



---

## Praxisprojekt VoxelVR

---

von  
Daniel Gofman  
und  
Simon Porten

an der Technology, Arts and Sciences TH Köln  
Campus Gummersbach  
im Studiengang Medieninformatik (Bachelor)

Betreuer: Prof. Dr. Martin Eisemann  
Technology, Arts and Sciences TH Köln

# **Abstract**

Virtual Reality ist eine Technologie, die vor wenigen Jahren ihren erfolgreichen Umbruch erlebt hat. Starke Bestreiter wie HTC und Oculus ermöglichen, dass diese aufstrebende Technologie neue Pforten der Unterhaltungskunst eröffnet. Dieses neue Medium zeigt völlig neue Möglichkeiten sowie Grenzen und sorgt damit dafür, dass auch alteingesessene Videospielentwickler vor neuen Herausforderungen stehen und viele neuartige Regeln lernen müssen. Als Informatikstudenten einer technischen Hochschule möchten wir ebenfalls durch ein praktisches Projekt Erfahrung in diesem Gebiet sammeln. Außerdem möchten wir damit erleben wie schwierig es ist dieses neue Genre zu betreten und darin zu entwickeln.

Infolgedessen präsentiert diese Dokumentation die Umsetzung eines Virtual Reality Voxel Editors im Rahmen eines Praxisprojekts des Medieninformatik Studienganges an der TH Köln. Es wird der Arbeitsprozess in Form der Recherche, Konzeption und Umsetzung dargestellt. Alle Ergebnisse, Erkenntnisse und offene Fragen, die auf diesem Weg gesammelt wurden, werden am Ende dieses Dokuments reflektiert.

Die durch das Studium an der Technischen Hochschule erlangten Kenntnisse und Fähigkeiten zum systematischen und wissenschaftlichen Arbeiten, sowie die Initiative zur Selbstständigkeit, sollen in diesem Projekt allumfassend genutzt werden um qualitative Ergebnisse abzuliefern. Diese sollen außerdem dabei helfen die Grenzen und Möglichkeiten einer neuartigen Technologie wie die der Virtual Reality kennenzulernen.

# Inhaltsverzeichnis

<b>Abstract</b>	i
<b>Abbildungsverzeichnis</b>	v
<b>Glossar</b>	vii
<b>1 Einleitung</b>	1
1.1 Virtual Reality und Voxel Editoren . . . . .	1
1.2 Ziel der Arbeit . . . . .	2
1.3 Vorgehensweise . . . . .	3
1.4 Die Entwicklungsumgebung Unity . . . . .	4
<b>2 Vorstellung vorhandener VR-Editoren</b>	5
2.1 Masterpiece VR 2017 . . . . .	5
2.2 Blocks - Google VR . . . . .	6
2.3 SculptrVR . . . . .	7
2.4 VoxVR . . . . .	9
2.5 Abstractron Voxulo . . . . .	10
2.6 Erkenntnisse der Marktrecherche . . . . .	11
<b>3 Grundlagen in VR-Ergonomie, -User Interface und -User Experience</b>	13
3.1 Motion Sickness & Gegenmaßnahmen . . . . .	13
3.2 Paradigmen zur Platzierung von Objekten und Interfaces in virtuellen Räumen .	15
3.3 Screen und User Interface Design in VR . . . . .	18
3.3.1 Distance-Independent Millimeter als VR-Maßeinheit . . . . .	18
3.3.2 Kenngrößen zum UI-Design . . . . .	19
3.4 Usability und Experience . . . . .	19
<b>4 Konzept</b>	22

4.1	Controlleraufbau und Tastenbelegung - HTC Vive . . . . .	22
4.2	Funktionsplanung . . . . .	24
4.2.1	Cubes platzieren . . . . .	24
4.2.2	Cubes entfernen . . . . .	25
4.2.3	Cubes einfärben . . . . .	26
4.2.4	Teleportation . . . . .	27
4.2.5	Analoge Controllersteuerung . . . . .	28
4.2.6	Cubes gruppieren . . . . .	29
4.2.7	Objekte manipulieren . . . . .	31
<b>5</b>	<b>Umsetzung</b>	<b>32</b>
5.1	Cube Generierung . . . . .	32
5.2	Benchmark . . . . .	33
5.3	Funktionsoptimierung (FO) . . . . .	35
5.3.1	FO: Cubes platzieren . . . . .	36
5.3.2	FO: Cubes entfernen . . . . .	37
5.3.3	FO: Cubes einfärben . . . . .	37
5.3.4	FO: Teleportation . . . . .	39
5.3.5	FO: Analoge Controllersteuerung . . . . .	40
5.3.6	Werkzeugauswahl . . . . .	41
5.3.7	Options-Menü . . . . .	42
5.3.8	Selektionssteuerung . . . . .	43
5.3.9	Cube Skalierbarkeit . . . . .	44
5.3.10	Cube Kombination . . . . .	45
<b>6</b>	<b>Diskussion</b>	<b>46</b>
6.1	Kritische Reflexion & Erkenntnisse . . . . .	46
6.1.1	Ablauf . . . . .	46
6.1.2	Fehlende durch Programmcode selbst definierte 3D-Primitive . . . . .	47
6.1.3	Fehlende Cube-Auflösung nach Cube-Kombination . . . . .	48
6.1.4	Spät durchgeführte Benchmarks . . . . .	49
6.1.5	Zeitverlust durch spontane Implementierung . . . . .	49
6.1.6	Unterschätzter Zeitaufwand der Fehlerkorrekturen . . . . .	50

6.1.7	Umsetzungspunkte der VR-Ergonomie . . . . .	50
6.2	Offene Fragen . . . . .	51
6.2.1	Welche Selektionssteuerung führt zu besserer Usability? . . . . .	51
6.2.2	Durch welche Optimierungsmethoden kann die Performance noch weiter verbessert werden? . . . . .	52
6.3	Themenrelevanz & Ausblick . . . . .	52
<b>7</b>	<b>Fazit</b>	<b>53</b>
<b>Literaturverzeichnis</b>		<b>55</b>
<b>A</b>	<b>Eidesstattliche Erklärung</b>	<b>57</b>
<b>B</b>	<b>Anleitung</b>	<b>58</b>

# Abbildungsverzeichnis

1.1	Minecraft Voxelwelt . . . . .	2
2.1	Masterpiece VR: User Interface . . . . .	6
2.2	Blocks Google VR: User Interface . . . . .	7
2.3	Blocks Google VR: Farbpalette . . . . .	8
2.4	SculptVR - Voxel Welt mit Wasser und Himmel . . . . .	8
2.5	VoxVR - Cubes platzieren . . . . .	9
2.6	VoxVR - Cube Level Beispiel . . . . .	10
2.7	Abstractron Voxulo - Farbpalette und Laserpointer . . . . .	11
3.1	Sichtweite 94° . . . . .	14
3.2	Kombinierte Sichtweite . . . . .	14
3.3	Geneigtes Sichtfeld . . . . .	16
3.4	Tiefenwahrnehmung in Relation zur Entfernung in VR . . . . .	16
3.5	Content Zones nach Alger . . . . .	17
4.1	HTC Vive - Tastenbelegung . . . . .	23
4.2	Cubes platzieren . . . . .	25
4.3	Cubes entfernen . . . . .	26
4.4	Farbauswahl Touchpad . . . . .	27
4.5	Teleportation . . . . .	28
4.6	Beispiel Analoge Controllersteuerung . . . . .	29
4.7	Gruppierte Entitäten . . . . .	30
5.1	Benchmark Graph 1 (FPS ohne Kombination) . . . . .	33
5.2	Benchmark Graph 2 (FPS mit Kombination) . . . . .	34
5.3	Unityfehler: Zu viele Verticies . . . . .	35

5.4	Cube Vorschau . . . . .	36
5.5	Farbauswahl . . . . .	37
5.6	Dunkelwert . . . . .	38
5.7	Verlauf der Farbauswahl . . . . .	39
5.8	Teleportation - Pointer . . . . .	40
5.9	Werkzeugauswahl . . . . .	41
5.10	Options-Menü . . . . .	42
5.11	Selektionssteuerung im Vergleich . . . . .	44
5.12	Skalierungsstufen . . . . .	44

# Glossar

<b>3D-Primitive</b>	Eine gewöhnliche Grundform der Computergrafik. Dazu zählen beispielsweise Würfel, Dreiecke, Zylinder, etc.
<b>Bug</b>	Ein Programmfehler. Ein unerwartetes falsches Ausführen der Programmlogik.
<b>Bugfix</b>	Das Beheben eines Programmfehlers.
<b>Cube</b>	Ein Würfel. Das grundlegende 3D-Element des Voxel Editors.
<b>Engine</b>	Im allgemeinen Sprachgebrauch: Videospiel-Mechanik. Vorprogrammierte Eigenschaften zur Existenz einer Spielwelt. Bereitstellung von physikalischen Berechnungen wie Bewegung, Licht, etc.
<b>GameObject</b>	Ein Behälter von 3D-Entitäten (z.B. von einem Cube). Ein Cube ist immer in einem GameObject enthalten. Ein GameObject kann einen bis ca. 5000 Cubes enthalten.
<b>gemappt</b>	Anglizismus des Englischen Verbs "to map" - (einzeichnen, abbilden). Im Zusammenhang der Computergrafik: Relevante Darstellungsinformationen auf ein 3D-Objekt übertragen.
<b>Mesh-Gitter</b>	Ein Maschenweites Netz in der Spielwelt. Eine Masche dieses Netzes besteht aus der Grundeinheit (Größe) eines Cubes.
<b>Steam Greenlight</b>	Eine Distributions-Plattform, auf der Indie-Spieleentwickler ihre Spiele präsentieren. Ein Abstimmungs- und Überprüfungsprozess bestimmt, ob und wann Spiele für den öffentlichen Markt geeignet sind.
<b>togglen</b>	Anglizismus des Englischen Wortes "toggle". Zu Deutsch: umschalten oder hin- und herschalten. Beschreibt die Fähigkeit zwischen zwei oder mehreren Funktionen per Knopfdruck umzuschalten.
<b>Tool</b>	Synonym für Werkzeug
<b>Vertices</b>	Die Punkte einer 3D-Primitive. Ein Cube hat beispielsweise acht Vertices.
<b>warpen</b>	Synonym für "teleportieren"

# 1 Einleitung

## 1.1 Virtual Reality und Voxel Editoren

Der Begriff Virtual Reality (zu Deutsch: virtuelle Realität) beschreibt die Wahrnehmung der Wirklichkeit und ihrer Eigenschaften in einem virtuellen, dreidimensionalen Raum. Dabei verwendet ein Nutzer<sup>1</sup> spezielle Hardware um körperliche Bewegung digital aufzunehmen und Interaktion zu ermöglichen. Am bekanntesten ist ein Head-Mounted Display, also eine Art Brille vor den Augen, um sich in einer digitalen Umgebung umzuschauen. Controller in den Händen und ein Laufband unter den Füßen erhöhen die Immersion und ermöglichen so, neben einer digitalen Sicht, auch ein virtuelles Handeln und Laufen (vgl. [360°-Know-How, 2017](#)).

Der Markt bietet viele Anwendungen im Unterhaltungsbereich, wie z.B. Videospiele, Filme und künstlerische Möglichkeiten wie Zeichen- und Malprogramme. Aber auch das Militär, die Medizin und andere Industriebereiche finden Verwendung für die Technologie und führen mit ihrer Hilfe kritische Simulationen aus oder bilden ihre Mitarbeiter fort (vgl. [Schäfer, 2017](#)).

Das Wort Voxel ist eine Analogie aus den englischen Begriffen “Pixel”, “Volume” und “Element” und bezeichnet einen dreidimensionalen Bildpunkt mit bis zu vier Eigenschaften in einem virtuellen Raum. Das sind seine X-Koordinate, seine Y-Koordinate und im Fall des Voxels auch eine Tiefenlage, die Z-Koordinate und die Opazität, also Durchsichtigkeit (vgl. [von IDG, 1991](#)). So werden Voxel in der Regel als Würfel dargestellt und bilden einen bestimmten Stil der Computergrafik. Diese Darstellungsart findet man beispielsweise in dem Videospiel Minecraft, wie in Bild 1.1 zu sehen ist. Voxel werden häufig als künstlerisches Gestaltungsmittel genutzt (vgl. [Sachs, 2016](#)).

---

<sup>1</sup>Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Sämtliche Personenbezeichnungen gelten gleichwohl für beide Geschlechter.



ABBILDUNG 1.1: Minecraft Voxelwelt  
<https://www.planetminecraft.com/project/minecraft-cube-world-v2/>

Editoren finden besonders häufig in Videospielen ihren Nutzen, aber auch in anderen Unterhaltungsmedien wie zum Beispiel der Filmindustrie. So handelt es sich bei Editoren um Programme, die zur Erstellung bestimmter Objekte optimiert wurden, um diese möglichst effizient zu erstellen. Ein bekanntes Beispiel sind Level-Editoren für Videospiele. Diese ermöglichen es einem Nutzer eine Welt mit Hilfe von vordefinierten Objekten zu erschaffen. Dies können unter anderem 3D-Modelle in Form von Welt-Elementen wie Steine, Bäume, Gebäude, etc. sein. Der Nutzer muss diese Elemente nicht manuell erstellen, sondern kann diese importieren, bearbeiten, platzieren und manipulieren um sie am Ende in ein Videospiel oder ähnliche Anwendungen zu exportieren (vgl. [Dudenverlag, 1988](#)).

## 1.2 Ziel der Arbeit

Ziel der Arbeit ist die Erschaffung eines Virtual Reality Voxel Editors. Es soll ein digitaler Raum erschaffen werden, indem sich ein Nutzer mit Hilfe von VR-Technologie bewegen kann, um Voxelobjekte mit Hilfe von Controllern in der Hand virtuell herstellen und bearbeiten zu können. Dieser Raum wird von Grund auf selbst gestaltet und entwickelt. Dafür werden im weiteren Verlauf der Dokumentation Konzeptionsziele und Funktionen definiert, die als Orientierung während der Umsetzungsphase dienen.

Ein weiteres Ziel ist, die Domäne der VR-Technologie durch dieses Projekt kennenzulernen. Dadurch sollen auch andere Studenten, die sich zukünftig mit einem Thema aus der VR-Domäne beschäftigen, einen Vorteil ziehen und nützliche Informationen aus den Erkenntnissen dieses Projekts entnehmen können.

Das Projekt wird in der Programmiersprache C# programmiert und in der Entwicklungsumgebung Unity (Version 2017.1.0) mit den Assets 'VRTK' und 'SteamVR' entwickelt. Während der Entwicklungsphase werden umgesetzte Funktionen mit der HTC-Vive (Version von 2017) live getestet, weshalb der Fokus der Arbeit in diesem Praxisprojekt auf einer Steuerung mit der HTC Vive liegt.

### 1.3 Vorgehensweise

Dieser Abschnitt zeigt die Herangehensweise an das Thema und beschreibt, was die Leser in welchen Kapiteln finden.

Im folgenden Unterkapitel (1.4) wird zunächst die Entwicklungsumgebung Unity erklärt und warum wir uns für diese entschieden haben.

Im zweiten Kapitel *Vorstellung vorhandener VR-Editoren* werden ähnliche Anwendungen auf dem Markt betrachtet und miteinander verglichen. Dabei liegt der Fokus auf den Grundfunktionen, der User-Interface-Gestaltung und der User Experience. Aus dieser Betrachtung werden Inspiration und Ideen für das eigene Projekt gewonnen werden. Wichtige Erkenntnisse aus diesem Vergleich werden dabei schriftlich festgehalten.

Bevor mit der Entwicklung einer VR-Anwendung begonnen werden kann, müssen die wichtigsten Grundlagen recherchiert und verstanden werden. Damit befasst sich das dritte Kapitel *Grundlagen in VR-Ergonomie, -User Interface und -User Experience*.

Darauf folgt die Konzeption in Kapitel 4. Dort wird die theoretische Planung beschrieben, die zu Beginn des Projekts erstellt wurde. Unter anderem werden dort geplante Funktionen aufgelistet und die Controllerbedienung erklärt.

Wie das Konzept umgesetzt wurde und was sich während der Arbeit verändert und angepasst hat, wird in Kapitel 5 *Umsetzung* erklärt. Funktionen haben sich durch Iterationsprozesse angepasst und wurden verbessert. Diese Änderungen finden sich ebenfalls in diesem Kapitel.

Die Diskussion über das Projekt befindet sich in Kapitel 6. Dort wird der Ablauf des Praxisprojekts reflektiert und besprochen. Erkenntnisse, die während der Arbeit gesammelt wurden, werden dort ebenso aufgelistet. Außerdem kann dort nachgelesen werden, welche Fragen mit dem Ende des Projekts offen geblieben sind und wie relevant das Thema bleibt.

Zum Schluss wird in Kapitel 7 ein Fazit gezogen.

## 1.4 Die Entwicklungsumgebung Unity

Unity ist eine Entwicklungs- und Laufzeitumgebung für Videospiele und ähnliche 3D-Grafik-Anwendungen. Unity bietet wie andere Engines (Unreal, Havok, etc.) Entwicklern den Vorteil bestimmte Aufgaben wie die Berechnung von Physik und Licht, Erstellung von Animation und andere Funktionen zu übernehmen und ist damit ein handliches Werkzeug bei der Programm-Erstellung. Außerdem bietet Unity ein einfaches Lizenzmodell, welches eine kostenlose Nutzung für dieses Projekt ermöglicht. Der Grund, warum sich für dieses Projekt für Unity entschieden wurde, ist unter anderem die Tatsache, dass wir durch unseren Studiengang der Medieninformatik bereits Erfahrung mit Unity als Game-Engine sammeln konnten. In Wahlpflichtfächern wie prozedurale Generierung virtueller Welten und 3D-Modellierung und Virtuelle Realitäten, sowie Computer Game Development mit Unity wurde bereits mit Unity gearbeitet. Ein weiterer Grund, ist wie bereits erwähnt das Lizenzmodell der Engine, welches keine Kosten für dieses Projekt entstehen lässt. Des Weiteren ist der Umgang mit der Entwicklungsumgebung sehr anfängerfreundlich und der umfangreiche Asset Store beinhaltet viele kostenlose Hilfsmittel zur Erarbeitung des Projektzieles. So können unter anderem Virtual Reality Assets installiert werden, welche die Kommunikation zwischen HTC Vive und Laufzeitumgebung ermöglichen (vgl. [Unity3D, 2017](#)).

## **2 Vorstellung vorhandener VR-Editoren**

Dieses Kapitel präsentiert Ergebnisse der Domänenrecherche. Dadurch bietet sich ein Überblick über die Lösungsansätze und Funktionen anderer VR-Editoren. Der Schwerpunkt fällt dabei auf die Usability und das User Interface Design, sowie die wichtigsten Grundfunktionen der Anwendungen. Interessant ist dieser Beobachtungsfokus, da sich die Usability Experiences vieler VR-Editoren stark voneinander unterscheiden und manche Editoren unzählbar viele Funktionen bieten, während andere sehr sparsam ausgestattet sind. Einige sind damit nicht sehr nutzerfreundlich, sondern kompliziert zu benutzen. Grund dafür sind fehlende Standards, da die VR-Technologie immer noch in ihren Anfängen steckt. Die Recherchearbeit befasst sich in Kapitel 3 außerdem mit den Grundlagen der VR-Ergonomie.

Im Folgenden werden Masterpiece VR 2017, Blocks - Google VR, SculptrVR, VoxVR und Abstractron Voxulo vorgestellt und besprochen. Diese Anwendungen kommen am nächsten an die anfängliche Projekt-Vision eines Virtual Reality Voxel Editors. Begonnen wird mit den bekanntesten und erfolgreichsten Anwendungen.

### **2.1 Masterpiece VR 2017**

Masterpiece VR ist ein 3D-Modellierungsprogramm, dass sich durch Flexibilität und Dynamik auszeichnet. Nutzer können völlig freihand lehmartige Skulpturen formen, färben, meißeln und vieles mehr. Alleinstellungsmerkmal ist der professionelle Bearbeitungsgrad. Über eine aktive Internetverbindung können mehrere Personen gleichzeitig an einer Skulptur arbeiten und durch weitere Postproduktionen können Figuren sogar animiert werden und physische Eigenschaften erhalten.

Das User Interface befindet sich bei Masterpiece VR in der linken Hand - kann bei Belieben aber auch gewechselt werden. Dort liegt ein großes Menü in dem alle Einstellungen gefunden werden können. Auf den ersten Blick versteht man die Funktionen der Anzeigen nicht sofort und muss diese zunächst ausprobieren. Die rechte Hand wird dort als Cursor benutzt um einzelne Einstellungen auszuwählen, wie in Abbildung 2.1 gezeigt wird. Dabei behält der Controller in der Anwendung seine Form und wird nicht optisch an die Funktion angepasst. Der linke Daumen kann auf dem linken Controller leicht das Touchpad bedienen um die Farbe des Brushes zu verändern. Dafür streicht der Finger über die Oberfläche an die gewünschte Farbe. Der Nutzer kann sich frei im Raum bewegen und das erstellte Objekt greifen um es ebenfalls zu drehen.



ABBILDUNG 2.1: Masterpiece VR: User Interface  
[https://www.youtube.com/watch?v=aY\\_SBULE4qQ](https://www.youtube.com/watch?v=aY_SBULE4qQ)

## 2.2 Blocks - Google VR

Blocks VR ist eine VR-Modellierungsanwendung von Google. Sie ist kostenlos erhältlich und fällt durch einen besonderen Grafikstil auf. Die Objekte wirken wie gefaltete Papierfiguren. Der Detailgrad der Texturen ist standardmäßig gering. Obwohl gewöhnlicherweise mit Würfelformen gearbeitet wird, können Objekte auch abgerundet werden. Das bedeutet, dass neben Cubes auch Kugeln, Zylinder und Spitzen geformt werden können.

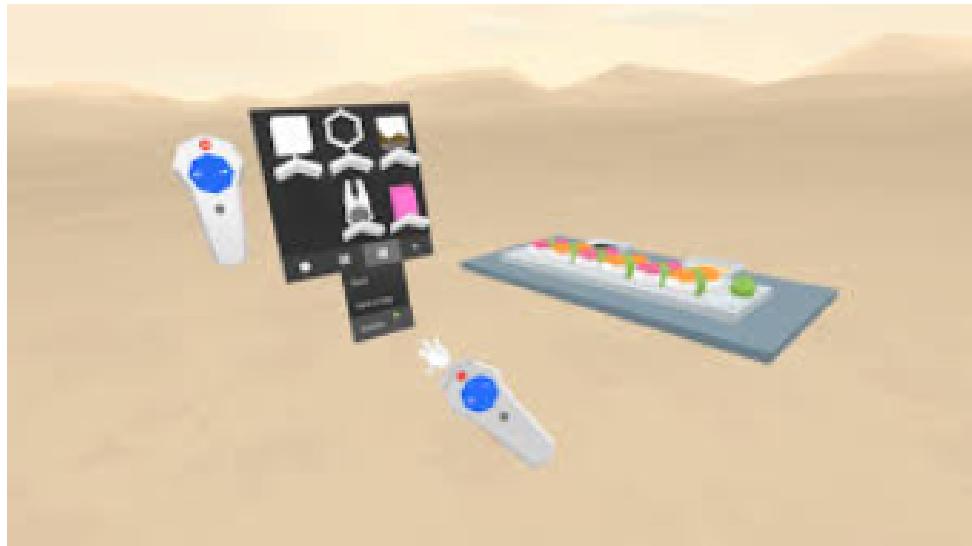


ABBILDUNG 2.2: Blocks Google VR: User Interface  
<https://www.roadtovr.com/google-announces-blocks-vr-app-like-tilt-brush-modeling-3d-objects/>

Auch hier können Nutzer Objekte völlig frei formen. Erstellte Objekte können unter Nutzern in einem Store geteilt werden und stehen zum Download zur Verfügung. Das UI befindet sich standardmäßig in der linken Hand. Dies kann nur getauscht werden, wenn der Nutzer bei jedem Programmstart das “Activator” Modul auf den linken Controller legt. Auch hier wird im ersten Fall die rechte Hand genutzt um einzelne Menüpunkte auszuwählen. Ein Nutzer kann die Optionen anzeigen indem die linke Hand hochgehalten und die Handfläche angeschaut wird. Bewegt der Spieler beide Hände zunächst nach unten und schnellt diese dann nach oben, kann er die Gravitation der Objekte ausschalten. Das Interface ist für unerfahrene Nutzer eher kompliziert. So bieten die Menüpunkte zu viel Auswahl und es ist anhand der Icons nicht immer deutlich welchem Zweck die einzelnen Werkzeuge dienen - siehe Abbildung 2.2. Auch die Handhabung unterscheidet sich zu Masterpiece VR. So findet ein Nutzer die Farbpalette hier auf der Rückseite seiner Hand, in dem er diese umdreht und dann einen Farbton auswählt, wie Abbildung 2.3 zeigt.

## 2.3 SculptrVR

SculptrVR ist ein von Nathan Rowe entwickelter VR Voxel Editor, der durch die Hinzugabe von natürlichen Elementen wie Wasser, Nebel, Licht, etc. besonders wird. So hat ein Nutzer die Möglichkeit mit einem VR Headset eine Minecraft-Welt zu erschaffen, in der er im Editor schon



ABBILDUNG 2.3: Blocks Google VR: Farbpalette  
<https://www.roadtovr.com/google-announces-blocks-vr-app -like-tilt-brush-modeling-3d-objects/>

Elemente wie zum Beispiel ein Meer hinzufügen kann. Das bringt den Schaffungsprozess sehr nah an das fertige Level. Außerdem können andere Nutzer an meinem Werk mitwirken und Objekte können wahlweise über physikalische Eigenschaften verfügen. Abbildung 2.4 zeigt eine Wasserwelt mit einer fliegenden Schildkröte



ABBILDUNG 2.4: SculptVR - Voxel Welt mit Wasser und Himmel  
<http://store.steampowered.com/app/418520/SculptrVR/>

SculptrVR löst sich allerdings, wie viele andere Konkurrenten, vom puren Cube-Look und ermöglicht ebenfalls ein Abrunden der Würfel (kommendes Feature 2017).

---

In der linken Hand befindet sich die Farbpalette und andere Menüpunkte. Durch Klicken auf Menüreiter können sich Nutzer in weitere Menüpunkte navigieren, wo jeweils Optionen zur Verfügung stehen. Dreht man diese um 180° findet der Nutzer weitere Einstellungen auf seiner Handfläche. In SculptrVR bewegt sich ein Nutzer fort indem er sich an eine gewünschte Stelle teleportiert oder mit den Controllern in der Hand eine Schwimmbewegung ausübt. Dabei kann er auswählen wie groß, also in welchem Maßstab, seine Figur erscheinen soll. Es ist also möglich, die Größe des Charakters zu verändern und z.B. sehr klein zu werden um Objekte zu betreten oder sehr groß zu werden um leichter gigantische Objekte zu erschaffen.

## 2.4 VoxVR

Vox VR ist ein Voxel Editor im Steam Greenlight. Er bietet die Basisfunktionen eines Standard Voxel Editors. Das sind das Platzieren von Blöcken, sowie das Entfernen und Einfärben. Das auf Steam erhältliche Programm teilt genau den Grafikstil, welcher auch in diesem Praxisprojekt eingehalten werden möchte. Ein Nutzer erstellt Objekte, indem er einzelne Cubes in einem Raum platziert (siehe Abbildung 2.5).

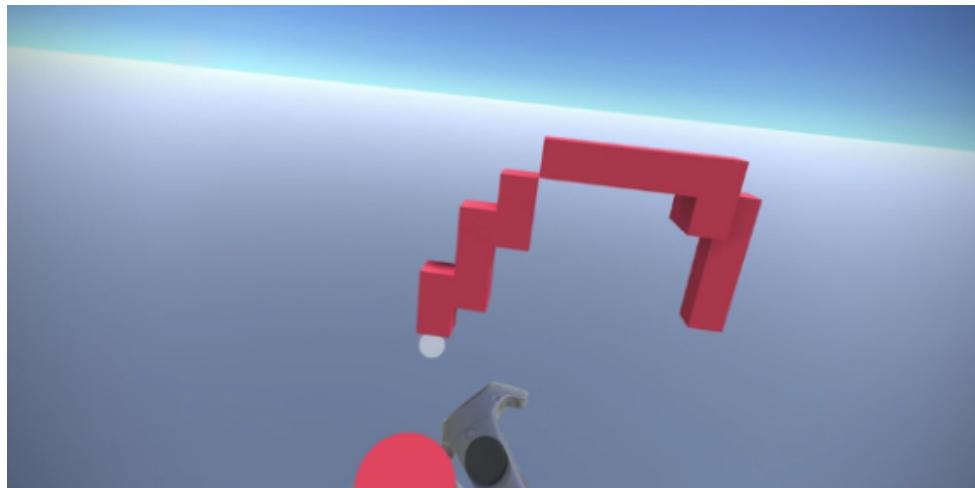


ABBILDUNG 2.5: VoxVR - Cubes platzieren  
<https://www.youtube.com/watch?v=Tc9GDFhK1jA>

Die Objekte stehen statisch im Raum während sich der Nutzer um sein Modell bewegen kann, um es zu bearbeiten. Das User Interface fällt hierbei sehr minimal aus. Es gibt keine Auswahlmöglichkeiten oder ein Menü. Ein Nutzer kann lediglich Cubes platzieren und mit den beiden

---

Daumen am Touchpad die Farbe der Würfel schon während der Platzierung verändern. Abbildung 2.6 zeigt eine in VoxVR erstellte Burg mit Landschaft. Klassische Voxel Objekte können im Anschluss in verschiedene Dateiformate importiert werden.



ABBILDUNG 2.6: VoxVR - Cube Level Beispiel

<https://www.youtube.com/watch?v=xmbJFmLn3E>

## 2.5 Abstractron Voxulo

Abstractron Voxulo ist ein kleines Projekt eines Hobby-Entwicklers. Das Programm verfügt ebenfalls lediglich über die Basics wie eine Farbpalette in der linken Hand und das Platzierungs-Tool in der rechten Hand mit dem Cubes gesetzt werden können. In Abbildung 2.7 ist zu sehen, wie der Nutzer mit einem Laserpointer die Farben auswählt. Interessant bei diesem Editor ist die Idee, dass die linke Hand gleichzeitig als Radierer wirkt mit welcher man Cubes entfernen kann. Damit wird das Arbeiten etwas agiler.

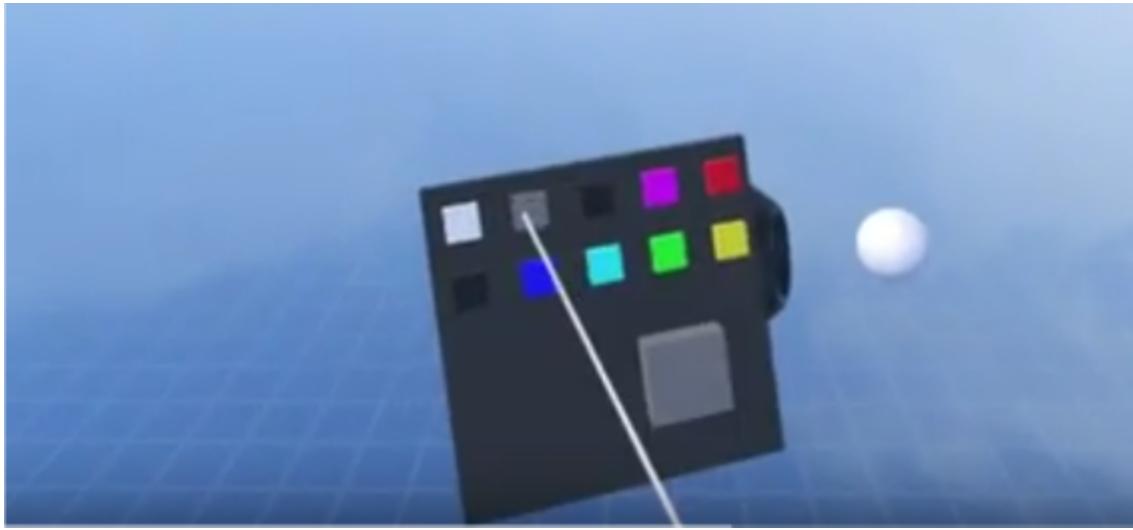


ABBILDUNG 2.7: Abstractron Voxulo - Farbpalette und Laserpointer  
<https://www.youtube.com/watch?v=KOdtT463uiU>

## 2.6 Erkenntnisse der Marktrecherche

Nach Betrachtung vorhandener VR-Editoren auf dem Markt wird deutlich, dass alle nach einem ähnlichen Schema arbeiten: Ein User benutzt die Controller in seiner Hand wie Werkzeuge, um Objekte zu erstellen. Ein Controller (meistens der rechte) dient dabei als Platzierungswerkzeug für 3D-Entitäten (Würfel, Zylinder, Kreise und ähnliches), während der linke Controller dazu dient die erstellten Objekte zu bearbeiten (zum Beispiel einfärben) oder ein User Interface zu bedienen, welches als Kontextmenü verschiedene Optionen besitzt. Ein User befindet sich mit seinen Objekten immer im gleichen Raum, um diese dort manipulieren zu können.

Grundfunktionen, die sich nach Betrachtung herauskristallisieren, sind das Platzieren, Entfernen, Einfärben und Manipulieren von Objekten. Außerdem werden dem User verschiedene Bewegungsmöglichkeiten ermöglicht: Die physische Bewegung durch den Nutzer selbst, das Teleportieren des Nutzers durch eine Teleportationsfunktion und eine analoge Controllersteuerung.

Der Grafikstil dieses Projekts soll sich auf eine einfache Darstellung konzentrieren - ähnlich wie VoxVR. Die Entitäten sollen sich auf einfarbige Würfel beschränken und keine abgerundeten Formen wie Zylinder oder Kreise annehmen. Ein zu großes und zu vielfältiges User Interface

wie bei Masterpiece VR bietet zwar viele Gestaltungsmöglichkeiten, fühlt sich bei der Bedienung aber sehr überwältigend und kompliziert an. Um ein selbsterklärendes und übersichtliches Programm zu gestalten, soll auf möglichst viel User Interface verzichtet werden. Die Bedienung soll selbsterklärend gestaltet werden. Einstellungen sollen soweit es möglich ist, über die Controller der HTC Vive getroffen werden.

## **3 Grundlagen in VR-Ergonomie, -User Interface und -User Experience**

Bevor man sich der Gestaltung von VR-Anwendungen widmen kann, müssen die ergonomischen Grundlagen verstanden werden. Denn die Ergonomie nimmt in diesem aktiven Medium eine große Rolle ein. Immerhin sitzen User bei der Verwendung eines Head Mounted Displays (HMD) nicht ruhig in einem Bürostuhl, sondern bewegen sich dabei, greifen nach Objekten, drehen sich um die eigene Achse, schlagen unter Umständen nach virtuellen Gegnern oder tätigen auch sonstige erschöpfende Handlungen. Außerdem unterscheidet sich die Art und Weise wie Nutzer das HMD im Vergleich zu konservativen Displays verwenden. Der Monitor steht nicht mehr einige Zentimeter entfernt auf dem Schreibtisch, sondern befindet sich in Miniaturausgabe direkt vor den Augen und der virtuelle Raum, der betrachtet wird liegt nicht mehr parallel und zweidimensional vor dem User, sondern umhüllt ihn mit einem 360 Grad Sichtfeld. Dabei entstehen völlig neue Gestaltungsregeln, was die Platzierung von 3D-Objekten und Interface-Flächen in einem virtuellen Raum angeht.

Deshalb ist es wichtig eine VR-Anwendung ergonomisch zu entwickeln, damit Nutzer vor Verletzungen geschützt werden und nicht an Augenschmerzen oder anderen Beschwerden leiden.

### **3.1 Motion Sickness & Gegenmaßnahmen**

Motion Sickness oder auch VR-Krankheit ist ein Krankheitsbild, das durch Verwendung von Head Mounted Displays in einem virtuellen Raum entstehen kann. Gewöhnlicherweise leiden Betroffene an Schwindel, Übelkeit und anderen Symptomen die auch bei Seekrankheit oder Reisekrankheit auftreten können. Dabei unterscheidet sich Motion Sickness aber durch die

fehlende physische Bewegung des Betroffenen von außen, wie es zum Beispiel während einer Bootsfahrt bei Wellengang der Fall wäre (vgl. [Patrao et al., 2017](#)).

Nicht jeder Mensch ist anfällig für Motion Sickness. Um die Symptome bei Usern zu verhindern, die darunter leiden, müssen ein paar technische und gestalterische Maßnahmen eingehalten werden.

Zu Beginn muss gewährleistet werden, dass die Hardware, also der PC oder die Spielekonsole über ausreichend Leistung verfügt um eine Bildwiederholrate von 90 Bildern pro Sekunde aufrecht zu erhalten. Es wird empfohlen den Sichtradius des Benutzers auf ca.  $94^\circ$  einzuschränken. Abbildung 3.1 gibt einen Eindruck der Sichtweite von 94 Grad. Je weniger Eindrücke einen Nutzer in einer VR-Anwendung ablenken, desto höher steigt die Orientierung.

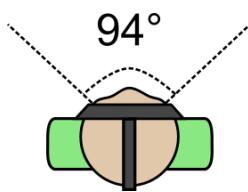


ABBILDUNG 3.1: Sichtweite  $94^\circ$

Bedenkt man dabei, dass ein Nutzer seinen Kopf zu beiden Seiten (links und rechts von der Mittellinie aus) dreht, wird ein Radius von  $204^\circ$  erreicht, wie Abbildung 3.2 demonstriert (vgl. [Alger, 2015](#)).

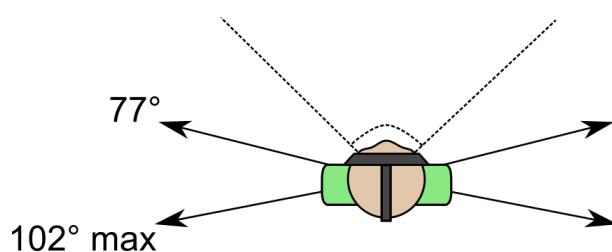


ABBILDUNG 3.2: Kombinierte Sichtweite

Die Geschwindigkeit in der die Körperbewegung des Nutzers zur Hardware übertragen und in der Anwendung umgesetzt wird, sollte bei weniger als 20ms liegen (Latenz). Fixe Bezugspunkte (Gegenstände, Nase, Cockpit) im Bild helfen bei der Wahrnehmung und suggerieren, dass die erlebte Bewegung nicht unnatürlich ist. Die Kamera, die dem Nutzer als Blick dient, darf nicht unnötig animiert und nicht unnatürlich bewegt werden. Während der Entwicklung

sollte immer wieder daran erinnert werden, dass die Kamerasicht eigentlich die Sicht aus den Augen des Nutzers ist. Deshalb dürfen keine Überschläge, Schraubdrehungen oder ähnliches Hektisches auftauchen. Eine analoge Controllersteuerung wie man sie aus First Person Videospielen kennt, kann eine gute Bewegungsalternative bzw. ein Zusatz sein. Des Weiteren sollte das HMD während der Konfiguration auf die Person eingestellt werden. Wichtige Faktoren, die dabei beachtet werden sollten sind zum Beispiel der Augenabstand und ein fester Halt der Brille auf dem Kopf (vgl. [Dannenberg, 2017](#)).

Während diese Maßnahmen die Motion Sickness schon stark reduzieren können, arbeiten die Produzenten von VR-Brillen immer noch an Weiterentwicklungen der Hardware, um die VR-Krankheit eines Tages komplett ausmerzen zu können.

## 3.2 Paradigmen zur Platzierung von Objekten und Interfaces in virtuellen Räumen

Die folgenden Paradigmen und Konventionen wurden von Samsung und Google definiert. Sie sind zur VR-Entwicklung äußerst wichtig und werden im folgenden Abschnitt kurz zusammengefasst.

Alex Chu, der ehemalige Lead Designer von Samsung VR, präsentierte 2014 wichtige Forschungsergebnisse, die VR-Designern der ganzen Welt ein Regelwerk bieten, wenn es darum geht zweidimensionale Interface-Konzepte in einen dreidimensionalen VR-Raum zu überführen (*Design: Transitioning from a 2D to 3D Design Paradigm*). Mike Alger, Interaction Designer für Virtual Reality bei Google, nutzte diese Erkenntnisse und definierte seine Idee der *Content Zones*, welche beschreibt in welcher Entfernung und welchem Winkel zum Nutzer welche Art von Interface liegen darf.

Chu beschreibt das FOV (Field Of View) von  $94^\circ$  eines VR-Nutzers und zeigt, dass eine Person ihren Kopf bequem bis  $77^\circ$  und maximal um  $102^\circ$  zu einer Seite (links oder rechts von der Mittellinie aus) drehen kann (siehe Abbildung 3.2). Abbildung 3.3 zeigt, dass der bequeme Neigungswinkel bei maximal  $60^\circ$  nach oben und bei maximal  $40^\circ$  nach unten liegt.

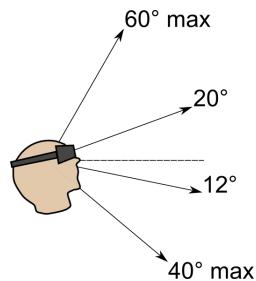


ABBILDUNG 3.3: Geneigtes Sichtfeld

Dabei hat der Kopf meistens eine Abwärtsneigung von  $6^\circ$  unter der Horizontlinie in der Bildmitte. Eine Rotation oder Neigung hinter dem jeweiligen Maximalwinkel wird nach kurzer Zeit als anstrengend empfunden. So sollten VR-Entwickler daran denken, dass Nutzer ein Interface niemals mit Anstrengungen hinter dem maximalen Dreh- und Neigungswinkel bedienen sollten. Außerdem erörtert Alex Chu in welcher Entfernung virtuelle Objekte in einem virtuellen Raum platziert werden sollten: Befinden sich Objekte im Abstand von bis zu 0.5 Metern, muss der Nutzer seinen Blick kreuzen. Ab einem Abstand von 1.3 Metern bis 20 Metern werden Objekte noch von der Tiefenwahrnehmung erkannt und wirken dreidimensional. Ab 20 Metern erscheinen 3D-Objekte schon so flach, dass sie nur mit Mühe von zweidimensionalen Entitäten zu unterscheiden sind (vgl. [Chu, 2014](#)). Abbildung 3.4 beschreibt obigen Absatz noch einmal mit Hilfe einer Grafik.

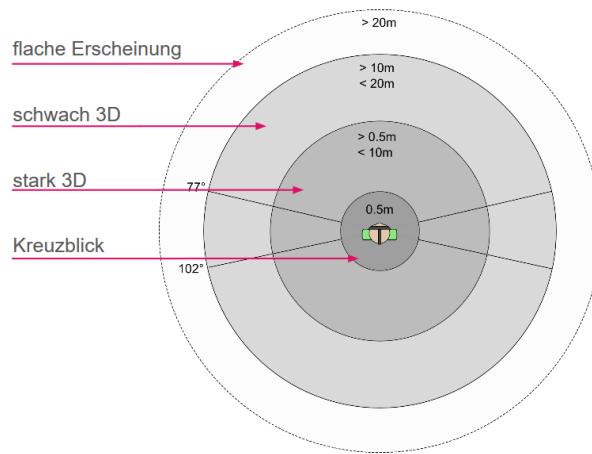


ABBILDUNG 3.4: Tiefenwahrnehmung in Relation zur Entfernung in VR

Alger nutzt diese Winkel und Entfernungen um Content Zones abzugrenzen. So sollten Interfaces und Objekte die vom Nutzer während der Anwendung am längsten angeschaut werden, bei einer Entfernung von 1,3 Metern liegen. Dabei neigt ein User seinen Kopf durchschnittlich  $15^\circ$

abwärts. Alles unter einem Abstand von 0,5m wird als No-No Zone bezeichnet. Dort sollte sich keine Grafik permanent aufhalten. Die Main Content Zone befindet sich in einem Winkel von  $144^\circ$  ( $-77^\circ$  bis  $77^\circ$ ), mit einem Abstand von bis zu 20 Metern ab der No-No Zone, vor dem User. Hier sollte alles liegen womit der Nutzer am häufigsten arbeitet. In Armreichweite des Nutzers befindet sich die **Main Work Zone**. Die **Touch Interface Zone** ergibt sich aus einer Subtraktion der Main Work Zone und der No-No Zone. Hinter dem Nutzer, ab einem Winkel von  $204^\circ$  ( $-102^\circ$  bis  $102^\circ$ ), liegt die Curiosity Zone für die ein Nutzer seinen ganzen Körper drehen muss. Zwischen der Main Content Zone und der Curiosity Zone befindet sich die Peripheral Zone. In dieser sollten keine essentiellen, sondern nur flüchtige Informationen liegen, die nicht immer wieder aufgegriffen werden müssen. Da die Linsen der HMD und die Sichtweiten immer kreisrund sind, müssen alle Interfaceanzeigen, die den Nutzer umgeben auch gebogen sein (vgl. [Alger, 2015](#)). Die wichtigsten Zonen werden in Abbildung 3.5 vorgestellt.

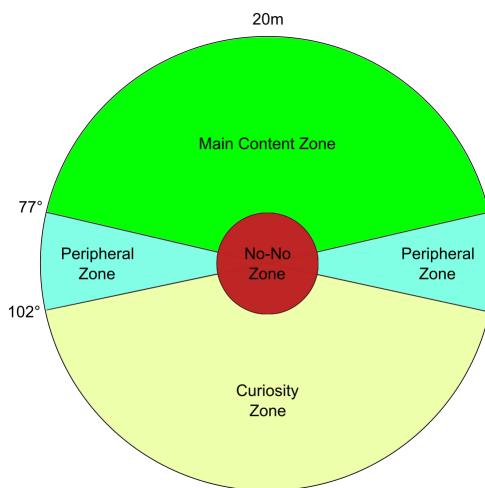


ABBILDUNG 3.5: Content Zones nach Alger

Google erklärt in seiner Googles Developer Konferenz 2017 zum Thema *Designing Screen Interfaces for VR*, dass das Zentrum des VR-UI ca.  $6^\circ$  unter der Horizontlinie liegen sollte, damit es im natürlichen Sichtzentrum der Augen von  $60^\circ$  liegt und immer mit einem Blick erfasst wird. Grund dafür ist die Ergonomie des menschlichen Auges, die besagt, dass ein bequemer Sichtradius bei  $-30^\circ$  (max.  $-35^\circ$ ) bis  $30^\circ$  (max  $35^\circ$ ) liegt - unabhängig vom FOV mit HMD (vgl. [Google I/O 17, 2017](#)).

### 3.3 Screen und User Interface Design in VR

Die Design-Experten von Google bemühen sich um den Platz des Vorreiters und Innovators in der VR-Branche. Durch Googles Daydream Labs wurden viele Erkenntnisse in der Gestaltung der User Interfaces für VR-Anwendungen gesammelt, welche im folgenden Kapitel zusammengefasst werden und als Leitfaden dienen sollen.

#### 3.3.1 Distance-Independent Millimeter als VR-Maßeinheit

Medien haben in ihrer Entwicklung eine auf sie abgestimmte und beabsichtigte Nutzungsdistanz erhalten, aus der Nutzer ihre Oberfläche betrachten sollen. Eine Armbanduhr betrachtet man aus halber Armlänge, sowie auch ein Handy; eine Kinoleinwand betrachtet man aus mehreren Metern und schaut dabei schräg aufwärts und Werbeplakate neben der Schnellstraße betrachtet man aus mehreren hundert Metern. All diese Objekte lassen sich in einem virtuellen Raum auch abbilden und beliebig in diesem platzieren. Doch die beabsichtigte Distanz des HMD-Mediums bleibt immer gleich und befindet sich unmittelbar vor den Augen des Users (vgl. [Google I/O 17, 2017](#)).

Deshalb benötigt ein VR-Entwickler eine gleichmäßige winkelförmige Größe, bzw. Einheit, die diese Objekte in Relation ihrer virtuellen Größe innerhalb eines virtuellen Raumes realistisch und gut erkennbar abbilden kann. Eine von Google entwickelte Maßeinheit, die diese Möglichkeit bietet, nennt sich **dmm** (distance-independent millimeter). Diese Einheit beschreibt jeweils einen Millimeter Höhe oder Breite aus einem Meter Entfernung vom Nutzer aus gesehen. Dabei entspricht ein distance-independent millimeter **einem Pixel**. Die Entfernung vervielfacht sich gleichmäßig zur Höhe / Breite. Das bedeutet, 2 dmm beschreibt 2 mm Höhe aus 2 Meter Entfernung und 0,5 dmm beschreiben eine Höhe von 0,5 mm pro halben Meter Entfernung zum Nutzer (vgl. [Google I/O 17, 2017](#)).

**Anwendungsbeispiel:** Angenommen ein Objekt steht in einem Meter Entfernung zum User und ist 480 dmm hoch und 400 dmm breit. So muss das nächste Objekt in zwei Metern Entfernung die doppelten Maße, also eine Höhe von 960 dmm und eine Breite von 800 dmm betragen. Logischer Weise muss das nächste Objekt in drei Metern Entfernung 1200 dmm breit und 1440

dmm hoch sein, damit alle drei Objekte aus der Ursprungsentfernung des Nutzers gleich groß aussehen und sich in bewegter Animation gleich schnell bewegen (vgl. [Google I/O 17, 2017](#)).

Für unsere Entwicklung ist zu beachten, dass die Maßeinheiten in Unity in Metern angegeben sind. Das bedeutet, für eine Design Übernahme in dmm-Angaben müssen diese Werte durch 1000 geteilt werden (vgl. [Sahin, 2015](#)).

### 3.3.2 Kenngrößen zum UI-Design

Die folgende Tabelle 3.1 zeigt Textgrößen, welche man für VR-Anwendungen sorglos benutzen kann. Außerdem zeigen sie eine minimale und optimal Treffgröße für Oberflächen, die ein Nutzer in VR berühren kann. Bei Verwendung ist allerdings darauf zu achten, dass sich diese Kenngrößen ändern werden, sobald die Pixeldichte der HMD-Produkte zunimmt (vgl. [Google I/O 17, 2017](#)).

TABELLE 3.1: UI Kenngrößen

Textgröße	Treffgröße
Überschrift:	40 dmm
Titel:	32 dmm
Unterüberschrift:	28 dmm
Fließtext	24 dmm
Beschriftung	20 dmm
Schaltflächen:	24 dmm
	Minimal: 64x64 dmm + 16 dmm Padding Optimal: 96x96 dmm + 16 dmm Padding

## 3.4 Usability und Experience

Usability hat dann sein Ziel erreicht, wenn die Anwendung so selbsterklärend ist, dass ein Nutzer sofort weiß, was er zu tun hat und intuitiv versteht, wie er Absichten im Programm umsetzen kann, ohne dass Ausführungsschritte erst erklärt werden müssen.

In VR übernimmt der Controller, bzw. das Werkzeug, das er digital darstellt, einen großen Teil dieser Aufgabe. Denn die Erscheinung des Controllers in der Anwendung bestimmt sein Nutzen. Angenommen ein Nutzer befindet sich in einem VR-Garten und sein Controller bildet eine Gießkanne ab, so versteht der Nutzer intuitiv, dass die Gießkanne gekippt werden muss, um

beispielsweise Blumen zu gießen. So initiiert eine Tennisschläger-Erscheinung eine Schlagausführung, eine Angel wird ausgeworfen und eine Pfanne wird mit ihrem Inhalt über eine Kochfläche gehalten. Ist das VR-Werkzeug der Controller-Haptik entsprechend designed, steigt die Immersion, da man sich als User gut vorstellen kann, den virtuellen Gegenstand im gleichen Moment wirklich in der Hand zu halten. Ein User erwartet eine gleiche physikalische Nutzung mit dem Objekt wie in der realen Welt. Auch eine handähnliche Darstellung sollte vermieden werden (vgl. [Google I/O 16, 2016](#)). Wird der statische Controller in der VR-Anwendung als menschliche Hand abgebildet, suggeriert diese Erscheinung zu viele Handlungsmöglichkeiten, die aus dem realen Leben bekannt sind, wie das Bewegen jedes einzelnen Fingers, Greifen oder das Ballen einer Faust, die mit der aktuellen Hardware aber nicht ermöglicht werden können.

VR-Controller arbeiten sehr präzise, trotzdem dürfen keine präzisen Handlungen eines Nutzers erwartet werden. Es müssen unpräzise Handlungen abgefangen werden, um sie daraufhin zu korrigieren. Angenommen ein Nutzer möchte in einem VR-Wohnzimmer ein Bild an die Wand hängen und wirft dieses gegen die Wand um sein Ziel zu erreichen. So sollte die Anwendung das Bild bei Kollision an der Wand abfangen und dieses gerade ausgerichtet aufhängen (vgl. [Google I/O 16, 2016](#)).

Interfaces oder Objekte, die vom Nutzer berührt oder benutzt werden, sollten animiertes oder audiovisuelles Feedback zurückgeben, damit sie selbsterklärend sowie intuitiv werden (vgl. [Google I/O 16, 2016](#)).

Immersion soll in VR immer die größte Rolle spielen. Kleine Räume können gigantisch wirken, wenn der Spielcharakter in Relation winzig ist. Licht, Schatten und Texturen helfen immer dabei Immersion zu erhöhen und die Tiefenwahrnehmung realistischer zu gestalten, um beispielsweise Bewegungen leicht erkennbar zu machen. Außerdem sollten wichtige Informationen erkenntlich visualisiert werden, indem sie z.B. aufleuchten. Die reale Welt ist nie komplett ruhig, so sollte eine VR-Anwendung auch immer in irgendeiner Form Akustik wiedergeben (vgl. [Google I/O 16, 2016](#)).

Ein Spielercharakter wird standardmäßig nur durch seine Hände (Controller) und seinem Kopf (HMD) in einer Anwendung visuell dargestellt. Einen Körper unterhalb dieser Gliedmaßen zu

platzieren, der beispielsweise Beine und einen Körper animiert darstellt, würde unnötige Rechenleistung bedeuten. So ist es äußerst komplex die Fußplatzierung vorauszusagen und zu berechnen, wenn man als Ausgangsinformation lediglich den Standort der Controller und der VR-Brille erhält. (vgl. [Google I/O 16, 2016](#)).

# **4 Konzept**

Das Gesamtkonzept in diesem Kapitel beschreibt die anfänglich geplante Vision des Virtual Reality Voxel Editors. Im folgenden Abschnitt wird zunächst die Tastenbelegung der HTC Vive Controller betrachtet. Ist klar, welche Bedienmöglichkeiten zur Verfügung stehen, kann überlegt werden, mit welchen Tastenkombinationen Funktionen ausgeführt werden sollen. Diese geplanten Funktionen werden anschließend als Soll-Zustand präsentiert. Für jede Funktion wird eine visuelle Darstellung (User Interface) und eine effiziente Bedienmöglichkeit (Usability) beschrieben. Außerdem wird erläutert, welcher Zweck durch die Funktionen erfüllt wird.

## **4.1 Controlleraufbau und Tastenbelegung - HTC Vive**

Dieses Unterkapitel beschäftigt sich mit der Frage, welche Tastenbelegung sinnvoll für einen VR Voxel Editor ist und berücksichtigt dabei die Erkenntnisse der Domänenrecherche. Außerdem folgt die Auseinandersetzung mit der Frage wie sich ein Nutzer im Raum idealerweise fortbewegen kann, um sein Objekt zu bearbeiten. Dafür werden verschiedene Lösungsmöglichkeiten betrachtet und diskutiert. Zu Beginn folgt die Grafik 4.1, um die aktuelle Tastenbelegung zu veranschaulichen.

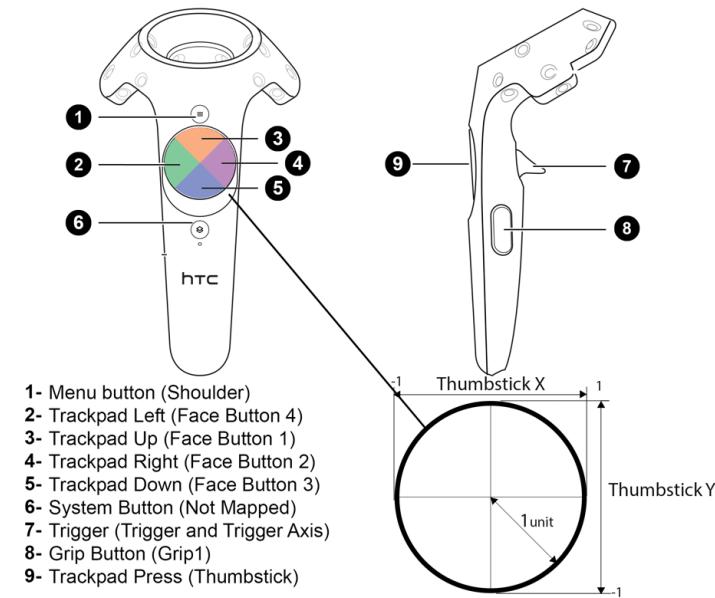


ABBILDUNG 4.1: HTC Vive - Tastenbelegung

<https://forums.unrealengine.com/development-discussion/vr-ar-development/78620-steam-vr-template>

Die HTC Vive Controller sind zwei baugleiche zylinderförmige Griffe mit einem Kreisrunden Sensorkopf. Auf dem Sensorkopf befinden sich mehrere eingedrückte Ovale, welche die eingeübten Sensoren markieren. Um Gewicht zu reduzieren und damit die Sensoren nicht verdeckt werden, besitzen die HTC Vive Controller eine durchlässige Aushöhlung im Sensorkopf. Die Ziffer 1 in der Grafik zeigt den Menü-Knopf. In der Regel lässt sich damit das Menü der aktiven Anwendung öffnen. Das Trackpad darunter besteht aus vier Flächen, die jeweils wie eine Taste agieren. Es lässt sich mit dem Daumen bedienen, reagiert auf Druck, aber auch auf einfache Berührung. Durch den Systemknopf (Punkt 6) lässt sich das Dashboard der SteamVR Umgebung öffnen. Der Trigger auf Rückseite des Controllers ist für den Zeigefinger bestimmt. Die Belegung und Bedienung erinnert an bekannte Konsolen. So hat der B-Knopf der Nintendo Wii an der Wii-Mote einen ähnlichen Ort und Nutzen wie der Trigger der HTC Vive. In den meisten Anwendungen wird der Trigger als Benutzen- oder auch Aktions-Taste verwendet. Das kann ein Abfeuern einer Handfeuerwaffe in einem Shooter, aber auch das Platzieren eines Blockes in einem Voxel Editor sein. An der unteren Seite besitzt der Controller links und rechts jeweils einen Grip-Button (vgl. Schilling, 2016).

Mithilfe der HTC Vive Controller werden die Handbewegungen der Nutzer erkannt, übernommen und damit adaptiert. Nach Betrachtung anderer Voxel Editoren stellt sich nun die Frage, welche Tastenbelegung sinnvoll und nutzerfreundlich wäre. Im Kapitel Funktionsplanung sind die Funktionen beschrieben. Dort folgt auch eine Beschreibung welche Tasten für welche Funktionalitäten belegt werden. Wichtig dabei ist eine barrierefreie und intuitive Steuerung zu bestimmen.

## 4.2 Funktionsplanung

Dieses Unterkapitel beschäftigt sich, wie zu Beginn des Kapitels beschrieben, mit den Funktionen des Editors, die zu Beginn des Projekts geplant wurden. Jede Funktion soll in Form eines Werkzeuges (Hammer, Spitzhacke, usw.) dargestellt werden. Die Werkzeuge sollen visuell in der Hand des Nutzers dargestellt werden. Mit ihrer äußereren Erscheinung in Form von Werkzeugen sollen die angebotenen Funktionen selbsterklärend sein, sodass der Nutzer versteht, welcher Nutzen daraus gezogen werden kann. Alle 3D-Modelle der Werkzeuge wurden von den Autoren dieses Praxisprojekts selbst in Blender erstellt. Außerdem ist es möglich, alle Tasten des rechten Controllers pro Werkzeug (Funktion) mit neuen Einsatzmöglichkeiten zu belegen. Bis zum späteren Kapitel Umsetzung wurden die Funktionen iterativ überarbeitet, ergänzt, angepasst und diverse Details auch begründet entfernt.

### 4.2.1 Cubes platzieren

Grundstein eines Voxel Editors ist das Platzieren von Voxel Elementen. Nutzer sollen die Möglichkeit erhalten Cubes (Würfel) in der Spielwelt zu platzieren um 3D-Objekte zu erstellen. Dargestellt werden soll diese Funktionen durch einen 3D-modellierten Zimmermannshammer, welcher durch seine Erscheinung die Absicht etwas herzustellen suggerieren soll. Dieser soll sich virtuell in der rechten Hand des Nutzers befinden. Ein grünes Highlighting soll dabei den aktuell selektierten Cube markieren. Außerdem soll die Markierung als Vorschau dienen und darstellen an welcher Stelle der nächste Würfel platziert werden kann, wenn der Nutzer seine Auswahl bestätigen sollte. Die Bausteine sollen innerhalb eines unsichtbaren dreidimensionalen Rasters aneinander gereit und verbunden werden.

Durch Betätigen des Grip-Buttons des rechten Controllers soll das Werkzeug durch die Werkzeugauswahl rotiert und ausgewechselt werden. Cubes sollen durch das Bewegen des Controllers selektiert und dementsprechend der aktive Cube immer markiert werden. Durch das Drücken der Trigger-Taste am rechten Controller soll ein Cube an der selektierten Stelle platziert werden können.

Durch das gegenseitige Einrasten der Cubes innerhalb des Mesh-Gitters soll ein hoher Detailgrad in Form von präziser Platzierung und leichter Handhabung garantiert sein. In Abbildung 4.2 ist zu sehen, wie zwei Cubes lückenlos nebeneinander platziert wurden. Auf eine realistische Schlag- oder Schwingbewegung des Hammers, um Cubes zu platzieren soll verzichtet werden. Damit sollen auch Nutzer in kleinen Räumen mehr Spielraum erhalten und Verletzungen vermieden werden. So ist der Nutzer nicht an die Reichweite seiner Armlänge gebunden und kann Cubes auch außerhalb dieser Distanz platzieren. Außerdem ist diese Einschränkung notwendig, um in einem Editor effizient arbeiten zu können. Müsste ein Nutzer für jeden Cube, von angenommen 100.000, eine Schwingbewegung ausführen, wäre das nicht nur zu anstrengend, sondern auch zu zeitaufwendig. Diese Erkenntnis wurde während der Umsetzungphase erprobt und wird im Kapitel Umsetzung weiter erläutert.

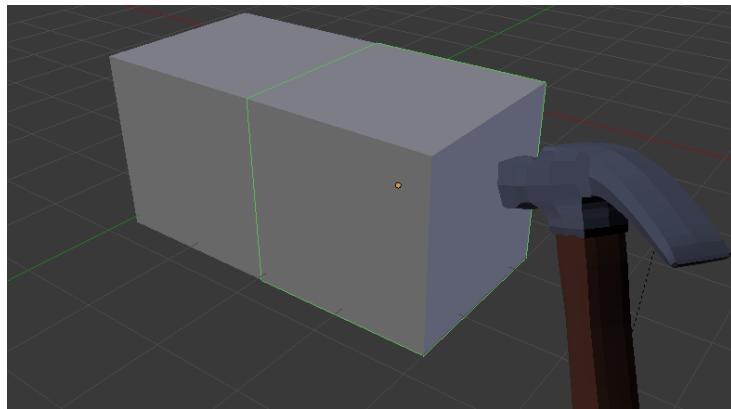


ABBILDUNG 4.2: Cubes platzieren

#### 4.2.2 Cubes entfernen

Gesetzte Cubes sollen auch entfernt werden können. Die Funktion des Entfernen soll durch eine Spitzhacke dargestellt werden, welche in der rechten Hand des Nutzers gehalten werden soll. Eine Spitzhacke symbolisiert den Abbau von Objekten. Werden Cubes mit der Spitzhacke

als Werkzeug selektiert, sollen diese durch ein gelbes Highlighting markiert werden, wie in Abbildung 4.3 zu sehen ist. Damit soll ein Unterschied zur Selektion mit dem Hammer erkennbar sein.

Die Spitzhacke soll ebenfalls durch Betätigen des Grip-Buttons ausgewählt werden. Die Selektion mit diesem Werkzeug soll wie der Hammer durch Controllerbewegung stattfinden. Betätigt der Nutzer die rechte Trigger-Taste während er die Spitzhacke trägt, sollen selektierte Cubes entfernt werden.

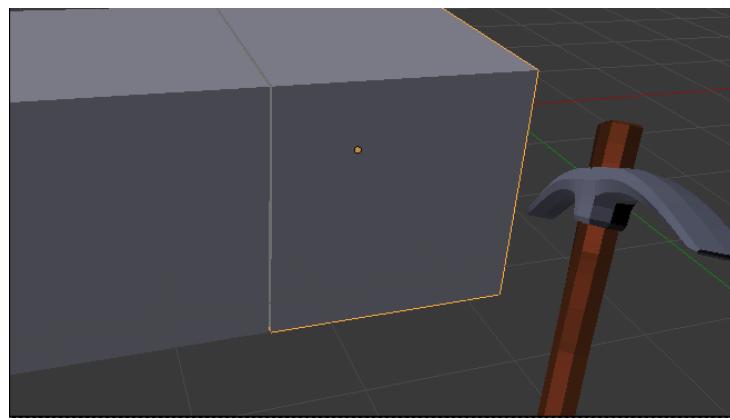


ABBILDUNG 4.3: Cubes entfernen

#### 4.2.3 Cubes einfärben

Cubes sollen unterschiedlich eingefärbt werden können. Dabei wird an zwei unterschiedliche Use Cases gedacht. Ein Nutzer möchte einem Cube vor dem Platzieren eine Farbe verleihen oder ein Nutzer möchte die Farbe eines platzierten Cubes nachträglich verändern.

Die Farbe soll auf dem Touchpad des linken Controllers permanent angezeigt werden. Zu sehen soll der Farbwert, die Sättigung und der Dunkelwert der auszuwählenden Farben sein.

Da sich die Farbpalette nicht auf dem rechten, sondern dem linken noch freien Controller befinden soll, kann der Nutzer das Touchpad jeder Zeit steuern um agil eine Farbe auszuwählen. Durch leichtes Streichen über das Touchpad soll zunächst ein Farbwert, dann eine Sättigung und anschließend ein Dunkelwert ausgewählt werden. Das GUI soll dabei kreisrund wie das Touchpad des Controllers aussehen und so die Immersion erhöhen - Abbildung 4.4 stellt dies grafisch dar.

Bei der Konzeptionierung dieses Features ist ein großer Unterschied zu anderen Editoren auf dem Markt zu erkennen. Andere Anwendungen wie Googles Blocks bieten nur eine Farbpalette die vom Nutzer in der linken Hand getragen und mit einem Laserpointer in der rechten Hand benutzt wird. Dabei muss der Nutzer mit dem Laserpointer auf die gewünschte Farbe zielen um diese auszuwählen. Diese Methode erweist sich als umständlich und zeitaufwändig.

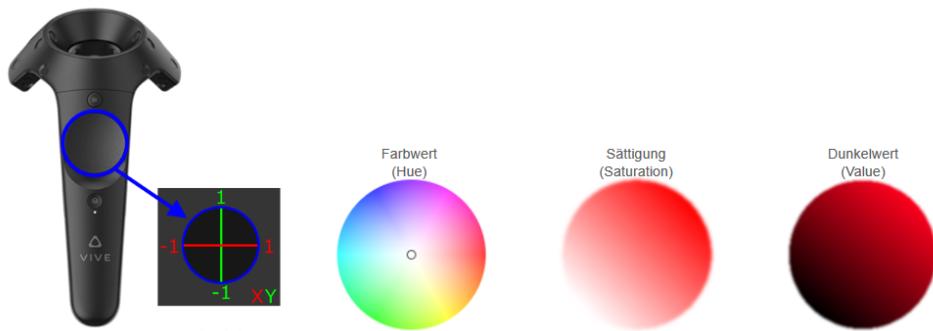


ABBILDUNG 4.4: Farbauswahl Touchpad  
<https://koenig-media.raywenderlich.com/uploads/2016/12/ViveControllerButtons.png>

#### 4.2.4 Teleportation

Nicht jeder Nutzer verfügt über einen privaten großen Raum, indem er VR-Anwendungen benutzt und sich fortbewegen kann. Damit Nutzer in Voxel VR jedes Objekt erreichen können, das sie bauen, reicht der physische Bewegungsradius durch die getrackte Bewegung der Sensorik der HTC Vive nicht aus. Eine Teleportations-Funktion bietet in diesem Fall komplett Bewegungsfreiheit, durch die sich ein Nutzer in oder auf komplexen und großen Objekten fortbewegen kann. Diese Fortbewegungsart hat den Vorteil, dass sie kaum Motion Sickness triggert, wenn sie wie ein effektloser Bildwechsel ausgeführt wird. Grund dafür ist die sehr schnelle filmschnittartige Bewegung, die das menschliche Hirn gut verarbeiten kann, bzw. aus dem Medienalltag gewohnt ist (vgl. [VRScout, 2016](#)).

Ist die Teleportations-Funktion in der Anwendung aktiviert, soll ein Lichtbogen aus der linken vorderen Controllerspitze projiziert werden. Am Ende des Lichtbogens soll eine Fläche erleuchten, welche die Stelle symbolisiert auf die sich der Nutzer teleportieren kann.

Dabei soll das Halten der Trigger-Taste des linken Controllers dazu dienen den Lichtbogen zu projizieren. Durch Neigen des Controllers soll der Lichtbogen bewegbar sein. Lässt der Nutzer

die Trigger-Taste los soll die Auswahl des Ortes bestätigt und der Nutzer dorthin teleportiert werden. Abbildung 4.5 stellt dieses Vorhaben beispielhaft vor.

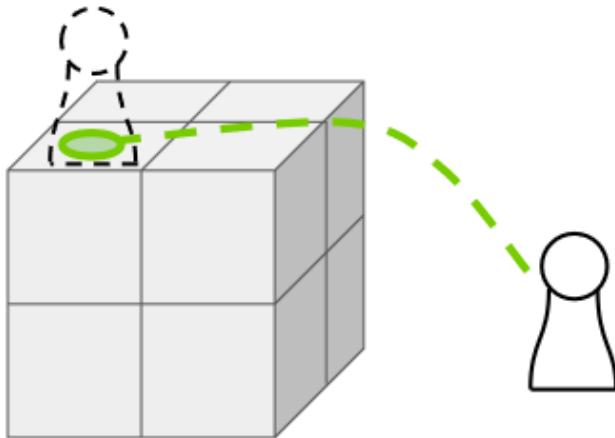


ABBILDUNG 4.5: Teleportation

#### 4.2.5 Analoge Controllersteuerung

Neben der getrackten Charaktersteuerung über die Sensoren der HTC Vive und der Teleportation, soll es noch einen weiteren Bewegungsmodus geben - die analoge Controllersteuerung. Diese soll eine feine und detaillierte Bewegung des Spielcharakters ermöglichen. Erfahrene Nutzer kennen diese Art und Weise der Steuerung bereits von Videospielen bei Bedienung eines Gamepads.

Der Nutzer soll sich mit Hilfe einer analogen Controllersteuerung über das rechte Touchpad statisch über eine Translationsachse (X-, Y-, und Z-Achse) im Raum bewegen können. Das Touchpad soll dabei ähnlich zu bedienen sein wie ein analoger Control-Stick eines Gamepads. Der Nutzer soll sich eine Achse auswählen auf der er sich bewegen möchte. Daraufhin soll er sich nur noch auf der ausgewählten Achse statisch fortbewegen können indem er mit dem Daumen über das Touchpad in die gewünschte Richtung streicht.

Dabei ist wichtig sich nur auf einer Translationsachse bewegen zu dürfen, damit Motion Sickness nicht ausgelöst werden kann. Durch diese Beschränkung kann eine natürlich Bewegung aus dem Alltag simuliert werden. Eine Bewegung über die X-Achse fühlt sich ähnlich wie eine Zugfahrt an, bei der aus dem Fenster geschaut wird, während sich der Zug nur horizontal

auf den Schienen nach rechts oder links bewegen kann. Eine Steigung und Senkung auf der Y-Achse limitiert, versteht der Nutzer schnell wie eine Aufzugfahrt welche entweder aufwärts oder abwärts gehen kann. Ein Fortgang über die Z-Achse fühlt sich wie eine Autofahrt als Beifahrer an, bei der sich das Fahrzeug entweder nach vorne oder nach hinten bewegt. So ist diese Bewegungsbeschränkung auf einer Translationsachse leichter zu verarbeiten und führte bei Tests nicht zu Motion Sickness.

Die Abbildung 4.6 zeigt einen Anwendungsfall der analogen Controllersteuerung: Angenommen ein Nutzer baut eine Säule aus drei Cubes, die jeweils fast größer sind als der Spielercharakter selbst. Nun möchte er den Cube in der Mitte weiter bearbeiten, doch erreicht diesen nicht. Mit seiner eigenen Körperbewegung erreicht er diese Höhe nicht und mit der Teleportation hat er lediglich die Möglichkeit die Oberfläche des höchsten Cubes zu betreten. Die einzige Möglichkeit zu diesem Zeitpunkt wäre, einen zweiten Cube auf der untersten Ebene neben dem ersten Cube zu platzieren und sich auf dessen Oberfläche zu teleportieren, um den mittleren Cube zu erreichen. Doch dies wäre mit extra Aufwand verbunden, da er nicht nur seine Sichtweise verändert und gegebenenfalls einen Achsensprung in Kauf nimmt, sondern den vierten Cube auch wieder entfernen müsste. Mit der analogen Controllersteuerung erhält der Nutzer nun die Möglichkeit sich sehr einfach auf der Y-Achse hinauf zu bewegen um den mittleren Cube zu erreichen. Anschließend könnte er sich wieder zurück auf den Boden begeben.

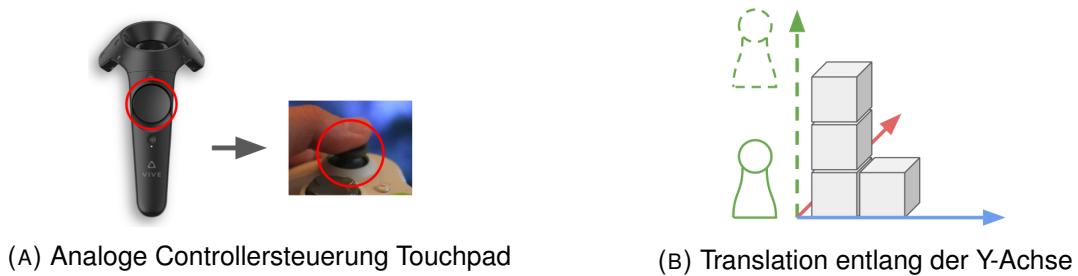


ABBILDUNG 4.6: Beispiel Analoge Controllersteuerung

#### 4.2.6 Cubes gruppieren

Mehrere erstellte Cubes sollen markiert und anschließend gruppiert werden können um daraus zusammengefasste GameObjects zu erstellen. Diese sollen damit als einheitliche Entität bearbeitbar werden, wie Abbildung 4.7 darstellt. Dies könnte eine einheitliche Einfärbung oder eine

Manipulation wie Translation, Skalierung, Rotation oder Verzerrung des Objekts sein. Einzelne Cubes eines Objekts sollen nicht manipulierbar sein, bis die Gruppe wieder aufgelöst wurde. Nur Objekte sollen manipulierbar sein.

Das Gruppierungswerkzeug soll wie die anderen Werkzeuge in der rechten Hand gehalten werden. Damit sollen Cubes markiert werden. Markierte Cubes sollen eine Außenlinie erhalten und leicht aufleuchten um erkennbar zu werden. Die Außenlinien der Cubes sollen an Ecken und Kanten Griffpunkte besitzen um mögliche manipulative Eingriffe sichtbar darzustellen.

Hat der Nutzer das Werkzeug zur Gruppierung mit der rechten Grip-Taste ausgewählt, soll er mit der gehaltenen rechten Trigger-Taste einen Kasten in die Luft zeichnen können, um platzierte Cubes damit zu markieren. Wird die Trigger-Taste losgelassen, sollen die Cubes markiert bleiben. Wird die Grip-Taste ein zweites Mal nach der Markierung gedrückt, soll die Markierung bestätigt werden und die Cubes zu einem Objekt zusammengefasst werden. Gelöst werden soll eine Gruppe erst dann wieder, wenn sie ein zweites Mal markiert wird und der Nutzer daraufhin die Grip-Taste betätigt.

Ein Anwendungsbeispiel soll den Nutzen dieser Funktion demonstrieren: Ein Nutzer baut aus vielen Cubes ein Haus. Nun möchte er dieses Haus nachträglich einfärben. Ohne die Gruppierungsfunktion müsste er jeden Cube einzeln ansprechen und färben. Nun hat er allerdings die Möglichkeit alle Cubes zu gruppieren und das erstellte Objekt einmal einzufärben. Daraufhin erhalten alle Cubes der Gruppe die gleiche Farbe.

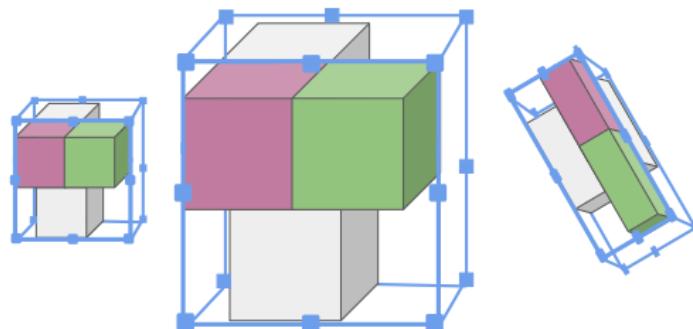


ABBILDUNG 4.7: Gruppierte Entitäten

#### 4.2.7 Objekte manipulieren

Gruppierte Objekte sollen manipulierbar sein. Da die Tasten für die Gruppierung allerdings schon belegt wurden, wird eine separate Manipulationsfunktion benötigt um mit jeder Taste neue Möglichkeiten zu bieten.

Objekte die mit dem Manipulationswerkzeug selektiert wurden, sollen ihre Kanten, Ecken und den Objektmittelpunkt mit hellblauen Punkten anzeigen. Daraufhin weiß der Nutzer, dass diese Punkte greifbar sind.

Mit einer Handbewegung und dem Halten der Trigger-Taste soll an den Ecken des Objekts gezogen werden um dieses zu vergrößern. zieht der Nutzer die Ecken zum Mittelpunkt soll das Objekt verkleinert werden. zieht der Nutzer an den Kanten soll das Objekt gestreckt und gestaucht werden. Durch das Greifen des Mittelpunktes soll das Objekt bewegt werden können. Wird der Mittelpunkt gegriffen und gleichzeitig mit dem rechten Daumen das Touchpad bedient, soll sich das Objekt drehen lassen.

# 5 Umsetzung

In diesem Kapitel soll ergebnisorientiert dokumentiert werden welche Arbeitsprozesse seit der Konzeption in das Projekt eingeflossen sind. Dazu gehört der iterative Prozess der Umsetzung der Funktionen. Als ergonomischen Standard platzieren wir die grafischen Anzeigen der Funktionen in der *Touch Interface*- bis *Main Work-Zone*. Das standardmäßige Field Of View der HTC Vive befindet sich bei  $112^\circ$  und wurde in diesem Projekt nicht verändert.

Durch mehrere Tests während des Projekts konnten immer wieder neue Erkenntnisse ermittelt werden, die zur Verbesserung der Anwendung beigetragen haben und dazu führten, dass die anfangs vorgestellten Funktionen angepasst und erneuert wurden.

Alle von den Autoren entwickelten Skripte innerhalb des Unity Projekts befinden sich im anhangenen Projekt im Ordner '/Assets/Scripts'. Falls Teile vom Code mit Hilfe von externen Quellen erzeugt wurden, sind Quellen in den Kommentaren innerhalb des Quellcodes genannt.

## 5.1 Cube Generierung

Nachdem verglichen wurde, wie andere Voxeleditoren die Cubes platzieren, fiel auf, dass die Cubes häufig an der Hand bzw. am Controller hängen. Dies hat den Nachteil, dass man sich häufig bücken muss um Cubes weiter unten zu platzieren oder häufig streckt um über andere Blöcke zu bauen. In diesem Projekt hängt der Cube zur Vorschau etwa 1-2 Meter entfernt vom Controller und bietet somit auch die Möglichkeit einen Block am Boden oder über andere Blöcke zu platzieren, ohne sich jeweils zu bücken oder zu strecken. Auf längerer Zeit gesehen ist dies Rücken- und Armschonend weil man aus einem festen Stand mit minimaler Bewegung

Blöcke vor sich platzieren kann. Derzeit entspricht jeder Platzierte Block in Unity einem GameObject und kann einzeln farblich beim Platzieren, sowie auch im Nachhinein verändert oder auch zerstört werden. Dies hatte den Vorteil dass sehr schnell und einfach ein erster Prototyp gebaut werden konnte (der auch modular erweiterbar ist), hat jedoch auch Konsequenzen, auf die im Benchmark eingegangen wird.

## 5.2 Benchmark

Im Verlaufe des Projekts wurde bemerkt, dass die maximale Anzahl von platzierbaren Objekten innerhalb des Editors relativ gering ist. Bei den Benchmark-Tests wurden so viele Blöcke erschaffen, bis es bemerkbare Leistungseinbrüche gab. Bei einem kompilierten Build lag dieser Wert bei etwa 3400 (Siehe Abbildung 5.1). Die Benchmarks wurden mit der Unity Version 2017.1.0f3 durchgeführt.

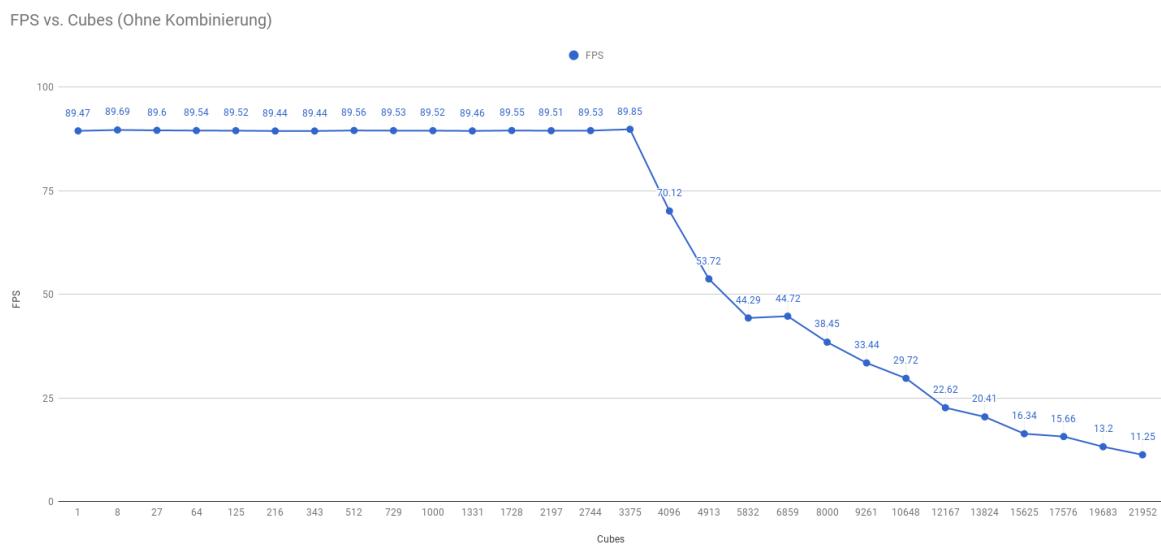


ABBILDUNG 5.1: Benchmark Graph 1 (FPS ohne Kombination)

Eine Recherche in der Unity Community bestätigte dieses Ergebnis, weil Unity anscheinend, je nach System, bei etwa 3000-8000 GameObjects Probleme hat, diese jede Sekunde (optimal bis zu 90 mal) neu zu berechnen. (vgl. [UnityAnswers, 2013](#)) Spezifischer liegt das Problem

beim Rendern vieler einzelner kleinen Objekte weil jedes Objekt einzelne Aufrufe auf der Grafikkarte verursacht. Wenn man alles zu einem Mesh kombiniert, können alle Objekte mit nur einem Aufruf gerendert werden.

Nachdem versucht wurde bei diesen Objekten die Kollisionsabfragen abzuschalten, hat sich die Performance nicht bemerkbar unterschieden. Erst nachdem alle Objekte zu einem großen Objekt (auch mit mehreren Kollisionsabfragen) und einem einzigen Mesh kombiniert wurden, hat sich die Performance signifikant verbessert (Siehe Abbildung 5.2 ).

FPS vs. Cubes (Compiled, mit Gruppierung)

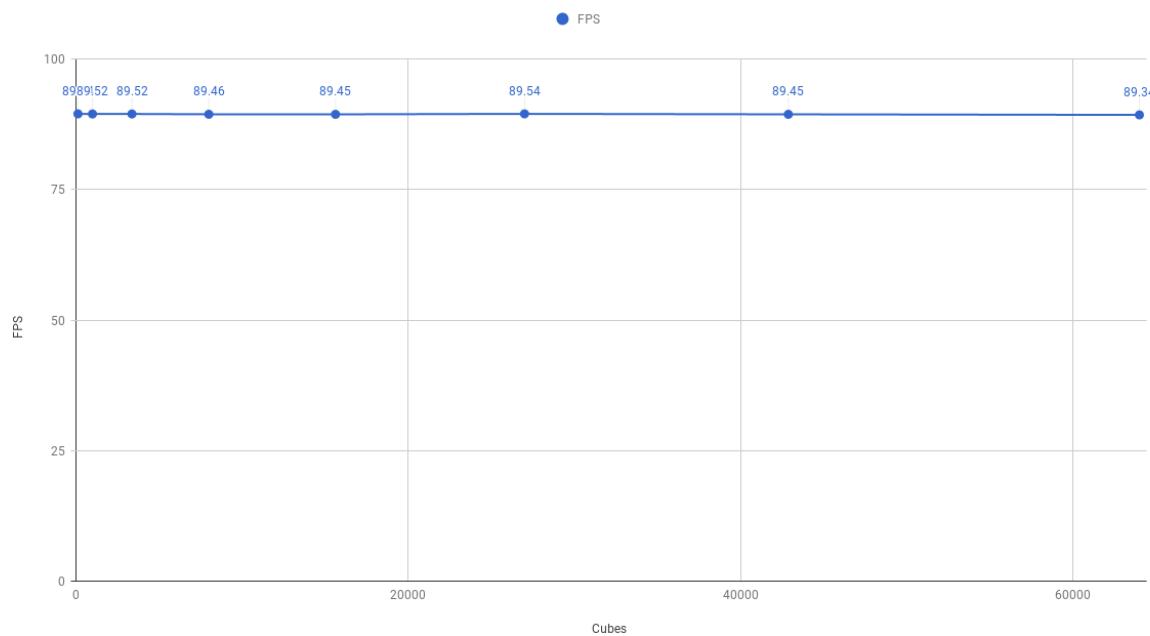


ABBILDUNG 5.2: Benchmark Graph 2 (FPS mit Kombination)

Die Benchmarks mussten früh abgebrochen werden, da das Erzeugen von den Blöcken irgendwann dazu geführt hat, dass das Programm eingefroren ist. Der Grund dafür wird in den ersten zwei Graphen sichtbar, denn bei den Tests wurden immer  $(5 * n)^3$  Würfel erzeugt, was dazu geführt hat dass bei späteren Aufrufen erst ein mal über 10.000 Blöcke gleichzeitig in der Szene existieren mussten bevor diese kombiniert wurden. Trotzdem sollte durch Abbildung 5.2 ersichtlich werden, dass sich selbst bei 60.000 Würfeln keine Performanceprobleme feststellen lassen wenn diese in Gruppen kombiniert werden, da die 60.000 Würfel kombiniert etwa 22 Gameobjects entsprechen.

Es können jedoch nicht unendlich viele Meshes kombiniert werden. Durch eine Fehlermeldung im Editor (Siehe Abbildung 5.3) wurde festgestellt, dass es ein Limit von maximal 65535 Verticies pro Mesh gibt, weil Unity die Verticies in einem 16-Bit Unsigned Integer speichert. Laut einer Tabelle mit den Datentypbereichen von Microsoft (Entwickler von C#) lässt sich dies nochmals bestätigen (vgl. [Microsoft, 2016](#))

```

UnityEngine.Mesh:CombineMeshes(CombineInstance[])
Assertion failed: Assertion failed on expression: 'count <= std::numeric_limits<UInt16>::max()'
UnityEngine.Mesh:CombineMeshes(CombineInstance[])
Assertion failed: Assertion failed on expression: 'count <= std::numeric_limits<UInt16>::max()'
UnityEngine.Mesh:CombineMeshes(CombineInstance[])
Assertion failed: Assertion failed on expression: 'count <= std::numeric_limits<UInt16>::max()'
UnityEngine.Mesh:CombineMeshes(CombineInstance[])
Combined 3375 meshes with 108000 verticies
UnityEngine.Debug:Log(Object)
Currently 3375 spawned blocks
UnityEngine.Debug:LogWarning(Object)

```

ABBILDUNG 5.3: Unityfehler: Zu viele Verticies

Durch diese Limitierung müssen also mehrere kombinierte Gruppen erstellt werden, wenn ein Mesh mehr als 65535 Verticies enthalten würde. Jeder Würfel in diesem Projekt enthält 24 Verticies, wodurch sich also ein Limit von etwa maximal 2730 Würfel pro Objekt ergibt. Es könnten theoretisch also  $3400 * 2730 = 9.282.000$  Würfel in der Welt existieren bis es signifikante Leistungseinbrüche entstehen.

Das Kombinieren von Würfeln hat jedoch den Nachteil, dass sich diese nicht mehr einzeln einfärben oder löschen lassen, sondern nur die ganze Gruppe. Deswegen wurde beschlossen eine Option anzubieten, die nur gleichfarbige Blöcke zu einer Gruppe zusammenfasst, um Leistungseinbrüchen entgegen zu wirken. Dies ist jedoch keine optimale Lösung und wird im späteren Verlauf der Dokumentation (Kritische Reflexion) diskutiert.

### 5.3 Funktionsoptimierung (FO)

Während der Entwicklungsphase wurde eine Funktion aus dem Konzept nach der anderen umgesetzt und daraufhin getestet. Die Tests waren notwendig um Programmier- und Usability-Fehler aufzuspüren, welche daraufhin ausgebessert und teilweise komplett behoben werden

konnten. Es folgt eine erneute Auflistung der Funktionen. In jedem Abschnitt wird erläutert welche Neuerungen und Anpassungen vorgenommen wurden. Diese spiegeln ergebnisorientiert den aktuellen Stand der Anwendung wieder.

### 5.3.1 FO: Cubes platzieren

Die Konzeption dieses Features besagte, dass die Vorschau eines zu platzierenden Cubes durch grüne Außenlinien des Cubes gezeigt werden soll. In Tests stellte sich allerdings heraus, dass die dünnen grünen Außenlinien in der Spielwelt nicht gut zu erkennen sind. Deshalb wird die Cube-Vorschau nun mit dem Cube selbst innerhalb der *Main Content-Zone* dargestellt, welcher vor dem Platzierung-Tool schwebt. Erst wenn der Nutzer diesen Cube mit der Trigger-Taste bestätigt, nimmt der Cube seinen Platz dort ein. Diese Darstellungsart hat den Vorteil, dass der Nutzer nun in der Vorschau auch schon die Farben des Cubes auswählen kann und diese sieht, bevor der Cube in der Welt platziert wird. In der folgenden Abbildung (5.4) sieht man die Vorschau eines vom Nutzer rot gefärbten Cube, bevor dieser in der Spielwelt platziert wird.

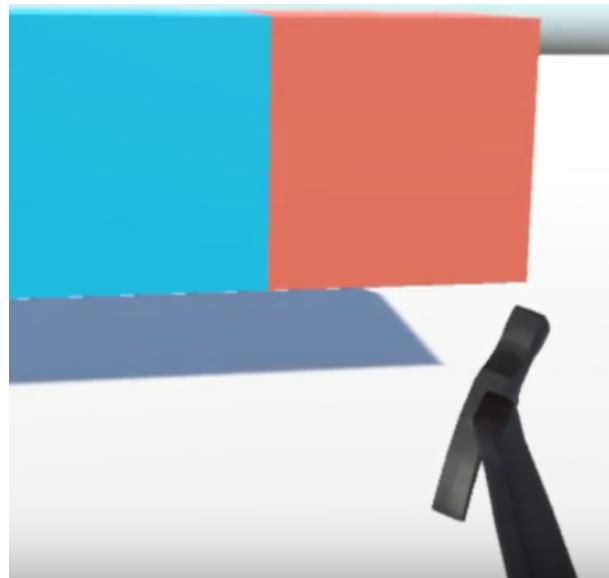


ABBILDUNG 5.4: Cube Vorschau

Eine weitere Neuerung ist, dass der Nutzer nun die Trigger-Taste gedrückt halten kann um stetig Cubes in der Welt zu platzieren. Dadurch ist ein sehr dynamisches und schnellen Erbauen möglich. Vor dieser Änderung musste pro Cube einmal die Trigger-Taste betätigt werden.

### 5.3.2 FO: Cubes entfernen

Cubes die mit dem Entfernung-Werkzeug selektiert werden, werden nicht mehr mit gelber, sondern roter Umrandung hervorgehoben. Die Farbe Rot suggeriert ein negatives Handeln wie abbrechen, abreißen und entfernen stärker als die Farbe Gelb und erhöht damit ebenso das Verständnis dieses Features. Hierbei kann der Nutzer ebenso die Trigger-Taste halten und muss sie für jeden zu entfernenden Cube nicht mehr wiederholt betätigen.

### 5.3.3 FO: Cubes einfärben

Ein neues Werkzeug symbolisiert die Funktion *Cube einfärben*. Der Nutzer kann neben dem Hammer und der Spitzhacke auch einen Pinsel auswählen. Dieser wird benötigt um den Anwendungsfall “Der Nutzer möchte einen Cube nach der Platzierung einfärben” abzudecken. Dafür muss der Nutzer lediglich den Pinsel in der Hand halten, eine Farbe auswählen, anschließend den gewünschten Cube selektieren und die Färbung mit der Trigger-Taste bestätigen. Ob der richtige Cube selektiert ist, erkennt der Nutzer an der roten Cube-Umrandung.

Auch die Farbauswahl wurde optimiert und findet nicht mehr in drei (Farbwert, Sättigung, Dunkelwert) sondern nur noch in zwei Schritten statt. Damit wurde der Aufwand des Einfärbens reduziert. Diese Reduktion konnte erreicht werden, indem der Farbwert und die Sättigung auf der Farbpalette kombiniert wurden. In der Abbildung 5.5 sieht man, dass sich in der Mitte des Touchpads die Farbe Weiß befindet.

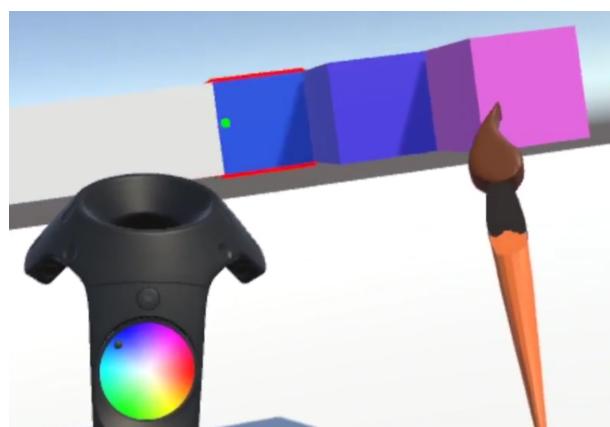


ABBILDUNG 5.5: Farbauswahl

Am Rand befinden sich die vollgesättigten Farben. Eine Mischung aus Farbwert und Sättigung kann also nun erreicht werden, indem eine Auswahl zwischen dem weißen Mittelpunkt und dem gesättigten Rand getroffen wird. Der Dunkelwert einer Farbe wird durch einen festen Druck auf das Touchpad ausgewählt. Wird das Touchpad nach oben gedrückt erhellt sich die Farbe. Wird es der unteren Hälfte entlang gedrückt verdunkelt sich die Farbe. In der Abbildung 5.6 sieht man die verdunkelte Farbpalette.

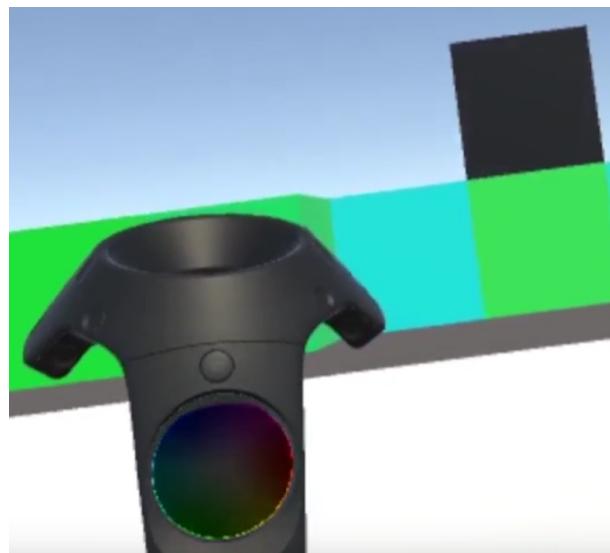


ABBILDUNG 5.6: Dunkelwert

Die Farbpalette ist immer auf der grafischen Anzeige des linken Controllers zu sehen. Neu hinzugefügt wurde ein Farbverlauf, der die letzten sechs ausgewählten Farben speichert. Zu sehen ist dieses Feature in der Abbildung 5.7. Damit wurde ein Erinnerungsprotokoll für Farben geschaffen, welches dabei hilft die gleichen Farben ein erneutes Mal auszuwählen. Denn es zeigte sich in Tests als schwierig mit dem empfindlichen Touchpad die exakt gleiche Farbe ein zweites oder mehrere Male erneut auszuwählen. Um den Farbverlauf zu öffnen, muss der Pinsel in der rechten Hand getragen werden und der linke Grip-Button gedrückt werden. Daraufhin erscheinen die letzten Farben in sechs Teilen kreisrund um der Farbpalette. Die Farbverlauf-Anzeige befindet sich damit in der *Touch Interface Zone*. Eine Farbe aus dem Farbverlauf wird anschließend ausgewählt, in dem der Nutzer mit dem linken Daumen in die Richtung der gewünschten Farbe wischt. Das selektierte Farb-Sechstel leuchtet zur besseren Erkennung leicht auf, wenn es ausgewählt wird.



ABBILDUNG 5.7: Verlauf der Farbauswahl

#### 5.3.4 FO: Teleportation

Die grafische Anzeige der Teleportation-Funktion stellt keinen Lichtbogen mehr dar. Ebenfalls wird das Ziel der Teleportation nun durch einen Pointer angezeigt. Beim Testen der Funktion fiel auf, dass die Weite des Lichtbogens nicht verändert werden kann und damit ungeeignet war sehr hohe oder sehr weite, aber auch äußerst nahe Flächen zu betreten. Stattdessen kann nun ein Pointer wie bei der Pointer-Selektion genutzt werden um Flächen zu selektieren. Wird die Auswahl bestätigt, wird der Nutzer auf(!) die Fläche des selektierten Objekts platziert. Damit gehen die Vorteile eines Lichtbogen (wie im Konzept erklärt) nicht verloren. Erste Überlegungen während Tests der Funktion waren, dem Nutzer eine Möglichkeit zu geben die Weite des Lichtbogens anzupassen. Dafür konnte allerdings keine aufwandslose und effiziente Tastensteuerung gefunden werden. Deshalb entschied man sich erneut für die grenzenlose Pointer-Lösung. Abbildung 5.8 zeigt den Teleportation-Pointer aus dem linken Controller.

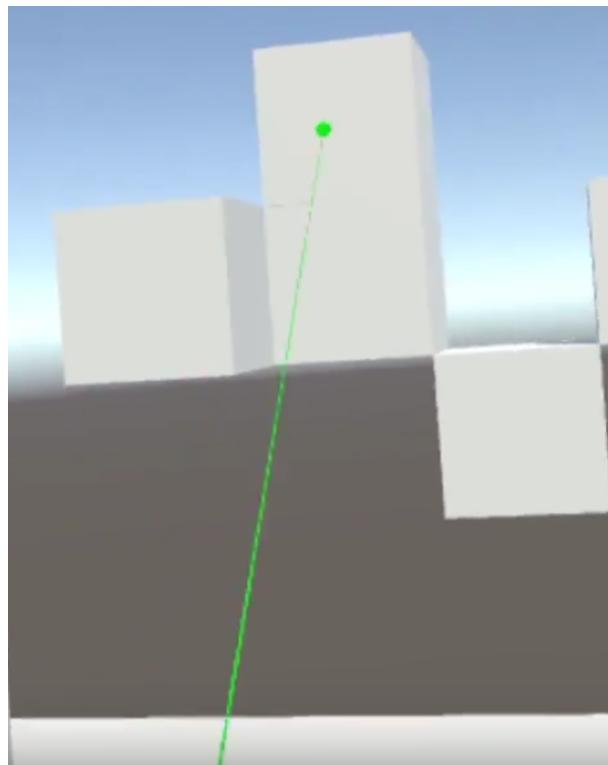


ABBILDUNG 5.8: Teleportation - Pointer

### 5.3.5 FO: Analoge Controllersteuerung

Die Bewegung zur Detailarbeit über eine analoge Controllersteuerung kann fortan nur noch über zwei Achsen (X- und Y-Achse) ausgeführt werden. Grund für die Entnahme der Z-Achsen-Bewegung sind die Ergebnisse aus Tests, die zeigten, dass die Teleportation als Z-Achsen-Bewegung ausreicht. Der Nutzer kann nun über ein leichtes Streichen mit dem Daumen über das rechte Touchpad seinen Charakter über die X-Achse (Bewegung nach rechts oder links) und über die Y-Achse (Bewegung nach oben oder unten) steuern. Diese Steuerungsmöglichkeit ist ein weiterer Grund die analoge Bewegung über die Z-Achse zu entfernen, da das Touchpad keine weitere Lösung bietet eine effiziente Steuerungsmöglichkeit über die Z-Achse zu ermöglichen.

### 5.3.6 Werkzeugauswahl

Eine grafische Übersicht zeigt die komplette Auswahl der Werkzeuge an, aus der sich der Nutzer eines nehmen kann. Zuvor konnte der Nutzer zwischen den Werkzeugen auswählen, indem diese durch Drücken der rechten Grip-Taste rotierend ausgewechselt wurden. Betätigt der Nutzer nun die Grip-Taste erscheinen alle Werkzeuge oberhalb des rechten Controllers. Sie erscheinen innerhalb der *Touch Interface Zone* über dem Controller, da davon ausgegangen wird, dass der Nutzer seine Hände meistens entspannt auf Hüfthöhe hält. Erscheinen sie oberhalb der Steuerhand, fallen sie sofort auf und können zügig ausgewählt werden. Der Nutzer kann sich daraufhin ein Werkzeug auswählen, indem er den Controller dorthin führt. Hat der Controller den Platz des Werkzeuges eingenommen, nimmt der Controller das Erscheinungsbild des Werkzeuges an und die dazugehörigen Funktionen werden aktiv. Durch diese Anzeige- und Auswahllösung kann der Nutzer am effizientesten ein neues Werkzeug auswählen. Während die Rotation der Werkzeuge sehr lange dauerte, weil jedes Werkzeug übersprungen werden musste, bis das gewünschte erreicht wurde, kann mit der jetzigen Anzeige jedes beliebige Werkzeug schnell ausgewählt werden. Des Weiteres bleibt Raum um beliebig viele neue Funktionen und Werkzeuge in die Anwendung einzufügen.

Bei Tests der Werkzeugauswahl wurde deutlich, dass Novizen und neue Nutzer anders mit diesem Feature umgehen. Während sich Anfänger neugierig ausprobieren und sich nicht direkt merken, wo sich welches Werkzeug befindet, arbeiten Novizen sehr schnell mit der Werkzeugauswahl. Sie müssen sich nicht erst orientieren, sondern wissen, dass sich z.B. der Hammer rechts befindet und wählen diesen in einer gekonnten Bewegung aus, ohne dessen Platzierung überprüfen zu müssen. Die folgende Abbildung stellt das Feature vor.



ABBILDUNG 5.9: Werkzeugauswahl

### 5.3.7 Options-Menü

Optionen können in einem separaten User-Interface manuell eingestellt werden und müssen nicht mehr durch Tastenkombinationen getoggelt werden.

Um das Menü zu öffnen, muss der Nutzer den linken Menü-Knopf drücken. Daraufhin erscheint die grafische Anzeige des Menüs vor dem linken Controller und wird an diesen angehaftet, sodass sich dieses ebenfalls über Handbewegungen neigen und schwenken lässt. Damit befindet sich die GUI in der *Touch Interface Zone*. Mit dem rechten Controller kann der Nutzer die gewünschte Optionen auswählen.



ABBILDUNG 5.10: Options-Menü

Die Abbildung 5.10 zeigt welche Optionen in dem Menü umgestellt werden können. Die Einstellung *Gitter* schaltet das Mesh-Gitter der Spielwelt ein und aus. Ist es ausgeschaltet kann der Nutzer Cubes freihand platzieren. Die Checkbox *Werkzeug Pointer* wechselt die Selektionssteuerung zwischen der Selektion durch Kopfbewegung und der Selektion durch einen Pointer in der Hand. Unter diesen zwei Möglichkeiten findet der Nutzer die Skalierungsstufen der Cubes. Die Stufe bestimmt die Größe der danach platzierten Cubes, bis der Nutzer sich für eine andere Skalierung entscheidet und die Größe für weitere Cubes anpasst. Der letzte Menüpunkt *Blöcke zusammenfassen* erstellt aus gleichfarbigen Cubes eine Kombination. Dies ist eine Designentscheidung die zu Gunsten höherer Performance hinzugefügt wurde. Mehr dazu kann im Kapitel *Benchmark* und *Kritische Reflexion* nachgelesen werden.

### 5.3.8 Selektionssteuerung

Es gibt zwei Möglichkeiten die eigene Selektion zu steuern. Eine Möglichkeit ist, den Bildmittelpunkt als Selektion zu fixieren. Um damit Cubes zu selektieren muss der Nutzer seinen Kopf bewegen und den Bildmittelpunkt auf die gewünschte Stelle legen. Die andere Möglichkeit ist, einen Laserpointer aus dem Werkzeug in der rechten Hand projizieren zu lassen um damit händisch Objekte anzuleuchten um sie zu selektieren. Beide Methoden zeigen Vor- und Nachteile und arbeiten ähnlich indem das Programm permanent berechnet mit welcher Fläche der Selektions-Punkt kollidiert (Kollision mit Boden oder Cube?). Eine Selektion durch Kopfbewegung mit Hilfe des Bildmittelpunktes ist praktisch für Detailarbeit, da der Kopf keine zittrige Position wie die Hand einnimmt und man sehr nahe und kleine Cubes so leichter selektieren kann. Doch für Objekte auf mittlerer und weiter Distanz ist die Pointer-Steuerung weit besser geeignet. Für solche Fälle müsste der Kopf sonst stetig hin und her bewegt werden, was die Nackenmuskulatur schnell ermüden lässt und zu Verletzungen führen kann. Während der Umsetzungsphase konnte leider keine empirische Studie durchgeführt werden, um beide Selektionsmöglichkeiten objektiv zu vergleichen. Aus diesem Grund sind beide Steuerungsmethoden in der Anwendung enthalten.

Im Options-Menü kann die Selektion umgeschaltet werden. Von diesen Selektions-Modi betroffen sind die Werkzeuge Pinsel und Spitzhacke. Der Hammer ist davon nicht betroffen. Hierbei ist eine händische Handhabung notwendig um die gewünschte Distanz zu messen, auf der ein Nutzer seinen Cube platzieren möchte.

Abbildung 5.11a zeigt die Selektion durch Kopfbewegung mit Hilfe des Bildmittelpunktes. Zu sehen ist dabei ein kleiner grüner Punkt in der Bildmitte.

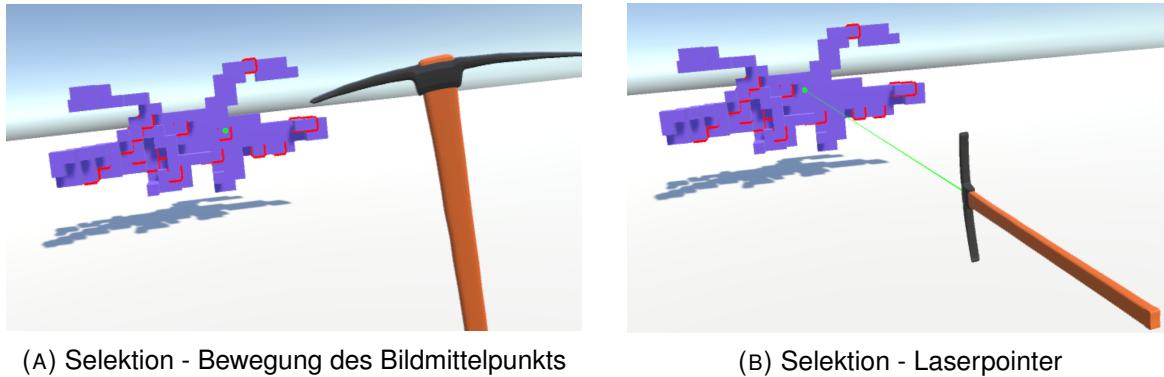


ABBILDUNG 5.11: Selektionssteuerung im Vergleich

Abbildung 5.11b zeigt die Selektion durch die Pointer-Steuerung. Aus dem Werkzeug Spitzhake leuchtet ein kleiner grüner Pointer, welcher durch Handbewegung verändert werden kann.

### 5.3.9 Cube Skalierbarkeit

Eine neue Funktion ist die Größenanpassung der Cubes. Mit unterschiedlichen Skalierungen der Cubes soll der Nutzer eine vielfältige und detailreiche Möglichkeit erhalten seine Fantasie-Objekte umzusetzen. Die Cube-Größe wird vor seiner Platzierung verändert. Dafür stehen vier unterschiedliche Größe-Stufen zur Verfügung. Im Options-Menü kann sich der Nutzer für eine Skalierungsstufe entscheiden und diese auswählen. Die Größe der Cubes kann jeder Zeit nach Belieben gewechselt werden. Zur Verfügung stehen die Größen:  $x1$ ,  $x0.5$ ,  $x0.25$  und  $x0.125$ . In der Abbildung 5.12 werden die vier Cube-Größen vergleichsweise dargestellt.

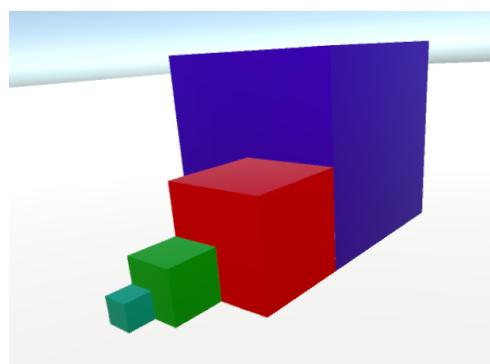


ABBILDUNG 5.12: Skalierungsstufen

### 5.3.10 Cube Kombination

Die Kombination gleichfarbiger Cubes ist eine Lösungsmethode der Performanceprobleme, welche im Kapitel Benchmark besprochen wurden. Während durchgeführter Leistungstests der Anwendung wurden Performanceschwächen festgestellt. Dies erschwerte die Anforderung eine VR-Anwendung mit konstanten 90 FPS zu schaffen. Um dieses Problem zu lösen wurde eine Methode hinzugefügt, die der Nutzer manuell ausführen kann, sobald Leistungseinbrüche spürbar werden. Durch das Ausführen dieser Methode werden alle gleichfarbigen Cubes zu einem GameObject kombiniert. Dadurch wird die totale Anzahl von GameObjects in der Szene verringert, wodurch die Laufzeitumgebung wieder leistungsvoller arbeiten kann.

Um die Cube-Kombination durchzuführen muss der Nutzer mit dem rechten Menü-Button das Options-Menü öffnen und die grüne Schaltfläche am Boden des Menüs bestätigen.

# 6 Diskussion

## 6.1 Kritische Reflexion & Erkenntnisse

Die kritische Selbstreflexion soll dabei helfen den Ablauf des Projektes zu bewerten und Schwierigkeiten zu beurteilen, um damit Erkenntnisse gewinnen zu können, die zukünftigen Projekten und Weiterentwicklungen einen positiven Mehrwert bringen. Zunächst wird der Projektlauf kurz zusammengefasst. Daraufhin werden konkrete Probleme während des Projekts hierarchisch nach negativem Einfluss aufgelistet und beschrieben welche Erkenntnisse daraus geschlossen werden.

### 6.1.1 Ablauf

Obwohl die Arbeit an dem Projekt stetig Fortschritte verzeichnen konnte, wurde der Meilensteinplan zwischen den Audits mehrmals angepasst. Während der erste Plan noch viel freien Zeitpuffer zu den letzten zwei Audits auf dem Terminplan prophezeite, füllte sich der Meilensteinplan bis zum sechsten Audit mit dutzenden neuen Ideen und Umsetzungsvorhaben, sowie Tests der Funktionen und Bugfixes. Zum einen wurde die notwendige Umsetzungszeit der einzelnen Funktionen unterschätzt, sodass es wenige Funktionen nicht mehr in den Prototyp zum Ende des Praxisprojekts geschafft haben. Zum anderen beanspruchte das Projekt sehr viel Recherchearbeit und Zeit, die während Tests und Iterationsschritten verging.

Der erste Meilenstein zum 8. November 2017 bestand in Fertigstellung der Recherche in der Domäne und VR-Technik. Darauf aufbauend wurden die ersten Ideen aufgestellt.

Der zweite Meilenstein wurde am 22. November erreicht, als die Planung der Funktionen mit dazugehöriger Usability-Umsetzungen konzeptioniert wurden. Der Schwerpunkt lag dabei auf der Überlegung von Notwendigkeiten bestimmter Funktionen in einem VR-Editor. Dafür musste sich unter anderem ausgiebig mit der Controllersteuerung der HTC Vive auseinander gesetzt werden.

Mit dem dritten Meilenstein zum 6. Dezember begann die Umsetzung des Konzepts. Nach dem Motto “First make it work, then make it nice” und “separate and conquer” wurde die lange Konzeptionsliste mit einer Funktion nach der anderen umgesetzt, getestet und iterativ überarbeitet.

Der vierte Meilenstein zum 20. Dezember konnte erreicht werden, nachdem die schwierigsten Bugfixes behoben werden konnten. Somit konnte am 10. Januar 2018 der fünfte Meilenstein erreicht werden, nachdem die Umsetzung erweiterter Funktionen begonnen werden konnte. Dazu zählen alle Funktionen, die außerhalb des anfänglichen Konzepts hinzugefügt wurden.

Zwischen dem vierten und fünften Meilenstein wurden Benchmarktests ausgeführt. Diese führten dazu, dass eine Methode gefunden werden musste, um die Performance der Anwendung stabil zu halten. Durch diese nicht eingeplante Erkenntnis verschob sich der Plan zum sechsten Meilenstein eine optimierte grafische Anzeige zu implementieren und die restlichen Funktionen abzuschließen. Daher konzentrierte sich die restliche Projektzeit auf die Problembehebung bestehender Herausforderungen wie der Performanceoptimierung und Bugfixes.

Der sechste Meilenstein wurde am 31. Januar mit dem ersten Prototypen erreicht, welcher zu Vorführungszwecken während der Abschlusspräsentation genutzt wurde.

### **6.1.2 Fehlende durch Programmcode selbst definierte 3D-Primitive**

Um dem Projekt einen schnellen Start und eine zügige Durchführung zu gewähren, wurde zu Beginn darauf verzichtet die Cubes als 3D-Primitive selbst zu zeichnen. Stattdessen wurden die vorgefertigten Primitiven von Unity genutzt. Ein Vorteil dieser Vorgehensart ist, die Zuverlässigkeit Seitens der Entwicklungsumgebung keine Programmierfehler oder Laufzeitfehler hervorzurufen. Die vordefinierten Cubes können einer Spielwelt schnell hinzugefügt und manipuliert werden. Der Nachteil ist hauptsächlich die Performanceschwäche, die durch die Art

und Weise der Renderingprozesse und Berechnungen hervorgerufen wird. Durch diese Entscheidung können nur eine geringe Anzahl (etwa 3000) Blöcke (Objekte) platziert werden, bis Performanceeinbrüche spürbar sind.

Diese Erkenntnis wurde gegen Ende der Umsetzung leider erst so spät gewonnen, dass eine Anpassung im Zeitrahmen des Praxisprojekts nicht mehr vorgenommen werden konnte. Das richtige Vorgehen sähe so aus, dass die Vertices, Kanten und Flächen der 3D-Primitive zunächst durch Programmlogik erstellt werden müssten, damit deren Datenstruktur jeweils verwaltet werden könnten. Vier Vertices bilden aus zwei gegen den Uhrzeigersinn erstellten Dreiecken eine Fläche eines Cubes. Diese Flächen werden gemapped und erhalten eine Textur. Sind alle Flächen erstellt und zu einem Cube als GameObject zusammengefasst, muss dieser noch gerendert werden. Mit dieser Methode könnten dann auch nur die Flächen gerendert werden, die auch tatsächlich sichtbar sind.

Als alternative Lösung zu diesem Problem wurde die Funktion Cube-Kombination eingeführt, die viele einzelne Objekte zu einem zusammenfasst, um so die Berechnung der Meshes zu optimieren und die Performance zu erhöhen.

### 6.1.3 Fehlende Cube-Auflösung nach Cube-Kombination

Die Funktion der Cube-Kombination (nachzulesen in den Kapitel “Benchmark” und “Funktionsoptimierung”) wurde außerplanmäßig gegen Ende des Praxisprojekts hinzugefügt, um die festgestellten Performanceschwächen ab einer bestimmten Cube-Anzahl zu korrigieren.

Die Kombination gleichfarbiger Cubes zu einem GameObject zeigt in der Laufzeitumgebung Unity große Stärken, um die Leistung der Anwendung anzuheben und eine stabile Performance bei 90 FPS zu halten. Allerdings nimmt der Nutzer dabei den Nachteil auf sich diese gleichfarbigen Cubes nicht mehr separat bearbeiten zu können. Kombinierte Cubes können nach der Kombination lediglich als eine Einheit bearbeitet (entfernt oder eingefärbt) werden. Es fehlt zum Stand des Prototypen eine Funktion um Cube-Kombinationen wieder aufzulösen. Grund für das Fehlen ist der Zeitmangel an dem Projekt bis zu Abgabe weiterarbeiten zu können in Betrachtung der Komplexität einer solchen Funktion. Jedoch besitzt Voxel VR, durch leichte Erweiterbarkeit, den Vorteil auch nach dem Praxisprojekt modular weiterentwickelt zu werden (zum Beispiel für eine Bachelorarbeit).

### **6.1.4 Spät durchgeführte Benchmarks**

Ein während der Hälfte des Projekts durchgeföhrter Benchmark machte deutlich, dass mehr Aufwand in die Performanceoptimierung der Anwendung fließen muss. Diese verhältnismäßig späte Erkenntnis destrukturierte den Meilensteinplan und sorgte dafür, dass Zeitverlust akzeptiert werden musste.

Dieses Problem kann verhindert werden, wenn Benchmarks öfter und früher während der Umsetzungsphase durchgeführt werden. Es sollten mindestens drei Benchmarks schon während der Konzeption eingeplant werden und zu Beginn, während und zum Ende des Projekts durchgeführt werden. Dabei sollten die gesammelten Informationen analysiert und verglichen werden, um früher einschätzen zu können welche Ressourcen benötigt werden.

### **6.1.5 Zeitverlust durch spontane Implementierung**

Zwei ursprünglich geplante Funktionen konnten bis zum Ende des Praxisprojekts nicht mehr in die Anwendung implementiert werden - "Cubes gruppieren" und "Objekte manipulieren". Dafür wurden während der Umsetzung und Iteration der ersten Funktionen bemerkt, dass neue und zuerst ungeplante Funktionen eingebaut werden müssen, da diese wichtiger erscheinen und deshalb kurzfristig implementiert wurden. Diese sind zum Beispiel: Das Options-Menü, die Selektionssteuerung, die Skalierbarkeit der Cubes und die Werkzeugauswahl.

Das Problem war also, dass durch spontane Implementierung ungeplanter Funktionen Zeit verloren ging das ursprüngliche Konzept zu verfolgen. Deshalb mussten wegen Zeitmangels zwei Funktionen ausgelassen werden - die Gruppierung & Manipulation der Cubes (siehe Konzept). Die Erkenntnis aus dieser Schwierigkeit ist, dass mehr Puffer-Zeit im Meilensteinplan verzeichnet sein muss. Diese Zeiten müssen verwendet werden, um zum einen Fehlerkorrekturen ausgiebig durchführen zu können und gleichzeitig noch Zeit einzuplanen, spontane Ideen dem Konzept hinzuzufügen.

### 6.1.6 Unterschätzter Zeitaufwand der Fehlerkorrekturen

Viele Bugs und Logikfehler des Programmcodes wurden erst nach langen Tests der Anwendung gefunden. Aus diesem Grund mussten immer wieder Zeitabschnitte zwischen der Weiterentwicklung des Projekts für Fehlerkorrekturen eingesetzt werden.

Um diesen Fehler zu vermeiden sollte der Meilensteinplan eines Softwareprojekts mehr Zeitpuffer für Bugfixes einplanen und dafür die Anzahl der Gesamtfunktionen etwas verringern.

### 6.1.7 Umsetzungspunkte der VR-Ergonomie

Mit einer konstanten Bildwiederholrate von 90 FPS (dank der Performance-Verbesserung), einem Sichtradius von ca. 112° und der Limitierung der analogen Controllersteuerung auf zwei Translationsachsen, wird der Motion Sickness Einhalt geboten. Ein Sichtradius von 94° Grad, wie von Alger empfohlen (siehe Kapitel 3.1 Motionsickness & Gegenmaßnahmen), wird mit den von uns genutzten 112° leicht überboten, da dies die Standardsichtweite der HTC Vive ist. Auf fixe Bezugspunkte wie einem Cockpit oder separate Gegenstände wird verzichtet, da während der Durchführung keine sinnvolle Verwendung der Bezugspunkte gefunden werden konnte. Die Kamera wird ausschließlich vom Nutzer durch das HMD kontrolliert und wird nicht unnötig animiert. Der Augenabstand muss vom Nutzer vor Benutzung der Anwendung eingestellt werden.

Grafische Darstellungen wie das Interface des Options-Menüs, Werkzeuge und das Platzieren der Cubes bleiben in der Touch Interface- und Main Content-Zone. Der Nutzer muss seinen Kopf dafür nicht aus den bequemen Winkeln hinaus neigen. Es werden keine weiteren Objekte im Raum platziert. Aus diesem Grund finden sich auch keine weiteren Medien in der Anwendung wieder, die eine spezielle Nutzungsdistanz benötigen würden. Das Options-Menü wird vom Nutzer über dem linken Controller gehalten und lässt sich ähnlich wie ein Touchpad mit dem Finger bedienen. Die Schriftgröße des Menüs ist deutlich lesbar und fällt in die Kenngrößen der UI-Standards (siehe Kapitel 3.3.2).

## 6.2 Offene Fragen

Während der Projektarbeit kamen Fragen auf, die zusätzliche Recherchearbeit oder Arbeitsaufwand in Form von Tests und Studien benötigen, um beantwortet werden zu können. Diese offenen Punkte sollten untersucht werden, bevor das Projekt im Rahmen einer Bachelorarbeit oder ähnlicher Ausarbeitung fortgeführt wird. Im folgenden Abschnitt sollen diese Fragen aufgelistet werden.

### 6.2.1 Welche Selektionssteuerung führt zu besserer Usability?

In VoxelVR kann der Nutzer entscheiden durch welche Art und Weise er Objekte selektiert. Zur Auswahl stehen eine Selektion durch Kopfbewegung bei der alles selektiert wird was sind in der Bildmitte befindet und eine Selektion durch Pointer bei der der Nutzer eine Art Laserpointer in der Hand hält um mit diesem Objekte zu selektieren.

Eine Selektion die an das HMD gebunden ist (Kopfbewegung), ist sehr präzise und eignet sich sehr gut für feine Detailarbeit bei nahen Objekten. Während eine handgesteuerte Selektion zitternde Bewegungen hervorrufen kann, bleibt eine Kopfsteuerung stabil. Außerdem garantiert diese auch, dass der Nutzer die selektierten Objekte immer im Sichtfeld behält, da die Bildmitte das Selektionswerkzeug ist. Jedoch ermüdet die Nackenmuskulatur nach einiger Zeit spürbar, da die HTC Vive immerhin ein Gewicht von 600g auf die Waage bringt. Dementsprechend könnte der Nutzer bei anhaltender Verwendung dieser Selektionssteuerung Verletzungen erleiden.

In persönlichen Tests fiel die Entscheidung allerdings auf die Selektion durch das Pointerwerkzeug. Dieses garantiert eine schnelle und dynamische Bearbeitung und ist sehr gut auf mittelweite bis weite Objekte anwendbar. Die Verwendung eines Pointers ähnelt einem Fingerzeig auf Objekte.

Alles in allem sind die genannten Selektionssteuerungen sehr praktisch und bieten jeweils Vor- und Nachteile, weshalb VoxelVR auch beide anbietet. Letztendlich soll der Nutzer die Möglichkeit erhalten seinen VR-Arbeitsplatz selbst gestalten zu können. Ebenfalls sollen Nutzer mit

körperlichen Einschränkungen, die aus speziellen Gründen eine der Selektionssteuerungen nicht benutzen können, nicht ausgeschlossen werden.

### **6.2.2 Durch welche Optimierungsmethoden kann die Performance noch weiter verbessert werden?**

Der Flaschenhals der Performance lag nach den Benchmarks bei der zu hohen Anzahl der GameObjects. Durch die Methode der Cube-Kombination konnte diese Anzahl extrem verringert und die Performance angehoben werden. Jedoch stellt sich die Frage welche weiteren Prozesse in den Tiefen der Laufzeitumgebung zu Leistungsverlusten führen. Untersucht werden können dazu noch weitere Ebenen. Dazu zählen die Renderingprozesse der 3D-Primitiven, sowie der Licht- und Schattenberechnung. Ebenso könnte eine verbesserte Verwaltung der Datenstrukturen der Programmlogik die Performance verbessern. Dazu zählt zum Beispiel die Methode der Baum-Datenstruktur “Octree”.

## **6.3 Themenrelevanz & Ausblick**

Dieses Praxisprojekt ermöglichte viele neue Blicke in das Genre der Virtual Reality, welche im Studiengang der Medieninformatik nur wenig betrachtet werden. So fanden Exkurse in die Entwicklung, Ergonomie, Psychologie, in das Usability-Engineering und in die Philosophie von Virtual Reality statt. Diese neue Technologie steht immer noch an ihren Anfängen und wird durch wissbegierige Pioniere der Szene unfassbar schnell erforscht und weiterentwickelt. Jede noch so kleine Erkenntnis könnte einen Denkanstoß auslösen, der bisherige Fortschritte ablöst und einen produktiven neuen Ansatz in der virtuellen Realität liefert. So ist es wünschenswert, dass dieses Praxisprojekt für zukünftige Studenten als Inspiration, Vorlage und Quelle dient, ein weiteres VR-Projekt durchzuführen oder VoxelVR weiterzuentwickeln.

## **7 Fazit**

Das Praxisprojekt des Virtual Reality Voxel Editors “Voxel VR” konnte erfolgreich mit einem vorführungsfertigen Prototypen abgeschlossen werden. Bis zu diesem Zeitpunkt konnten die Meilensteine zufriedenstellend erreicht werden und zu jedem Audit-Termin neue Ergebnisse präsentiert und diskutiert werden. Zwischenergebnisse konnten ebenfalls mit dem Betreuer besprochen werden um neue Denkanstöße in die Arbeit einfließen zu lassen.

Das Ziel einen VR Voxel Editor zu erstellen konnte mit Abschluss dieses Praxisprojekt erreicht werden. Dieser wurde nach durchgeführter Domänenrecherche von Grund auf erschaffen und hat sich dabei stetig weiterentwickelt. Damit wird eine Anwendung als Ergebnis ausgeliefert die sich im Bereich der Virtual Reality Unterhaltung und Kreativitätsförderung befindet. Voxel-VR bietet dabei eine simple und selbsterklärende Bedienung für Nutzer die sich mit virtuellen Realität bereits vertraut machen konnten. Wie geplant kommt das Programm ohne überfüllte User Interfaces aus und ist auch in der Zukunft leicht mit neuen Funktionen erweiterbar, in dem die Werkzeugauswahl um weitere Werkzeuge ergänzt wird, welche sich an das gleiche Prinzip halten, das auch in den hier verwendeten Grundfunktionen benutzt wurde (Neue Tastenbelegungen pro Funktion). Das Konzept diente während der Umsetzung als Orientierung. Chronologisch wurde eine Funktion nach der anderen umgesetzt und daraufhin getestet. Alle Schwierigkeiten und neuen Erkenntnisse konnten während der Tests dokumentiert werden. Dies war notwendig um Umsetzungsprozesse iterativ durchführen zu können. So wurden Funktionen immer wieder verbessert und die Notwendigkeit von Anpassungen deutlich. Mit jedem Test wurde die komplette Anwendung ausgeführt und immer wieder an ihre Grenzen gebracht.

Dieses Praxisprojekt ermöglichte es Einblicke in die Entwicklung von VR-Anwendungen zu erlangen. Nach intensiver Vorarbeit in Form von Recherche und online Kursen war es bereits möglich ein eigenes Konzept für ein VR-Programm zu entwickeln. Auch, wenn Virtual Reality

stetig Fortschritte verzeichnet und noch nicht für alles Regeln und Standards existieren, kann sich jeder Entwickler mit Interesse in dem Gebiet einen Einstieg in dieser Domäne finden. Wichtig dabei ist stets über Neuerungen informiert zu bleiben.

VR ist eine aufstrebende Erfindung und wird vermutlich die Zukunft in vielerlei Anwendungen darstellen. Trotz dieser Zukunftsaussicht steckt VR und die dazugehörige Hardware noch in den Kinderschuhen und muss wissenschaftlich geprüft (kurzfristige und langfristige Konsequenzen) und technisch weiterentwickelt werden. In dieser Arbeit durften wir als Studenten in die erstaunliche Welt der VR und Voxelgrafik eindringen und einen Blick in die Zukunft werfen.

# Literaturverzeichnis

- 360°-Know-How (2017). Was ist virtual reality? <https://omnia360.de/blog/was-ist-virtual-reality/>. [Online Abgerufen am 26.09.2017].
- Alger, M. (2015). Visual design methods for virtual reality. <https://drive.google.com/file/d/0B1917cJ7tVJyRkpUM0hVYmxJQ0k/view>. [Online Abgerufen am 13.11.2017].
- Chu, A. (2014). Vr design: Transitioning from a 2d to 3d design paradigm. <http://voicesofvr.com/117-alex-chu-on-designing-milk-vr-for-the-samsung-gear-vr/>. [Online Abgerufen am 13.11.2017].
- Dannenberg, B. (2017). Motion sickness in vr: Wie es entsteht und was ihr dagegen tun könnt. <https://vr-world.com/was-tun-bei-motion-sickness-in-vr/>. [Online Abgerufen am 13.11.2017].
- Dudenverlag (1988). Duden informatik ein sachlexikon für studium und praxis (german edition). <http://amazon.com/o/ASIN/3411024216/>. [Online Abgerufen am 12.11.2017].
- Google I/O 16, D. (2016). Daydream labs: Lessons learned from vr prototyping (google i/o 2016). <https://www.youtube.com/watch?v=lGUmTQgbiAY>. [Online Abgerufen am 15.11.2017].
- Google I/O 17, D. (2017). Designing screen interfaces for vr (google i/o '17). <https://www.youtube.com/watch?v=ES9jArHRFHQ>. [Online Abgerufen am 14.11.2017].
- Microsoft (2016). Datentypbereiche). <https://msdn.microsoft.com/de-de/library/s3f49ktz.aspx>. [Online Abgerufen am 20.01.2018].

- Patrao, B., Pedro, S., and Menezes, P. (2017). How to deal with motion sickness in virtual reality. <http://scitecin.isr.uc.pt/Proceedings/Papers/EPCGI/17.pdf>. [Online Abgerufen am 12.11.2017].
- Sachs, S. (2016). Voxel art: Reducing the greebles. <https://blog.sketchfab.com/voxel-art-reducing-greebles/>. [Online Abgerufen am 18.09.2017].
- Sahin, B. (2015). *Spieleentwicklung mit Unity: 2D- und 3D-Games für Desktop, Web & Mobile*. BookRix.
- Schilling, A. (2016). Htc vive in der praxis: Aufbau und funktionsweise. <http://www.hardwareluxx.de/index.php/artikel/consumer-electronics/gadgets/38985-htc-vive-in-der-praxis-teil-1-aufbau-und-funktionsweise.html>. [Online Abgerufen am 08.10.2017].
- Schäfer, B. (2017). Fünf spannende anwendungsbereiche von virtual reality. <https://vr-world.com/virtual-reality-und-wo-sie-zu finden-ist/>. [Online Abgerufen am 12.11.2017].
- Unity3D (2017). Unity3d infos. <https://unity3d.com/de/public-relations>. [Online Abgerufen am 20.09.2017].
- UnityAnswers (2013). Max number of gameobjects (5000?) [online forum kommentar]). <https://answers.unity.com/questions/408298/max-number-of-gameobjects-5000-1.html>. [Online Abgerufen am 20.01.2018].
- von IDG, C. (1991). Voxel statt pixel: Aktuelle methoden der 3d-darstellung. <https://www.computerwoche.de/a/voxel-statt-pixel-aktuelle-methoden-der-3d-darstellung,1140210>. [Online Abgerufen am 23.09.2017].
- VRScout, Tambovtsev D., F. N. P. O. (2016). How to avoid the effect of motion sickness in vr. <https://vrscout.com/news/avoid-motion-sickness-developing-for-vr/#>. [Online Abgerufen am 12.11.2017].

## **A Eidesstattliche Erklärung**

Wir versichern, die von uns vorgelegte Arbeit selbstständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, haben wir als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die wir für die Arbeit genutzt haben, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 31. Januar 2018

Daniel Gofman

Simon Porten

## B Anleitung

Auf der beigelegten CD (oder alternativ auf [Github](#)) befindet sich das gesamte Unityprojekt. Um das Programm direkt (ohne den Editor) auszuführen, befindet sich im Unterordner /Builds/ eine ausführbare Datei 'VoxelVR\_Final.exe'.

Um das Projekt in Unity zu importieren, kann alternativ in Unity ein neues Projekt erstellt werden und über das Menü 'Assets -> Import Package -> Custom Package' die beiliegende Datei 'VoxelVR\_PPGofmanPortenWS1718.unitypackage' importiert werden. Für dieses Projekt ist die Unity Version 2017.1.0 empfohlen.