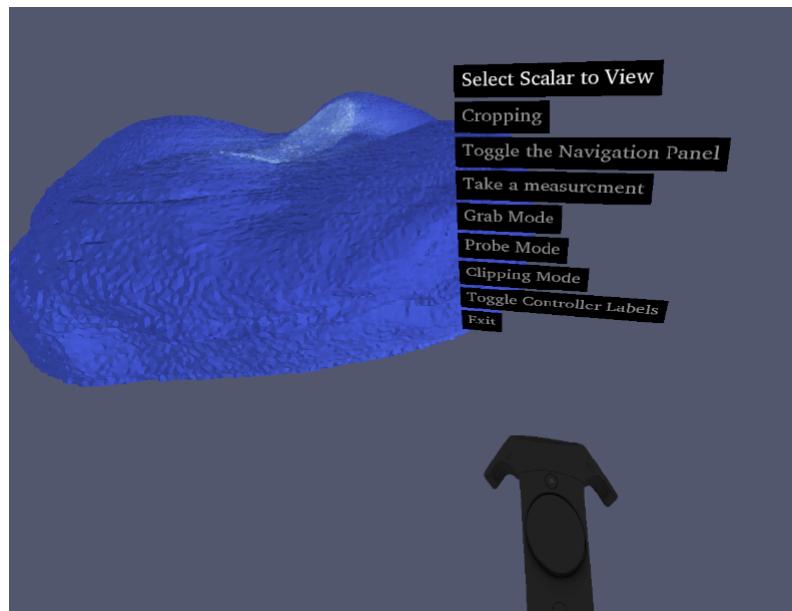


Abbildung 3.5.: Ansicht der VR-Version von ParaView (Eigener Screenshot)

3.4. 3D Slicer

3D Slicer ist ein Programm zur Analyse von medizinischen Daten. Ebenso wie ParaView, nutzt 3D Slicer das Visualisation Toolkit (VTK) als Grafikbibliothek zur Darstellung der Daten. Der Fokus des Programms liegt auf dem Import von medizinischen Bilddateien, die in 3D-Modelle umgewandelt werden. Hierbei ist die Segmentierung verschiedener Bestandteile eine wichtige Funktion.



Abbildung 3.6.: Logo von 3D Slicer [8]

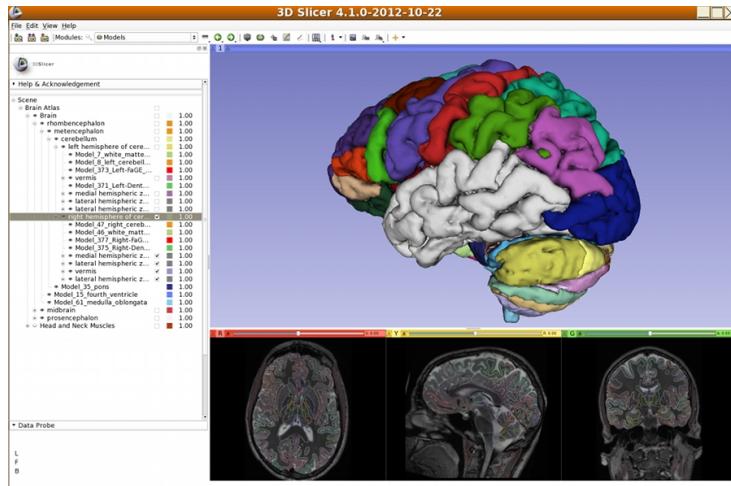


Abbildung 3.7.: Benutzeroberfläche von 3D Slicer [9]

3.5. Ergebnisse des Vergleichs

Beim Vergleich der Anwendungen weist jedes Programm Vor- und Nachteile auf.

Z88O ist bereits bei dem Softwarepaket Z88OS enthalten, das für die Finite-Elemente-Analyse (FEA) verwendet wird. Es ist zwar durch die minimalistische Darstellung für regelmäßige Strukturen geeignet, jedoch kann damit die komplexe Struktur eines Zahns nicht bewertet werden.

3D Slicer legt den Fokus auf die Generierung eines 3D-Modells aus Bilddaten und visualisiert diese. Da im Laufe des Simulationsprozesses bereits die Modelle berechnet werden (vgl. A.3. Simulationsprozess) und die Visualisierung weniger Möglichkeiten als ParaView bietet, wird 3D Slicer nicht verwendet.

ParaView bietet von den betrachteten Programmen die vielfältigsten Darstellungsoptionen. So können neben den Oberflächendaten auch die im Inneren befindlichen Daten betrachtet werden. Die Darstellung der inneren Daten geht jedoch wegen der geringen Berechnungsgeschwindigkeit mit einer Einschränkung der Interaktivität einher. Die VR-Version von ParaView hilft, ein räumliches Verständnis für die Simulationsergebnisse zu erhalten. Jedoch ist sie durch ihren experimentellen Status nur eingeschränkt nutzbar. Im Vergleich zu anderer VR-Software nutzt ParaView kaum bewährte Methoden zur Benutzerführung in der virtuellen Realität und wirkt dadurch nicht intuitiv.

Sowohl Z88O als auch 3D Slicer sind bei den Darstellungsmöglichkeiten ParaView unterlegen. ParaView hingegen liefert durch die oben genannten Einschränkungen noch nicht die optimale Lösung. ParaView besitzt als einzige Software die Kompatibilität zur virtuellen Realität. Eine bessere Unterstützung durch zusätzliche Darstellungsarten und das Einhalten von bewährten Bedienungsmethoden würde zu einem deutlich besseren Ergebnis führen.

4. Konzept - Auswahl des Entwicklungsansatzes

Um das gewünschte Ziel - eine interaktive VR-Anwendung zur Darstellung der Simulationsergebnisse - zu erreichen erscheinen zwei Möglichkeiten als zielführend.

Zum einen könnte die aktuelle VR-Version von ParaView angepasst werden. Dadurch könnte auf die bereits bestehenden Strukturen aufgebaut werden. Es müssten jedoch relativ grundlegende Entwicklungen bei der Verwendung der VR-Bibliotheken überarbeitet und ergänzt werden, um eine interaktive Erfahrung für den Nutzer zu schaffen. Um das Potential der virtuellen Realität auszuschöpfen, müssten weitere Renderer zur Darstellung implementiert werden. Der geschätzte Aufwand für die Implementierung und die Einarbeitung in die vorhandenen Programmstrukturen ist relativ hoch.

Zum anderen könnte eine neue Anwendung erstellt werden, die auf einer aktuellen Grafik-Engine basiert. Dadurch können viele Designregeln zur virtuellen Realität einfacher umgesetzt werden. Die Kompatibilität zu verschiedenen VR-Geräten könnte erhöht werden, da die Engines die Ein- und Ausgabe auf einem gemeinsamen Abstraktionsniveau verwalten. Da die verschiedenen Engines jedoch keine Unterstützung für die Ausgabedateien von Z88OS und VTK bieten, müsste hierfür erst die Kompatibilität hergestellt werden.

Im Folgenden wird die Entwicklung einer neuen Anwendung verfolgt, da hierbei das Experimentieren mit neuen Darstellungsmöglichkeiten deutlich einfacher und schneller erfolgen kann. Durch das Arbeiten auf einer höheren Abstraktionsebene bezüglich der Verwendung der virtuellen Realität kann die Anwendung deutlich intuitiver gestaltet werden. Dass die Anwendung von Beginn an auf die Forscher der Poliklinik angepasst werden kann, kommt der Bedienbarkeit ebenfalls zugute.

5. Umsetzung

5.1. Virtuelle Realität

Um die Anwendung in der virtuellen Realität darzustellen wird entsprechende Hardware benötigt. Smartphone VR-Systeme eignen sich hierfür nicht, da die Smartphones eine zu geringe Systemleistung besitzen, um die detailliert digitalisierten Zähne darzustellen. Außerdem ist die Qualität der virtuellen Realität durch fehlende Trackingsysteme geringer.

Bei den hochwertigeren Systemen konnten in den letzten Jahren größere Mengen verkauft werden, da sie im Vergleich mit professionellen Systemen zu vergleichsweise günstigen Preisen angeboten werden. Vor allem die Sony Playstation VR, Oculus Rift und HTC Vive wurden oft verkauft [10]. Für diese Arbeit eignet sich die Playstation VR nicht, da diese an die Playstation 4 Konsole gebunden ist. Die VR-Systeme Oculus Rift und HTC Vive sind in der Ausstattung sehr ähnlich. Beide Systeme verwenden Basisstationen zum Tracking der Brille und der verwendeten Controller. Ein leistungsstarker Computer berechnet jeweils das anzuseigende Bild. Die Entscheidung in dieser Arbeit fiel auf die Verwendung der HTC Vive, da die Hochschule über ein Exemplar verfügt. Darüber hinaus kann das Tracking in einem größeren Raum verwendet werden und ist genauer. Die Oculus Rift könnte bei Verfügbarkeit jedoch eine gute Alternative darstellen, da die Brille von vielen als bequemer beschrieben wird und innovativere Controller besitzt. [11]

5.2. Engine

Um die VR-Brille effizient zu verwenden, wird eine Engine benötigt. Die Kompatibilität der weit verbreiteten Unity Engine [12] und Unreal Engine [13] zu VR-Geräten begrenzt sich vor allem auf die aktuellen Geräte im Endkundensegment wie HTC Vive, Oculus Rift, Playstation VR und Smartphone Systeme. Die Kompatibilität der Grafik-Engine Vizard von WorldViz ist zu deutlich mehr VR-Ausstattungen vorhanden [14]. Dadurch können auch VR-Geräte angesprochen werden, die für ein professionelles Umfeld entwickelt wurden. Die Entwicklung mit Vizard erfolgt in der Programmiersprache Python.

WorldViz wurde 2002 gegründet und arbeitet seitdem an VR-Lösungen. Entsprechende Expertise bezüglich der virtuellen Realität sollte dadurch gegeben sein. [15]

Vizard wurde bereits bei Projekten an der Hochschule erfolgreich eingesetzt. Ansprechpartner bei Problemen sind somit vorhanden. Die Hochschule Augsburg verfügt darüber hinaus auch über eine professionelle Lizenz, die Supportanfragen an den Hersteller enthält.

5.3. Programmstruktur

Da die Anwendung verschiedene Probleme wie die Bedienung durch den Nutzer und das Auslesen und Visualisieren der Simulationsdaten löst, wurde die Anwendung in verschiedene Klassen unterteilt. Die wichtigste Klasse ist „ToothVR“. Hier werden die verschiedenen Komponenten verwaltet und übergreifende Funktionen implementiert. Als Schnittstelle zu den Simulationsergebnissen dient die Klasse „Simulation_Data“. Dabei werden ParaView-Funktionen auf die Simulationsergebnisse angewendet und in 3D-Modelle umgewandelt (weiteres dazu in 5.5. Implementierung mit ParaView). Die Klasse „GrabAndZoom“ ermöglicht das Verschieben, Vergrößern und Verkleinern von Modellen, wodurch sie intuitiv betrachtet werden können. Die Verwaltung der einzelnen Greif- und Zoomvorgänge erfolgt über die Klassen „_Grab“ und „_Zoom“. Das Modul „Controls“ wiederum nimmt die Eingaben des Benutzers entgegen und bereitet diese für eine bessere Verwendung auf.

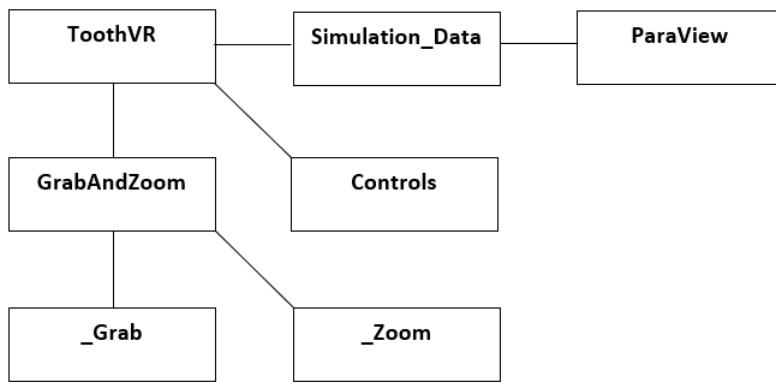


Abbildung 5.1.: Zusammenhang der verschiedenen Komponenten

5.4. Erste Prototypen mit PyVTK

Zu Beginn der Entwicklung wurde die Python Bibliothek PyVTK [16] verwendet, um die vtk-Dateien auszulesen. Dadurch konnte Zugriff auf die Zellen, deren zugehörigen Eckpunkte und die Simulationsergebnisse erhalten werden. Die Darstellung der Daten wurde als Punktwolke vorgenommen. Hierbei wurde von jeder Zelle der Schwerpunkt berechnet und über Vizard als Punkt dargestellt (vgl. Formel 5.1). Die Punkte wurden je nach darzustellenden Informationen eingefärbt.

$$s = \frac{1}{n} \sum_{i=1}^n x_i \quad (5.1)$$

Mit:

s = Schwerpunkt

n = Anzahl der Eckpunkte, bei Tetraedern 4

x = Menge der Eckpunkte der Zelle

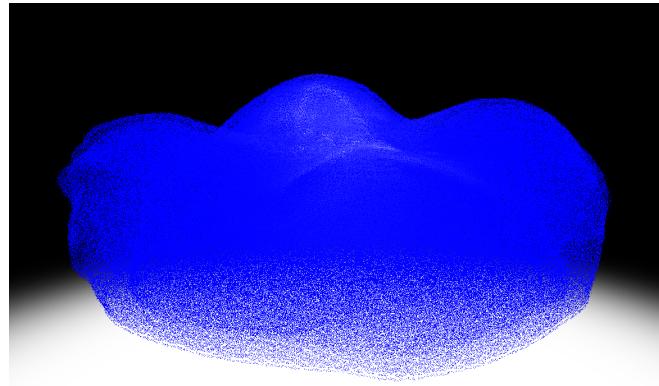


Abbildung 5.2.: Darstellung eines Zahns als Punktfolge (Eigener Screenshot)

Neben der Darstellung als Punktfolge sollte die Oberfläche auch wie in herkömmlichen Programmen angezeigt werden (vgl. Abbildung 3.4). PyVTK bietet jedoch keine Möglichkeit zu erkennen, ob die Fläche einer Zelle an der Oberfläche liegt und somit dargestellt werden soll oder nicht. Jede Seite von allen Zellen zu rendern ist keine Option, da dies zu hohem Rechenaufwand führt.

Um mit möglichst geringem Aufwand die Oberfläche zu erzeugen, wurde versucht, die vtk-Dateien über ParaView in ein für Vizard lesbares Dateiformat umzuwandeln. Beim Vergleich der kompatiblen Formate bieten nur das Polygon File Format (.ply) und das Stereolithography File Format (.stl) eine gemeinsame Schnittstelle. Bei beiden Formaten war es nur möglich die Struktur der Oberfläche zu exportieren. Die Farbe der Oberflächen konnte nicht exportiert werden.

5.5. Implementierung mit ParaView

ParaView bietet zur Bearbeitung von Daten neben der eigenen Benutzeroberfläche auch eine Python-Schnittstelle. Darüber können verschiedene Funktionen wie das Laden und Speichern von Dateien und das Setzen von Schwellenwerten und Querschnitten aufgerufen werden. Der Zugriff kann über die von ParaView bereitgestellten Python-Interpreter „PvPython“ und „PvBatch“ oder - wie in dieser Arbeit - über den Import der ParaView Bibliotheken erfolgen [17, Abschnitt „Getting Started“]. Der Hauptanwendungszweck dieser Schnittstelle ist die automatisierte Verarbeitung größerer Datenmengen. In dieser Arbeit wird die Schnittstelle jedoch zum Zugriff auf die Datenstrukturen von ParaView verwendet.

Durch das Einbinden der Python-Bibliotheken konnte schließlich auf die Oberflächen-daten zugegriffen werden. Der Ablauf sieht folgendermaßen aus: Zuerst werden die Simulationsergebnisse geladen. Auf diese Daten werden die Filter „Extract Surface“ und „Triangulate“ angewendet (Listing 5.1, Zeilen 2 und 7). Die genauen Einstellungen für die Filter wurden über die „Trace“-Funktion von ParaView ausgelesen. Die Filterergebnisse enthalten eine Liste der Dreiecke, die die Oberfläche bilden. Bei einer Iteration über die Dreiecke werden mit Hilfe von Vizard die einzelnen Polygone erstellt (Listing

5.2, Zeilen 13 bis 16 und Zeilen 30 und 31). Hierbei werden auch die Simulationswerte (z.B. Spannung) für die Farbe ausgelesen (Zeilen 7 bis 11) und die Normalenvektoren für die Beleuchtung berechnet (Zeilen 18 bis 29).

```

1 # Apply ExtractSurface and Triangulate filter
2 extractSurface = pv.ExtractSurface(Input=material_data)
3 self._filters += [extractSurface]
4 # Settings are generated by ParaView Trace function
5 extractSurface.PieceInvariant = 1
6 extractSurface.NonlinearSubdivisionLevel = 1
7 triangulate = pv.Triangulate(Input=extractSurface)
8 self._filters += [triangulate]
```

Listing 5.1: Auszug aus der Datei Simulation_Data.py ab Zeile 83

```

1 # Generate surface model by iterating over every polygon
2 polys = data.GetPolys()
3 polys.InitTraversal()
4 id_list = vtk.vtkIdList()
5 cell = polys.GetNextCell(id_list)
6 while not cell == 0:
7     # Get the value of the cell for color calculation
8     value = data.GetCellData().GetArray(color_array_name)
9         .GetValue(polys.GetTraversalLocation() / 4)
10    color = Simulation_Data._get_color(minimum, maximum, value)
11    viz.vertexColor(color[0], color[1], color[2])
12
13    # Read the points (po) of the polygon
14    po = []
15    for j in range(id_list.GetNumberOfIds()):
16        po.append(data.GetPoints().GetPoint(id_list.GetId(j)))
17
18    # Calculate the normal vector of the polygon
19    # Normal vector is needed for better lighting
20        # Calculate the cross product of vectors
21        # from point 0 to 1 and 0 to 2
22    ab = (po[1][0] - po[0][0], po[1][1] - po[0][1], po[1][2] - po[0][2])
23    ac = (po[2][0] - po[0][0], po[2][1] - po[0][1], po[2][2] - po[0][2])
24    normal = numpy.cross(ab, ac)
25        # Set the magnitude of the normal vector to 1
26    normal = normal/numpy.linalg.norm(normal)
27
28    # Create the polygon with the normal vector and the points
29    viz.normal(normal[0], normal[1], normal[2])
30    for position in po:
31        viz.vertex(position[0], position[1], position[2])
32    cell = polys.GetNextCell(id_list)
```

Listing 5.2: Auszug aus der Datei Simulation_Data.py ab Zeile 219

Für die Punktwolkenansicht kann auf PyVTK verzichtet werden, da über die ParaView-Bibliotheken ebenfalls auf die Simulationsergebnisse zugegriffen werden kann. Über die Reduzierung auf eine Bibliothek kann die Schnittstelle für die Generierung der Ober-

fläche, als auch der Punktwolkenansicht vereinheitlicht werden. PyVTK weist im Vergleich mit der ParaView-Lösung mehrere Nachteile auf. Die Ladezeit einer Datei ist bei PyVTK deutlich höher, es unterstützt keine vtk-Dateien im binären Format und PyVTK bietet deutlich weniger Funktionen zum Bearbeiten der Daten als ParaView.

5.6. Auswahl der Materialien

Bei der Betrachtung eines restaurierten Zahns ist die Beschaffenheit der Restauration sehr wichtig. Um diese hervorzuheben, sollten einzelne Materialien ein- und ausgeblendet werden können.

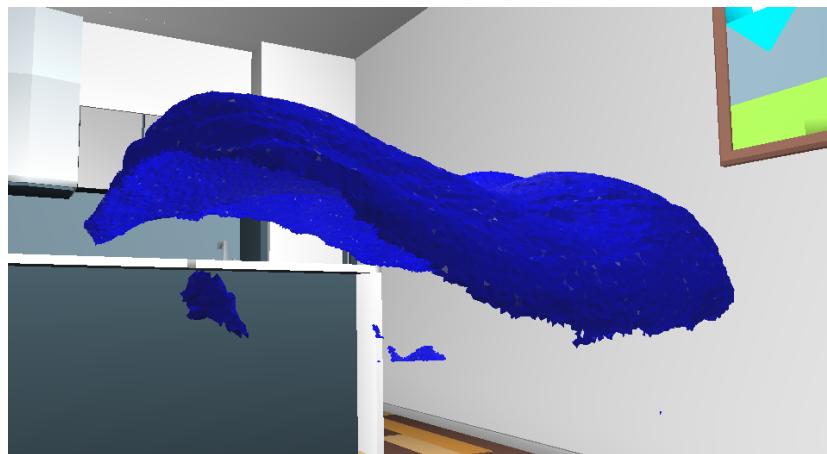


Abbildung 5.3.: Beschränkung der angezeigten Materialien auf die Restauration (Eigener Screenshot)

Damit der Wechsel zwischen den verschiedenen Materialien schnell erfolgen kann, werden vor der Generierung des 3D-Modells die Daten separiert. Die Separierung erfolgt über einen Schwellenwert, der auf das Material angewendet wird (vgl. Listing 5.3). Für die einzelnen Materialien wird anschließend die Generierung der Modelle, wie in 5.5. Implementierung mit ParaView beschrieben, durchgeführt. Die erstellten Modelle werden in eine Kindbeziehung zu einem gemeinsamen Gruppenobjekt von Vizard gesetzt, damit bei Veränderungen der Position, Rotation und Skalierung die einzelnen Objekte sich als ein zusammengehörendes Objekt verhalten.

```

1 # Iterate over every material
2 material_range = self.vtk_data_local.GetCellData()
3     .GetArray(cfg.material_name).GetRange()
4 for material in range(int(material_range[0]), int(material_range[1])+1):
5     # Separate the data by material
6     material_data = pv.Threshold(Input=data)
7     self._filters += [material_data]
8     # Settings are generated mostly by ParaView Trace function
9     material_data.Scalars = ['CELLS', cfg.material_name]
10    material_data.ThresholdRange = [material, material]
```

Listing 5.3: Auszug aus der Datei Simulation_Data.py ab Zeile 68

Die Steuerung der Funktion erfolgt über die Unterteilung des Touchpads des rechten Controllers entsprechend der Anzahl an Materialien. Die Unterteilung erfolgt dynamisch, damit eine beliebige Anzahl an Materialien verwendet werden kann.

Weitere Funktionen, die auf ähnliche Weise die Integration der ParaView-Funktionen verwenden, können im Anhang unter „D. Weitere umgesetzte Funktionen“ gefunden werden.

5.7. Entwicklung der Interaktion

Damit die Simulationsergebnisse optimal betrachtet werden können, sollte der Benutzer die Möglichkeit besitzen, die Position und Größe des Objekts anzupassen.

Um Objekte zu greifen und somit deren Position zu verändern, bietet Vizard mit dem Konfigurationstool Vizconnect die Möglichkeit, sogenannte „Grabber“ zu erstellen (mehr zu Vizconnect unter D.5. Steuerung mit Vizconnect). Mit der richtigen Konfiguration passen die „Grabber“ ihre Position automatisch an die Bewegungscontroller der HTC Vive an. Über eine definierte Taste kann nun ein Objekt festgehalten werden. Wird die Taste wieder losgelassen, bleibt das Objekt an der neuen Position. In der Standardkonfiguration ist es mit einem „Grabber“ nur möglich ein Objekt in einem vordefinierten Abstand zu seinem Ursprungspunkt zu greifen. Dieser Abstand ist dabei unabhängig von den Ausmaßen und der Skalierung des Objekts. Da das Objekt jedoch auch vergrößert werden soll, könnte es vorkommen, dass der Ursprungspunkt von den interessanten Stellen deutlich entfernt ist. Eine hinreichende Interaktion wäre somit nicht möglich. Durch ein Programm von Jeff [18, 3. Beitrag] konnte das Verhalten an die Ausmaße angepasst werden.

Die Anpassung der Größe sollte sich ähnlich wie das Zoomen in Bilder auf Touchscreens verhalten. Bei Touchscreens erfolgt der Zoom, indem an zwei Positionen das Bild mit den Fingern festgehalten und der Abstand der Finger angepasst wird (vgl. Abbildung 5.4). Durch die unmittelbare Anpassung der Skalierung befinden sich die Finger während des Zoomvorgangs immer über den ursprünglichen Punkten. Der Benutzer kann dadurch die resultierende Größe bereits im Vorfeld sehr gut abschätzen. Die VR-Programme Google Blocks [19], Google Tilt Brush [20] und MasterpieceVR von Brinx Software Inc. [21] nutzen ebenfalls diese Art zum zoomen. Diese Programme können verwendet werden, um 3D-Modelle zu erstellen.

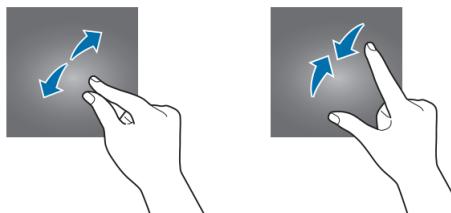


Abbildung 5.4.: Zoom bei einem Smartphone [22]

Um dieses Zoomverhalten ebenfalls zu erzielen, wurde beim Greifen des Objekts über-

prüft, ob der andere „Grabber“ bereits das selbe Objekt greift. War dies der Fall, so wurde der zweite „Grabber“ deaktiviert und die Skalierung an die Distanz zwischen den beiden „Grabbern“ angepasst.

$$s^* = s * \frac{d^*}{d} \quad (5.2)$$

Mit:

s^* = neuer Skalierungsfaktor

s = Skalierungsfaktor zu Beginn des Zoomvorgangs

d^* = aktuelle Distanz zwischen den „Grabbern“

d = Distanz zwischen den „Grabber“ zu Beginn des Zoomvorgangs

Da bei dieser Implementierung die Distanz zwischen den „Grabbern“ und dem Objekt nicht skaliert, verschiebt sich das Objekt während des Zooms sehr kontraintuitiv. Durch die Verwendung der „Grabber“ von Vizard konnte hier nichts angepasst werden. Die Lösung hierfür war eine eigene Implementierung des Greifens und Zoomen.

Beim Greifen wird das zu greifende Objekt in eine Kindbeziehung zu einem Objekt gesetzt, das der Position des Controllers folgt. Diese Kindbeziehung hat zur Auswirkung, dass der Abstand und die Rotation des Objekts in Bezug auf den Controller beibehalten wird.

Die Implementierung des Zooms erfolgt ebenfalls über die Verwendung einer Kindbeziehung. Hierbei orientiert sich das Objekt, das die Position des einen Controllers übernimmt, in die Richtung des anderen Controllers. Der Skalierungsfaktor wird nicht mehr auf das gegriffene Objekt, sondern auf das Controllerobjekt angewendet (vgl. Formel 5.2). Der Abstand des Controllers zum gegriffenen Objekt skaliert dadurch mit und die Ausrichtung zum anderen Controller bleibt erhalten. Beim Greifen des Objekts mit beiden Controllern bleiben beide Controller somit an dem Objekt haften.

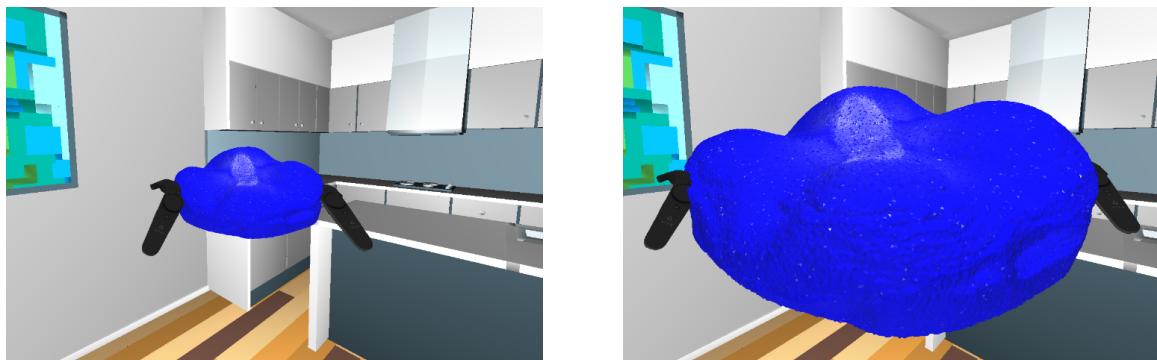


Abbildung 5.5.: Start und Ende des Zoomvorgangs (Eigene Screenshots)

6. Diskussion

Die umgesetzte Anwendung wurde zum Abschluss auf die Geschwindigkeit getestet. Die Tests wurden auf zwei verschiedenen Systemen durchgeführt. Das System, das im Folgenden als „i7“ bezeichnet wird, besitzt einen Intel Core i7-6700K mit einer Taktrate von 4,00 GHz, 16 GB RAM, eine Nvidia GTX 1070 und Microsoft Windows 10 Pro als Betriebssystem. Das andere System, das hier als „Xeon“ bezeichnet wird, enthält eine Intel Xeon CPU E3-1270v3 mit einer Taktrate von 3,50 GHz, 8 GB RAM, eine AMD Radeon R9 390 und Microsoft Windows 7 Professional als Betriebssystem.

Die in den folgenden Diagrammen angegebene Anzahl von Zellen bezieht sich auf die Testdateien, die zur Messung verwendet wurden. Zu der Anzahl der Primitiven zählen die Polygone und Punkte, die von Vizard gerendert werden.

Beim Laden eines Modells werden zwei Schritte durchgeführt, die potentiell mehr Zeit beanspruchen können. Zum einen das Laden der Simulationsergebnisse über ParaView (vgl. Abbildung 6.1) und zum anderen die Umwandlung der Daten in das 3D-Modell (vgl. Abbildung 6.2).

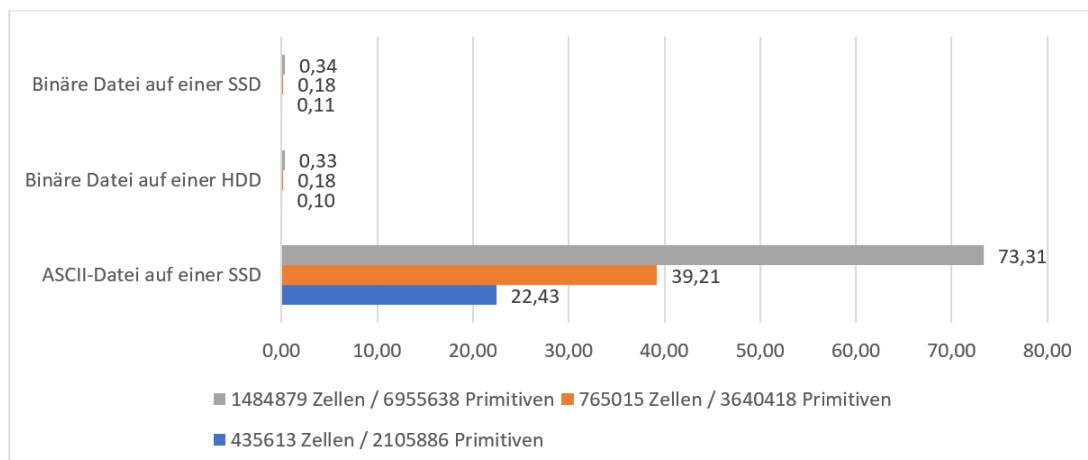


Abbildung 6.1.: Dauer zum Laden einer Datei in Sekunden (Durchschnitt von 3 Messungen)

Die benötigte Zeit zum Laden einer Datei ist sehr gering, sofern die Datei im binären VTK-Format vorhanden ist. Ob die Datei auf einer SSD oder HDD abgespeichert ist, hat keine Auswirkung. Dateien im ASCII-Format führen hingegen zu deutlich höheren Ladezeiten. Durch die einmalige Umwandlung von ASCII-Dateien ins binäre Format über ParaView kann die Ladezeit bei mehrfacher Betrachtung verkürzt werden.

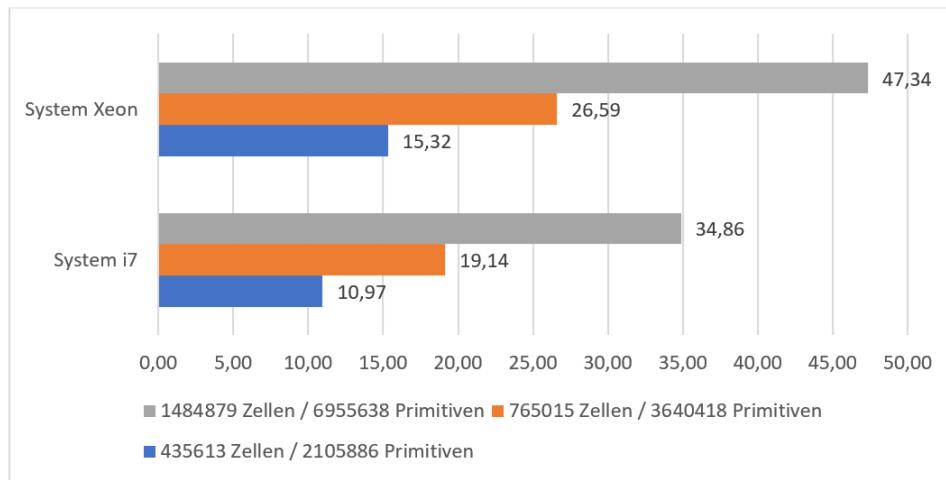


Abbildung 6.2.: Dauer der Erstellung des Modells in Sekunden (Durchschnitt von 3 Messungen)

Die Umwandlung der Simulationsergebnisse in 3D-Modelle ist von der Rechenleistung des Systems abhängig. Hier werden über ParaView Filter auf die Daten angewendet und in Schleifen über die Polygone iteriert. Der Speicherort der Datei hat hierbei keinen Einfluss.

Neben den Ladezeiten wurde auch die Bildrate gemessen. Die Messung wurde bei normaler Benutzung und mit aktiviertem Auslesen von Messdaten durchgeführt (vgl. D.2. Auslesen einzelner Werte).

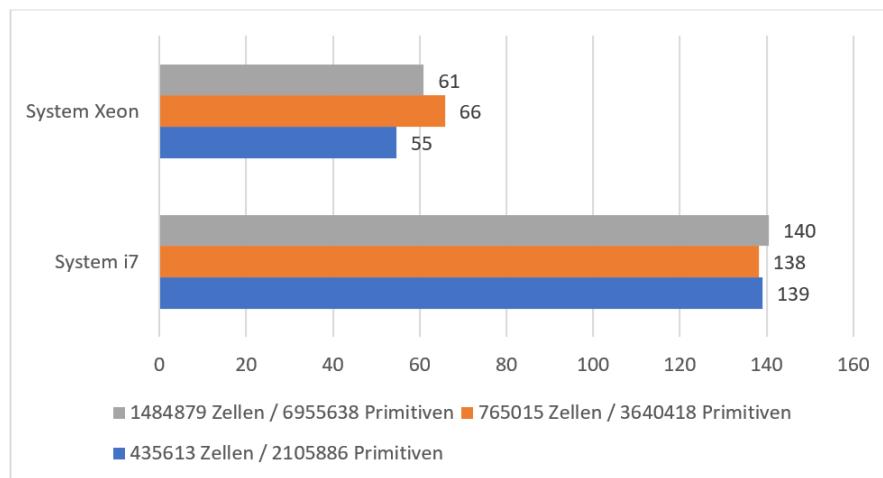


Abbildung 6.3.: Bilder pro Sekunde bei normaler Benutzung (Durchschnitt von 3 Messungen)

Die Ergebnisse beinhalten während der normalen Benutzung keine Veränderungen in Abhängigkeit von der Anzahl der vorhanden Zellen (vgl. Abbildung 6.3). Ein solches Ergebnis könnte durch eine Begrenzung der Bildrate ausgelöst werden, wodurch kein

Rückschluss auf die maximale Leistung möglich ist.

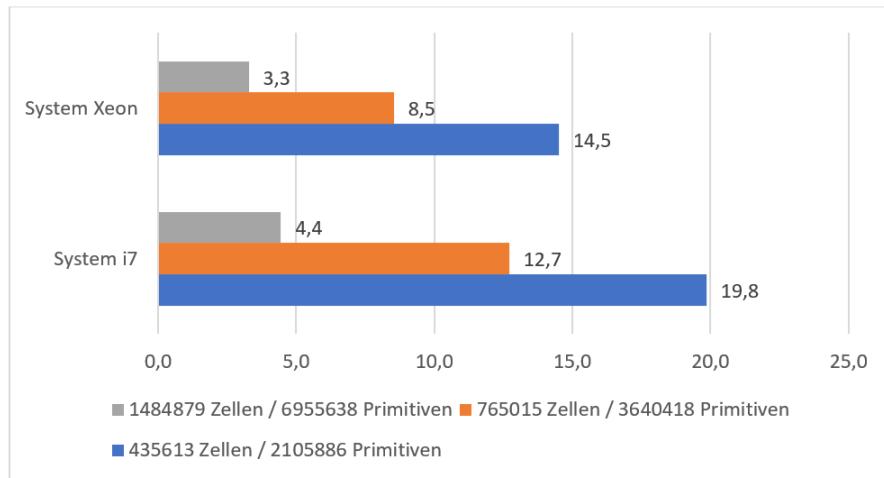


Abbildung 6.4.: Bilder pro Sekunde bei aktivierter Messung (Durchschnitt von 3 Messungen)

Bei der aktivierte Messung von Simulationsdaten sinkt die Bildrate deutlich unter die Begrenzung (vgl. Abbildung 6.4). Ein deutlicher Unterschied ist abhängig von der Anzahl vorhandener Zellen zu erkennen. Die maximal erreichten 19,8 Bilder pro Sekunde liegen deutlich unter den benötigten 90 Bildern, die für eine angenehme Benutzung der Software von Nöten ist [23, Abschnitt „VR sickness“][24, Abschnitt „Technische Daten VIVE“].

7. Schlussfolgerung

Durch die gelungene Kombination der Programme ParaView und Vizard konnte eine Anwendung erstellt werden, die einen genauen Einblick in die Struktur eines Zahns ermöglicht. Dabei konnte ohne die gezielte Optimierung des Programmcodes eine Bildrate erreicht werden, die für die Darstellung in der virtuellen Realität geeignet ist. Lediglich die Messfunktion von Simulationswerten führt zu einer niedrigen Bildrate. Da die Funktion jedoch separat aktiviert werden muss und meist nur kurz verwendet wird, ist dieses Problem von geringer Bedeutung.

Bereits in den ersten Prototypen wirkte die Darstellung der Simulationsergebnisse sehr plastisch. Durch die Darstellung der Simulationsergebnisse als Punktwolke können die Werte im Inneren des Zahns besser betrachtet werden. Die räumliche Wirkung der virtuellen Realität hilft dabei, die Verteilung der Spannungen innerhalb des Zahns besser nachvollziehen zu können. Die intuitive Implementierung des Zooms führte bei Nutzertests zu erhöhtem Interesse an detaillierteren Stellen. Die Bedienung der Querschnittsfunktion führte jedoch teils zu Verwirrung. So setzten manche Nutzer versehentlich die Querschnittebene so, dass sämtliche Punkte verschwanden und das Modell somit nicht mehr sichtbar war. Weitere Funktionen konnten implementiert werden, die dem Nutzer einen besseren Einblick in die Simulationsergebnisse ermöglichen (vgl. D. Weitere umgesetzte Funktionen).

Die Verwendung der ParaView-Bibliotheken zum Auslesen der Daten wirkt sich positiv auf die Erweiterbarkeit der Anwendung aus. Komplexe Aufgaben wie das Setzen eines Querschnitts konnten relativ einfach implementiert werden. Hierfür mussten die Interaktionsmöglichkeiten mit Vizard geschaffen werden und lediglich ein zusätzlicher Filter für ParaView aufgerufen werden. Das Modell konnte deshalb mit geringem Aufwand angepasst werden.

Die Entwicklung mit Vizard gestaltete sich durch Python als verwendete Programmiersprache sehr angenehm. Durch den Interpreter konnte die Funktionsweise von neuen Bibliotheken schnell getestet werden. Die Integration von ParaView konnte durch die angebotene Schnittstelle ebenfalls gut durchgeführt werden. Die Kompatibilität von Vizard zu vielen anderen VR-Geräten wird auch bei weiteren Entwicklungen hilfreich sein. Die von Vizard bereitgestellten Interaktionsmöglichkeiten mit Objekten in der virtuellen Realität unterliegen jedoch im Vergleich mit anderen Grafik-Engines. Vizconnect bietet zwar einige hilfreiche Tools an, andererseits sind deren Konfigurationsmöglichkeiten relativ limitiert. Die teils unvollständige Dokumentation erschwert ebenfalls die Konfiguration.

8. Ausblick auf weitere Entwicklungen

Das im Rahmen dieser Arbeit erstellte Programm besitzt Potential für zusätzliche Entwicklungen. Sowohl der Umfang als auch Details an den aktuellen Funktionen können verbessert werden.

So sollte innerhalb des Programms dem Benutzer beigebracht werden, wie er mit der virtuellen Welt interagieren kann. Dies könnte darüber erfolgen, dass kleine Beschreibungen an den Tasten der Controller angezeigt werden, die deren Funktionalität erklären.

Mit steigender Funktionalität muss auch die Erreichbarkeit der Funktionen angepasst werden. Aktuell können alle wichtigen Funktionen über Tasten aufgerufen werden. Weitere Funktionen könnten in einem Menü angezeigt werden, das z.B. in einem festen Abstand zu einem Controller angezeigt wird und somit ähnlich wie die Mischpalette eines Künstlers wirkt.

Der interaktive Wechsel zwischen den verschiedenen Ergebniswerten könnte helfen ein noch besseres Verständnis für die Simulationsergebnisse zu erhalten. Aktuell werden nur die Vergleichsspannungen dargestellt. Über Änderungen in der Konfigurationsdatei kann das Modell auch anhand des Materials eingefärbt werden. Darstellungen für Verschiebungs- und Kraftvektoren und Spannungstensoren müssen noch ergänzt werden. Durch die 9 Komponenten von Spannungstensoren gestaltet sich die Darstellung schwierig. Hierbei könnte auf die Darstellung der Eigenvektoren zurückgegriffen werden.

Der Farbverlauf der Ergebniswerte ist aktuell fest auf blau, weiß und rot eingestellt. Eine granulare Definition über die Konfigurationsdatei könnte helfen wichtige Spannungsbereiche zu erkennen. So könnte der Farbwechsel deutlicher hervorheben, bei welcher Spannung ein Material bricht.

Ein Hinweis auf die Punkte mit der höchsten Vergleichsspannung könnte dem Nutzer ebenfalls potentielle Bruchstellen zu entdecken. Die Darstellung könnte über Pfeile erfolgen, die auf entsprechenden Stellen zeigen. Sollten benachbarte Zellen die höchsten Vergleichsspannungen besitzen, so sollte dies erkannt und nur ein Pfeil benutzt werden.

Bei der Suche nach der optimalen Restauration werden mehrere Simulationen durchgeführt. Um diese besser vergleichen zu können, wäre eine Unterstützung zum gleichzeitigen Betrachten mehrerer Simulationen von Vorteil. Hierfür müssten sowohl die Interaktionsmöglichkeiten als auch die Einbindung von ParaView angepasst werden. So müsste entschieden werden, ob z.B. Querschnitte, das Ausblenden von Materialien und das Wechseln zwischen den Darstellungsmodi auf alle oder nur einzelne Simulationen angewendet werden sollen.

Die Integration weiterer Filter von ParaView könnte die Darstellung der Daten weiter verbessern. So könnte der Nutzer selbstständig einen Schwellenwert festlegen, der alle

niedrigen und somit uninteressanten Vergleichsspannungen herausfiltert.

Die vtk-Dateien liegen nach dem Simulationsprozess im ASCII-Format vor. Ein Python-Skript, das diesen Vorgang automatisiert, könnte für die Anwender hilfreich sein, da damit die Ladezeiten deutlich reduziert werden.

Anhang

A. Details zur Zahnerhaltung

Die in diesem Kapitel beschriebenen Informationen wurden - soweit nicht anders vermerkt - bei einem Besuch der Poliklinik für Zahnerhaltung und Parodontologie der Ludwig-Maximilians-Universität in München in Erfahrung gebracht.

Die Begrüßung und einleitende Worte erfolgten von Prof. Dr. med. dent. Karl-Heinz Kunzelmann. Eine Führung durch die Laborräume inklusive der Erklärung der Struktur eines Zahns und seiner Restauration sowie der Darstellung des Scavorgangs mit Hilfe des Mikro-CT erfolgte durch den Studenten der Zahnmedizin Robert Bednarek. Zum Abschluss erfolgte die Besprechung bezüglich der Anforderungen an die Software, die zur Verbesserung der Forschung verwendet werden soll. Teilnehmer des Besuchs waren Prof. Dr. Peter Rösch, Kommilitone Dennis Rockstein und Andreas Wundlechner.

A.1. Allgemeine Zahnmedizin

Das Gebiss des Menschen besteht aus mehreren Bestandteilen. Im Ober- und Unterkiefer sind die Zähne verankert. Wird ein Zahn betrachtet, so wird der Zahn an der gleichen Stelle im anderen Kiefer **Antagonist** genannt. Die Zähne werden grob in zwei verschiedene Arten unterteilt:

Schneidezähne: Sie befinden sich im vorderen Bereich des Kiefers. Sie dienen zur Portionierung der Nahrung beim Abbeißen.

Backenzähne: Sie befinden sich im hinteren Bereich des Kiefers. Sie zermahlen die Nahrung, damit diese im Magen besser verdaut werden kann.

Die Zähne wiederum bestehen, was ihre Stabilität angeht, aus 3 verschiedenen Komponenten.

Zahnmark: Innerer Teil des Zahns. Das Zahnmark enthält sowohl die Nerven, als auch die Gefäße zur Versorgung des Zahns mit Nährstoffen.

Zahnbein: Das Zahnbein ist der größte Bestandteil eines Zahns. Es ist relativ weich und umgibt das Zahnmark. Das Zahnbein stabilisiert die Struktur des Zahns. Hierfür kann es auch Kräfte aufnehmen und verteilen.

Zahnschmelz: Der Zahnschmelz ist eine dünne, harte Schicht um das Zahnbein herum. Der Zahnschmelz ist das härteste Material im menschlichen Körper. Diese Härte ist von Nöten, damit der Zahn trotz hohen Kräften auf Dauer nicht abgerieben wird.

A.2. Restaurationen

Bei Restaurationen werden inzwischen meist Keramik oder Komposite verwendet. Früher wurden oft auch Goldlegierungen verwendet. Durch die unterschiedlichen Eigenschaften der Materialien unter Last, eignen sie sich je nach Form der Restauration und Position des betroffenen Zahns besser oder schlechter.

Keramik ist ein sehr hartes Material. Es ist härter als der Zahnschmelz. Dadurch ist eine Keramikfüllung sehr haltbar. Die auf die Füllung wirkenden Kräfte werden jedoch im Vergleich zum Zahnschmelz verstärkt auf das Zahnbein und die Wurzel weitergeleitet. Sollten die Kräfte zu groß für die Wurzel werden, so kann sich der Zahn aus dem Gebiss lösen.

Keramik ist darüber hinaus auch spröde. Wird die Keramik zu hohen Kräften ausgesetzt, so können sich irreparable Risse bilden. Bricht die Füllung, so muss sie komplett ausgetauscht werden.

Goldlegierungen sind im Vergleich relativ weiche Materialien. Auf die Füllung wirkende Kräfte werden verstärkt absorbiert. Auf das Zahnbein und die Wurzel wirken somit geringere Kräfte als bei einem gesunden Zahn. Im Vergleich zu anderen Materialien verformen sich Goldlegierungen leichter. Dadurch wird der Antagonist geschont. Durch die Verformung bricht die Füllung nicht und hält somit sehr lang. Da die Goldlegierungen wegen ihrer Farbe deutlich auffallen, werden sie vor allem für die hinteren Zähne verwendet.

Komposit besteht aus Kunststoff, das mit Füllmaterialien wie Glas, Keramik oder Quarz vermischt wurde [25]. Es besitzt einen Härtegrad zwischen Keramik und Goldlegierungen. Es schont somit den Antagonisten mehr als Keramik. Da es jedoch ebenfalls spröde wie Keramik ist, bricht es schneller.

A.3. Simulationsprozess

Damit systematisch nach besseren Restaurationen geforscht werden kann, wurde ein Simulationsprozess benötigt, der realistische Ergebnisse liefert. Zu Beginn des Prozesses muss der Zahn gescannt werden. Das Scanco Micro CT 40 erstellt hierfür die Bilder, die anschließend mit ITK, Fiji und itk-SNAP in die verschiedenen Materialien segmentiert werden. Anhand der segmentierten Materialien wird mit Hilfe von octave und iso2mesh ein Netz aus Tetraedern erstellt (vgl. Abbildung A.1). Die entstehende Struktur und die Materialeigenschaften werden für die Finite Elemente Analyse des Programms Z88OS verwendet. Hierbei werden anhand einer Kraft die Spannungen und Verschiebungen berechnet, die im Inneren des Zahns herrschen. Die Analyse der Ergebnisse wird mit ParaView und scipy vorgenommen. Sämtliche verwendeten Programme sind Open Source. Zur Steuerung der Programme und zur Umwandlung der Dateitypen werden Python-Programme von Prof. Dr. Peter Rösch verwendet. [2][26, Seite 86 - 88]

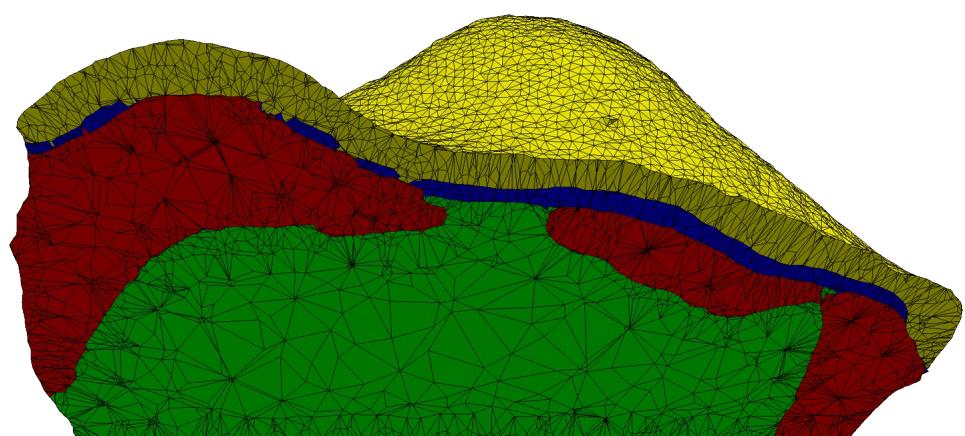


Abbildung A.1.: Visualisierung der berechneten Tetraeder [2]

B. Bedienungsanleitung

B.1. Installation

Um den Quellcode auszuführen, werden folgende Programme benötigt:

- Vizard: Version 5.8
- ParaView: Version 5.4.1
- SteamVR: Version 1518226924 vom 10. Februar 2018
- psutil: Version 5.4.3
- numpy: Version 1.13.3

Die Installation von psutil und numpy erfolgt über den Paketmanager von Vizard. Damit das Programm die Installation von ParaView findet, muss das Verzeichnis mit den ParaView Bibliotheken in der Datei config.py hinterlegt werden. Um das Programm zu starten, muss die Datei ToothVR.py mit Vizard ausgeführt werden.

B.2. Tastenbelegung

Im Quellcode liegen Steuerungskonfigurationen für die Benutzung mit Tastatur und Joystick (getestet mit Microsoft XBox 360 und XBox One Controllern) oder mit SteamVR-Headsets (getestet mit HTC Vive). Durch das Ausführen der Konfigurationsdateien (vizconnect_config_controller.py / vizconnect_config_steamvr.py) kann deren Konfiguration angepasst werden. Es wird empfohlen das Programm mit der HTC Vive zu bedienen.



Abbildung B.1.: Zuordnung der Namen zu den Tasten [27]

Standardlayout bei der Benutzung von SteamVR:

- Neue Datei laden: Tastatur, N
- Umgebung wechseln: Tastatur, R
- Verschieben des Zahns: Trigger eines beliebigen Controllers betätigen.
- Größe des Zahns anpassen: Trigger beider Controller betätigen.

Beim Greifen mit beiden Controllern kann die Größe angepasst werden. Hierbei werden die Stellen, an denen sich die Controller befinden, festgehalten. Der Rest des Zahns passt sich entsprechend an. Das Verhalten ist ähnlich dem Zoom in Fotos und Webseiten auf Smartphones und Tablets.

- Querschnitt erstellen: Beliebige Grip-Taste

Durch die erste Betätigung erscheint eine Ebene, die wie der Zahn verschoben werden kann. Durch eine erneute Betätigung wird der Schnitt anhand der Ebene durchgeführt.

- Darstellung der Messwerte durchschalten: Linker Controller, Menüknopf
Es werden abwechselnd die Oberfläche und die Punktwolke, nur die Punktwolke oder nur die Oberfläche dargestellt.

- Pixelgröße der Messpunkte erhöhen: Linker Controller, Touchpad auf der oberen Hälfte klicken.

- Pixelgröße der Messpunkte verkleinern: Linker Controller, Touchpad auf der unteren Hälfte klicken.

- Messung der Simulationsergebnisse aktivieren/deaktivieren: Rechter Controller, Menüknopf

- Anzeigen der einzelnen Materialien aktivieren/deaktivieren: Rechter Controller Touchpad

Hierbei ist zu beachten, dass das Touchpad in gleichgroße Sektoren eingeteilt wird. Die durchnummerierten Materialien werden von der rechten Seite startend nacheinander zugewiesen (vgl. Abbildung B.2).



Abbildung B.2.: Aufteilung des Touchpads des rechten Controllers in gleich große Sektoren, je nach Anzahl der Materialien (Eigene Grafik)

C. Beschreibung des Entwicklungsprozesses

C.1. Organisation der Arbeit

Um die gestellten Aufgaben strukturiert zu erledigen, wurde die Organisation an SCRUM orientiert. SCRUM ist ein Verfahren, um die gemeinsame Arbeit an einem Projekt in Teams zu organisieren. Es bietet viele Hilfsmittel, um die Kommunikation zu fördern und dadurch die Produktqualität und Entwicklungsgeschwindigkeit zu erhöhen. Für diese Arbeit wurden jedoch nur Methoden zur Strukturierung der verbleibenden Aufgaben verwendet. So wurden User-Storys erstellt, die einen Titel, Beschreibung, Priorität, Komplexitätsschätzung und Akzeptanzkriterien enthalten. Die Storys mit der höchsten Priorität wurden in Sprints umgesetzt. Die ausgewählten Storys wurden in Tasks aufgeteilt, um einzelne Arbeitsschritte zu definieren. Schließlich wurden die Storys und Tasks an das Taskboard gehängt. Während des Sprints durchliefen die Tasks die Spalten „Ready“, „In Arbeit“, „Review / Dokumentation“ und „Done“ (vgl. Abbildung C.1). Täglich wurden die Positionen der Tasks und das Burn-Down-Chart aktualisiert.

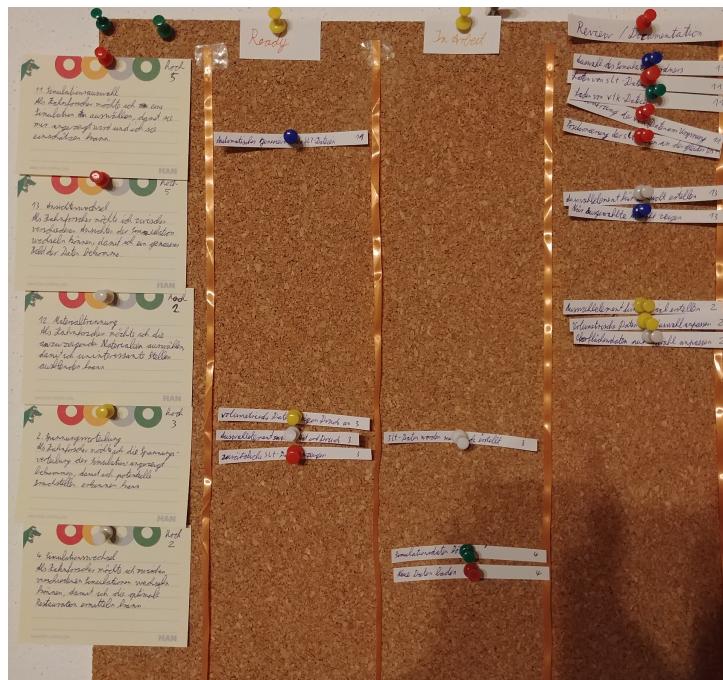


Abbildung C.1.: Taskboard zur Organisation der Aufgaben (Eigenes Foto)

Durch die User-Storys und das Taskboard konnte eine gute Übersicht erhalten werden, welche Aufgaben als nächstes zu erledigen sind. Das Umsetzen der Aufgaben wurde durch die Strukturierung zu Beginn der Sprints zielgerichtet. Durch das Verwenden einer Burn-Down-Chart konnte besser nachvollzogen werden, wie weit von der Zielsetzung abgewichen wurde. Die Verwendung der SCRUM-Methoden beschleunigte die Entwicklung.

C.2. Entwicklungsumgebung

Für die Entwicklung von Software werden verschiedene Programme benötigt. Neben einer Entwicklungsumgebung, die das Erstellen von Programmcode unterstützt, werden auch Programme zur Bearbeitung von Mediendateien, wie Bilder und 3D-Modelle, benötigt.

C.2.1. Vizard

Das von WorldViz entwickelte Vizard wurde für die größten Teile der Entwicklung verwendet. Vizard bietet Unterstützung für viele Anwendungsbereiche. So besteht Kompatibilität zu vielen verschiedenen VR-Ausstattungen [14]. Die Konfiguration der Steuerung kann über Vizconnect erfolgen. Dadurch kann die Steuerung unabhängig von der vorhandenen Hardware entwickelt werden.

Die Grafik-Engine bietet mit verschiedenen Beleuchtungsarten die Möglichkeit dynamische und realistische Umgebungen zu erstellen. Die Verwendung der eingebauten Physiksimulation kann ebenfalls den Realismus erhöhen.

Vizard enthält auch einen Inspector, der es erlaubt Objekte in Szenen zu verschieben. Hier können auch Lichtquellen hinzugefügt und konfiguriert werden.

Gesteuert werden die einzelnen Komponenten über die Programmiersprache Python. Die interpretierte Sprache bietet Unterstützung zur funktionellen als auch objektorientierten Programmierung. Durch die leicht verständliche Syntax ist Python für Einsteiger gut geeignet.

C.2.2. ParaView

Das von Kitware entwickelte Open Source Programm zur Datenanalyse und -visualisierung, wird in dieser Arbeit neben dem Aufruf der Bibliotheken auch zur Entwicklung von Programmcode verwendet.

So werden für das Anwenden von Filtern entsprechende Einstellungen benötigt. Diese können über die „Trace“-Funktion ausgelesen werden. Hierfür wird die Funktion aktiviert, der Filter über die ParaView Oberfläche angewendet und die Funktion wieder gestoppt. Als Ergebnis werden sämtliche Aktionen als Python Code ausgegeben.



Abbildung C.2.: Logo von Vizard [28]



Abbildung C.3.: Logo von ParaView [6]

ParaView wird auch zur Konvertierung der vtk-Daten verwendet. Diese werden vom ASCII in das binäre Format umgewandelt. Die Verwendung von binären Dateien führt zu einer Verkürzung der Ladezeiten.

C.2.3. Blender

Blender ist ein kostenloses Open Source Werkzeug zum Erstellen und Bearbeiten von 3D-Modellen. Es beinhaltet viele verschiedene Funktionen, wie z.B. das Bearbeiten von Texturen, die Berechnung von Beleuchtung und die Erstellung von Animationen. In dieser Arbeit wird Blender jedoch nur zum Verschieben und Löschen von Objekten verwendet.



Abbildung C.4.: Logo von Blender [29]

C.2.4. Google Blocks

Google Blocks ist eine VR-Anwendung, die zur Erstellung und Bearbeitung von 3D Modellen verwendet wird. Durch die intuitive Verwendung der virtuellen Realität ist es selbst für einen Laien möglich, einfache Objekte zu gestalten. Die Integration des Dienstes Google Poly ermöglicht den Zugriff auf die Kreationen anderer Nutzer. Diese können betrachtet und an die eigenen Vorstellungen angepasst werden.



Abbildung C.5.: Logo von Google Blocks [19]

C.2.5. Gimp

Gimp ist ein kostenloses Open Source Softwarepaket zur Bearbeitung von Bildern. Der Funktionsumfang umfasst unter anderem die Auswahl und Maskierung von Bereichen, Verwendung von Ebenen für Vorder- und Hintergründe und Anpassung der Farben. In dieser Arbeit wurden Texturen für Objekte erstellt und angepasst.



Abbildung C.6.: Logo von Gimp [30]

D. Weitere umgesetzte Funktionen

Neben den im Hauptteil besprochenen Funktionen wurden einige Features implementiert, die den Wert der Software für den Nutzer erhöhen. Diese sollen hier etwas genauer betrachtet werden.

D.1. Setzen von Querschnitten

Über das Betätigen einer Grip-Taste an der Seite der Vive-Controller wird eine Ebene eingeblendet, die zum Setzen eines Querschnitts verwendet wird. Die Ebene kann wie das Zahnmodell gegriffen und verschoben werden. Über erneutes Betätigen der Grip-Taste wird die Ebene wieder ausgeblendet und der Querschnitt wird durchgeführt. Hierfür wird der Querschnitt in ParaView berechnet und sämtliche Vizard-Objekte des Zahns werden neu generiert. Dieser Prozess kann beliebig wiederholt werden. Dabei ist zu beachten, dass der Querschnitt immer auf die Originaldaten angewendet wird. Das Setzen eines neuen Schnitts lässt den vorangegangenen verschwinden, damit ungünstig gesetzte Schnitte schnell behoben werden können.

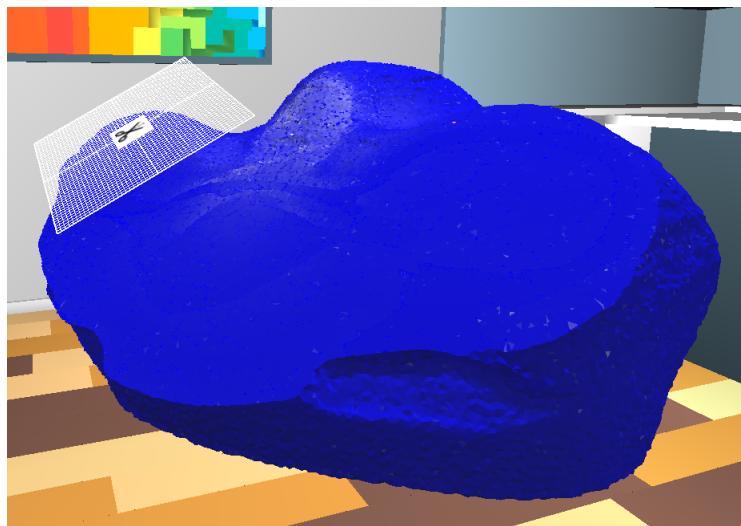


Abbildung D.1.: Durchgeführter Querschnitt und Querschnittsebene (Eigener Screen- shot)

Die Querschnittsfunktion konnte durch die Integration von ParaView einfach entwickelt werden. Wegen der relativ langen Berechnungszeit kann das Setzen des Querschnitts

nicht interaktiv durchgeführt werden. Der Nutzer sieht das Ergebnis dadurch erst nach dem Bestätigen des Schnitts.

Die Integration der Funktion verbessert den Einblick in die Simulationsergebnisse. Vor allem die Punktfolkenansicht profitiert von Querschnitten, da irrelevante Bereiche ausgeblendet werden können und somit im Hintergrund nicht mehr sichtbar sind.

Bei Nutzertests konnte festgestellt werden, dass diese Funktion den Anwender auch verwirren kann. So setzten manche Nutzer den Querschnitt so, dass das komplette Modell weggeschnitten wurde und somit verschwand. Andere Nutzer fanden es nicht intuitiv, dass der vorangegangene Schnitt beim Setzen eines neuen verschwindet.

Um diese Funktion zu verbessern, sollte vor einem Schnitt überprüft werden, ob überhaupt etwas vom Modell vorhanden bleibt. Ebenfalls könnten Schnitte auf bereits geschnittene Objekte angewendet werden. Hierbei könnte eine zusätzliche Funktion zum Zurücksetzen des Modells auf seinen Ausgangszustand Abhilfe schaffen.

D.2. Auslesen einzelner Werte

Die Visualisierung der Werte über das Einfärben des Zahns lässt nur Rückschlüsse über die Verteilung zu. Ob ein Material bricht kann über diese relativen Werte nicht erkannt werden. Die Farben orientieren sich schließlich an dem kleinsten und größten verfügbaren Wert und nicht an den Materialeigenschaften. Das Auslesen konkreter Werte ermöglicht jedoch den Vergleich mit Materialeigenschaften, um das Versagen der Restauration zu erkennen.

Über den Menüknopf auf dem rechten Controller kann das Auslesen von einzelnen Simulationswerten aktiviert und deaktiviert werden. In kurzem Abstand vor dem Controller erscheint eine grüne Sphäre, an der die Messung durchgeführt wird. Die Messung erfolgt über die ParaView-Funktion „ProbeLocation“ (vgl. Listing D.1). Der gemessene Wert wird etwas oberhalb als Zahlwert ausgegeben. Je nach Einstellung in der Konfigurationsdatei kann die Vergleichsspannung oder die Materialnummer ausgelesen werden. Es werden direkt die ausgelesenen Werte ohne Umrechnung in andere Einheiten angezeigt.

```

1 def get_probe_value(self, position):
2     """
3     Returns the value on the probe location.
4     """
5     # Apply ProbeLocation filter with configuration
6     # found by ParaView Trace function
7     probeLocation = pv.ProbeLocation(Input=self.vtk_data,
8         ProbeType='Fixed Radius Point Source')
9     self._filters += [probeLocation]
10    probeLocation.ProbeType.Center = position
11
12    local = pv.servermanager.Fetch(probeLocation)
13    return local.GetPointData().GetScalars(cfg.coloring_name).GetValue(0)

```

Listing D.1: Auszug aus der Datei Simulation_Data.py ab Zeile 326

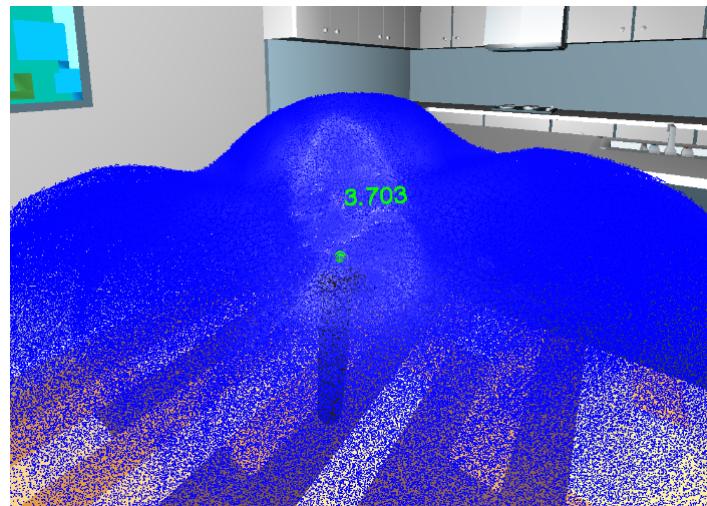


Abbildung D.2.: Auslesen der Vergleichsspannung (Eigener Screenshot)

Die Aktivierung und Deaktivierung der Funktion erfolgt wegen der hohen Berechnungsdauer. Je nach der Größe des Datensatzes kann das Auslesen eines einzelnen Wertes die Bildrate deutlich unter 30 Bilder pro Sekunde herabsenken. Um ein gutes Erlebnis zu garantieren und Unwohlsein zu verhindern, sollte eine konstante Bildrate von 90 Bildern pro Sekunde erreicht werden [23, Abschnitt „VR sickness“][24, Abschnitt „Technische Daten VIVE“]. Wenn die Funktion nicht benötigt und somit deaktiviert wird, kann eine deutlich höhere Bildrate erzielt werden. Um die Bildrate während der Verwendung zu verbessern, könnte die Funktion asynchron in einem anderen Thread ausgeführt werden. Dadurch wäre die Berechnung eines neuen Bilds von der Geschwindigkeit der Funktion unabhängig.

Durch die Bindung der Anzeige des Messwerts an den Messpunkt kann sie vom Modell überdeckt werden. Um die Messwerte weiterhin zu erkennen, könnte die Reihenfolge des Renderings angepasst werden. Die Anzeige könnte als letztes gerendert werden, wodurch sie immer sichtbar ist. Hierbei könnte die Anzeige jedoch unrealistisch wirken, da sie zwar räumlich hinter dem Objekt wahrgenommen wird, jedoch vor dem Objekt sichtbar ist. Die bessere Lösung könnte die Positionierung der Messwerte an einem anderen Ort sein. Mögliche Orte sind am anderen Controller oder an einer festen Position in der Umgebung zu finden.

D.3. Anpassung der Punktwolkenansicht

Bei der Erstellung der Punktwolkenansicht ist bei Vizard festzulegen, welche Größe die Punkte besitzen sollen. Wird die Größe zu niedrig gewählt, dann kann der Nutzer einzelne Werte nur schwierig erkennen. Werden die Punkte zu groß erstellt, dann können nur die Punkte nahe der Oberfläche erkannt werden. Der Zoomfaktor hat ebenfalls Einfluss darauf. Da die Größe der Punkte auch von dem Interessenbereich des Nutzers abhängt, wurde dem Nutzer ermöglicht die Punktgröße eigenständig zu verändern. Das erfolgt

über das Drücken der oberen bzw. unteren Hälfte des Touchpads am linken Controller.

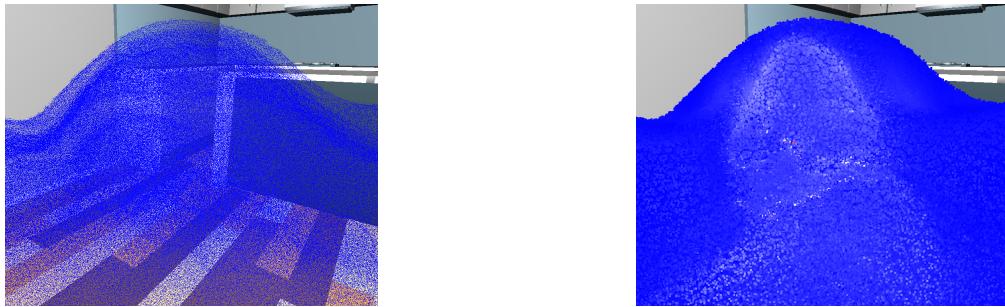


Abbildung D.3.: Variation der Punktgröße (Eigene Screenshots)

D.4. Implementierung der Umgebung

Die Umgebung bei einer VR-Anwendung hat Einfluss auf das Wohlbefinden des Nutzers. Bei zu kleinen Räumen können Nutzer z.B. Platzangst bekommen [31, Abschnitt „Rehabilitation“]. Als Grundlage wurden die Modelle „Cozy Kitchen“ von Lachlan Sleight [32] und „classroom“ von Jonathan Granskog [33] von der Google Poly Plattform verwendet. Die Kombination und Anpassung der Modelle erfolgte in Google Blocks. Hierbei wurden von „Cozy Kitchen“ unerwünschte Objekte wie ein Esstisch und Obstschalen entfernt, Wände und Decke vervollständigt und eine Lampe und ein Gemälde von „classroom“ hinzugefügt. In Blender wurden die Objekte anschließend etwas genauer positioniert und als fbx-Datei exportiert. Im Inspector von Vizard wurden schließlich Lichtquellen hinzugefügt, damit die Umgebung ausreichend beleuchtet wird.



Abbildung D.4.: Umgebung zum Betrachten der Simulationsergebnisse (Eigener Screen-
shot)

Wegen den Wänden ist die Sichtweite bei dieser Umgebung jedoch beschränkt. Die alternative Umgebung besteht lediglich aus einem Kreis als Bodenfläche, dessen Textur mit Gimp erstellt wurde, und Lichtquellen zur optimalen Ausleuchtung. Da die alternative Umgebung aus sehr simpler Geometrie besteht, benötigt diese nur sehr wenig Rechenleistung.

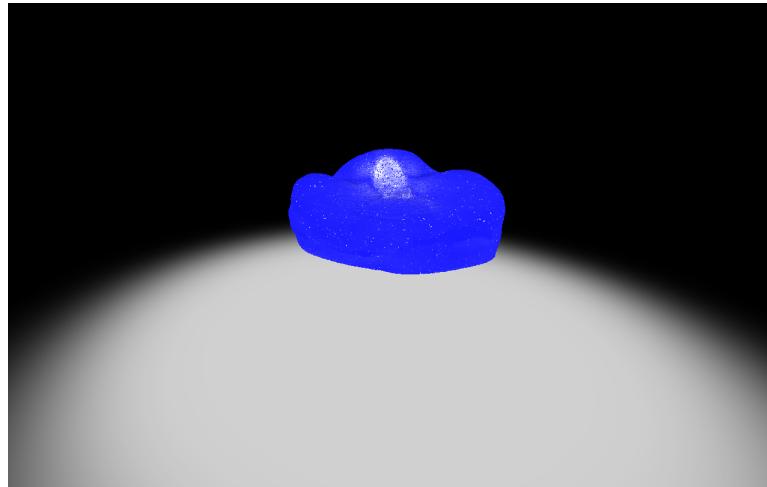


Abbildung D.5.: Alternative Umgebung (Eigener Screenshot)

Durch die Nutzertests konnte festgestellt werden, dass die alternative Umgebung, wegen der Dunkelheit und den wenigen Referenzpunkten, als unangenehm empfunden wurde. Die begrenzte Sichtweite der Standardumgebung wurde jedoch nicht als sonderlich störend wahrgenommen.

D.5. Steuerung mit Vizconnect

Um die Steuerung in Vizard zu realisieren gibt es zwei Möglichkeiten. Zum einen könnte innerhalb des Programmcodes ein Event registriert werden, das durch den Tastendruck ausgelöst wird. Zum anderen könnte die Steuerung über Vizconnect konfiguriert werden. Hierbei wird mit Hilfe des Konfigurationsassistenten festgelegt, welche Taste zu einer Aktion oder einem Event zugeordnet wird. Es können von Vizard vorkonfigurierte Aktionen, wie das Greifen von Objekten oder die Bewegung im Raum, ausgewählt werden oder eigene Events erstellt werden. Die Events können innerhalb des Programmcodes verwendet werden, um eigene Aktionen auszulösen.

In dieser Arbeit wurden beide Steuerungsmöglichkeiten verwendet. Wegen dem allgemein gehaltenen Zugriff wurden möglichst viele Funktionen über eigene Events in Vizconnect konfiguriert. Kompatibilität zu neuer VR-Hardware herzustellen, erfolgt somit größtenteils über die Erstellung einer neuen Konfigurationsdatei via Vizconnect.

Die Auswahl der Materialien ist abhängig von der verfügbaren Anzahl selbiger. Eine solch generische Funktion kann über Visconnect nicht realisiert werden. Diese Funktion musste manuell für die Steuerung mit Tastatur oder HTC Vive programmiert werden.

ANHANG D. WEITERE UMGESETZTE FUNKTIONEN

Hierbei wurde in dem Modul, das für die Steuerung zuständig ist, direkt auf die Eingaben der Geräte zugegriffen.

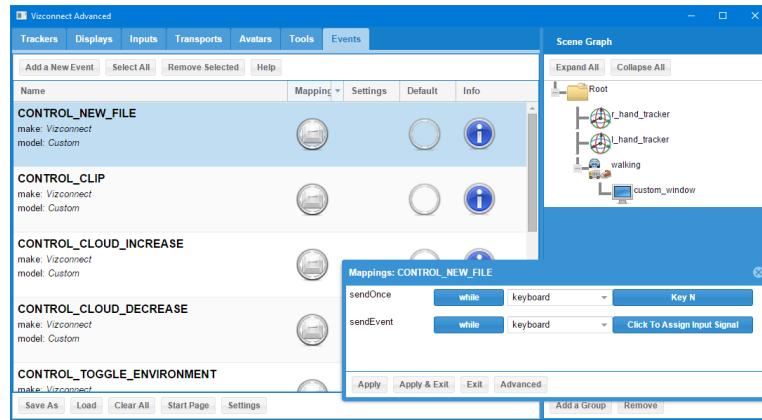


Abbildung D.6.: Konfiguration der Events über Vizconnect (Eigener Screenshot)

D.6. Automatische Steuerungserkennung

Durch die Verfügbarkeit mehrerer Konfigurationsdateien für die Steuerung wurde der Wechsel über die Hauptkonfigurationsdatei des Programms durchgeführt. Um schnell zwischen den verschiedenen Konfigurationen zu wechseln, wurde eine zusätzliche Methode implementiert, die erfasst, ob der Prozess von SteamVR aktuell läuft und entsprechend zur VR-Steuerung wechselt.

E. Begriffsverzeichnis

E.1. Grafik-Engine

Eine Grafik-Engine - oft auch nur kurz als Engine bezeichnet - ist ein Framework, das dem anwendenden Entwickler eine Vielzahl an Funktionen zur Erstellung, Darstellung und Interaktion mit einer virtuellen Welt bietet. Darunter zählen z.B. Systeme zur Beleuchtung von Objekten, Anwendung von Bildschirmeffekten und Abfragen von Benutzereingaben. Eine Engine kann dabei 2D- und/oder 3D-Grafiken darstellen.

Viele Grafik-Engines enthalten darüber hinaus auch weitere Funktionen, wie Systeme zur Simulation von Physik und zur Verwendung von Audio.

E.2. Rendering

Rendering (Verb: rendern) bezeichnet den Vorgang, in dem die Daten eines Modells oder Szene in ein Bild umgewandelt werden. Dieser Vorgang erfolgt bei interaktiven 3D-Anwendungen mehrfach pro Sekunde, um ein flüssiges Bedienungsgefühl zu erzeugen.

E.3. VR - Virtuelle Realität

Die virtuelle Realität beschreibt die Darstellung einer virtuellen Welt in einer für den Nutzer real wirkenden Form. Hierfür nutzen aktuelle Geräte eine Brille mit eingebautem Display. Durch die Abdeckung des Sichtfelds mit dem Display sieht der Nutzer lediglich die virtuelle Welt. In Kombination mit Trackingsystemen kann das angezeigte Bild an die Kopfbewegung angepasst werden. Durch die Anpassung verbessert sich die Immersion für den Nutzer deutlich.

Aktuelle VR-Geräte bieten unterschiedliche Erfahrungen. So besitzen die günstigen Smartphone Lösungen wie Google Cardboard und Samsung Gear VR lediglich 3 Freiheitsgrade in der Bewegung. Hierbei passt sich die Sicht des Nutzers lediglich an die Rotation an. Um 3 weitere Freiheitsgrade zu erhalten werden deutlich komplexere Systeme benötigt. Die Geräte Sony Playstation VR, Oculus Rift und HTC Vive messen ihre Position in Relation zu den Basisstationen. Durch die 6 Freiheitsgrade kann sich der Nutzer frei im Raum bewegen und umsehen. Das Tracking kann auch für die Controller verwendet werden, damit diese in der virtuellen Welt abgebildet werden können und intuitive Interaktionen mit den virtuellen Objekten ermöglichen.

Die räumliche Wirkung wird auch durch die stereoskopische Darstellung der Inhalte unterstützt. Das verbaute Display wird in zwei Teile aufgeteilt. Jeder Teil ist nur für

ANHANG E. BEGRIFFSVERZEICHNIS

ein Auge sichtbar. Durch eine entsprechend versetzte Position beim Rendering für das zweite Auge entsteht für den Nutzer ein dreidimensionales Bild.

Literatur

- [1] *50 Millionen Zahnfüllungen im Jahr.* 29. Apr. 2013. URL: <https://www.ovb-online.de/weltspiegel/millionen-zahnfuellungen-jahr-2878613.html> (Abgerufen am 06.02.2018).
- [2] Peter Rösch und Karl-Heinz Kunzelmann. „Finite Element Analysis of Ultrathin Occlusal Veneers“. In: ConsEuro 2015.
- [3] *Strukturanalyse.* URL: <http://www.ansys.com/de-de/products/structures> (Abgerufen am 02.03.2018).
- [4] *Abaqus/Standard.* URL: <https://www.3ds.com/de/produkte-und-services/simulia/produkte/abaqus/abaqusstandard/> (Abgerufen am 02.03.2018).
- [7] Ken Martin u. a. *Taking ParaView into Virtual Reality.* 22. Sep. 2016. URL: <https://blog.kitware.com/taking-paraview-into-virtual-reality/> (Abgerufen am 15.02.2018).
- [10] *Virtual Reality Headset Shipments Top 1 Million for the First Time.* 27. Nov. 2017. URL: <https://www.canalys.com/newsroom/media-alert-virtual-reality-headset-shipments-top-1-million-first-time> (Abgerufen am 05.03.2018).
- [11] Benjamin Schäfer. *Oculus Rift vs. HTC Vive: Der große VR-Brillen-Vergleich 2018.* 7. Sep. 2017. URL: <https://vr-world.com/oculus-rift-vs-htc-vive-vr-brillen-vergleich-2017/> (Abgerufen am 05.03.2018).
- [12] *Ein Build, Bereitstellung überall.* URL: <https://unity3d.com/de/unity/features/multiplatform> (Abgerufen am 18.02.2018).
- [13] *Unreal Engine for AR, VR & MR.* URL: <https://www.unrealengine.com/en-US/vr> (Abgerufen am 18.02.2018).
- [14] *Native Device Support.* URL: http://docs.worldviz.com/vizard/Native_support.htm (Abgerufen am 18.02.2018).
- [15] *About WorldViz.* URL: <http://www.worldviz.com/about-worldviz-virtual-reality-software/> (Abgerufen am 05.03.2018).
- [16] Pearu Peterson. *PyVTK.* 29. Mai 2017. URL: <https://github.com/pearu/pyvtk> (Abgerufen am 19.02.2018).
- [17] *ParaView/Python Scripting.* URL: https://www.paraview.org/Wiki/ParaView/Python_Scripting (Abgerufen am 26.03.2018).
- [18] Jeff. *Vizconnect Grabber Grab Area.* 18. Dez. 2015. URL: <http://forum.worldviz.com/showthread.php?t=5564> (Abgerufen am 20.02.2018).

- [19] *Introducing Blocks*. URL: <https://vr.google.com/blocks/> (Abgerufen am 20.02.2018).
- [20] *Painting from a New Perspective*. URL: <https://www.tiltbrush.com/> (Abgerufen am 21.02.2018).
- [21] *Shape Your Creativity*. URL: <https://www.masterpiecevr.com/> (Abgerufen am 21.02.2018).
- [23] Susan Michalak. *Guidelines for Immersive Virtual Reality Experiences*. 3. Juli 2017. URL: <https://software.intel.com/en-us/articles/guidelines-for-immersive-virtual-reality-experiences> (Abgerufen am 08.03.2018).
- [24] *VIVE VR-SYSTEM*. URL: <https://www.vive.com/de/product/> (Abgerufen am 08.03.2018).
- [25] *Komposit*. URL: <https://www.zahn-lexikon.com/index.php/k/2324-komposit> (Abgerufen am 28.02.2018).
- [26] *gP Forschung*. 2015. URL: <https://www.hs-augsburg.de/Binaries/Binary8849/gp-forschung-2015.pdf> (Abgerufen am 07.02.2018).
- [31] Christoph Runde und Urs Künzler. *Virtuelle Techniken in Medizin und Medizintechnik*. 27. März 2013. URL: <https://www.digital-engineering-magazin.de/virtuelle-techniken-medizin-und-medizintechnik> (Abgerufen am 08.03.2018).
- [32] Lachlan Sleight. *Cozy Kitchen*. URL: <https://poly.google.com/view/6b0TntCJyyK> (Abgerufen am 08.03.2018).
- [33] Jonathan Granskog. *classroom*. URL: <https://poly.google.com/view/860RGZSCitx> (Abgerufen am 08.03.2018).

Bildquellen

- [2] Peter Rösch und Karl-Heinz Kunzelmann. „Finite Element Analysis of Ultrathin Occlusal Veneers“. In: ConsEuro 2015.
- [3] *Strukturanalyse*. URL: <http://www.ansys.com/de-de/products/structures> (Abgerufen am 02.03.2018).
- [4] *Abaqus/Standard*. URL: <https://www.3ds.com/de/produkte-und-services/simulia/produkte/abaqus/abaqusstandard/> (Abgerufen am 02.03.2018).
- [5] *Z88OS*. URL: <https://z88.de/z88os/> (Abgerufen am 10.02.2018).
- [6] *ParaView*. URL: <https://www.paraview.org/> (Abgerufen am 07.03.2018).
- [8] *3D Slicer*. URL: <https://www.slicer.org/> (Abgerufen am 07.03.2018).
- [9] *The Publication Database Hosted by SPL*. URL: <http://www.spl.harvard.edu/publications/gallery?entriesPerPage=10&selectedCollection=11> (Abgerufen am 16.02.2018).
- [19] *Introducing Blocks*. URL: <https://vr.google.com/blocks/> (Abgerufen am 20.02.2018).
- [22] *Use Galaxy S7 Touchscreen Gestures*. URL: <http://gadgetguideonline.com/s7/galaxy-s7-online-manual/use-galaxy-s7-touchscreen-gestures/> (Abgerufen am 20.02.2018).
- [27] *Controller*. URL: <https://www.vive.com/de/accessory/controller/> (Abgerufen am 09.03.2018).
- [28] *WorldViz Technical Support*. URL: <https://support.worldviz.com/hc/en-us> (Abgerufen am 06.03.2018).
- [29] *Blender*. URL: <https://www.blender.org/> (Abgerufen am 07.03.2018).
- [30] *GIMP - GNU Image Manipulation Program*. URL: <https://www.gimp.org/> (Abgerufen am 07.03.2018).