Fachbereich 4: Informatik

Simulation von Augmented Reality Brillen auf Virtual Reality Brillen in Unity

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Marcel Pohl

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	M.Sc. Gerrit Lochmann (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im November 2016

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐

Koblenz, 15.11.2016 ..... M. Pohl

(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit
Marcel Pohl
(Mat. Nr. 213 200 275)

Thema: Simulation von AR Brillen auf VR Brillen in Unity

Augmented Reality ist neben Virtual Reality eines der Gebiete, für die in Zukunft eine große Vielfalt an Anwendungsbereichen denkbar ist. Insbesondere im Alltag können AR Brillen dazu verwendet werden, uns in verschiedensten Situationen mit nützlichen Informationen zu versorgen. Da AR Brillen von verschiedenen Herstellern jedoch sehr unterschiedliche Spezifikationen (z.B. Field of View) aufweisen, gestaltet sich eine Plattformübergreifende Entwicklung von Anwendungen schwierig.

Im Rahmen dieser Arbeit soll das Entwickeln von Anwendungen für verschiedene AR Systeme vereinfacht werden. Der Ansatz hierfür ist die Verwendung einer VR Brille, welche verschiedene Umgebungen darstellen und darin unterschiedliche AR Brillen simulieren kann. Ziel der Arbeit ist es eine VR Anwendung zu entwickeln, die es ermöglicht gewünschte Umgebungen darzustellen und darin die Ansichten bestimmter AR Brillen einzufügen. Auf diese Weise lässt sich überprüfen, ob die Anwendung auf den Zielsystemen richtig angezeigt wird.

Für die Entwicklung der Anwendung wird die Unity Game Engine verwendet werden, das primäre Zielsystem ist die Samsung Gear VR. Eine Portierung auf weitere VR Systeme ist möglich.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche technischer Daten verschiedener AR Brillen
2. Umgang mit der Unity Game Engine, insbesondere auch dem VR Modul
3. Konzeption einer VR Anwendung zur Simulation von AR Brillen
4. Entwicklung der VR Anwendung in Unity
5. Dokumentation der Ergebnisse

Koblenz, 29.04.2016

M. Pohl

– Marcel Pohl –

S. Müller

– Prof. Dr. Stefan Müller –

Zusammenfassung

Augmented Reality besitzt viele denkbare Anwendungsbereiche, in denen Alltag oder Arbeitsprozesse vereinfacht werden können. Dadurch, dass viele Hersteller sehr unterschiedliche Augmented Reality Brillen anbieten, wird die Auswahl des richtigen Systems und eine systemübergreifende Entwicklung jedoch erschwert. Im Rahmen dieser Bachelorarbeit wird daher eine Anwendung entwickelt, mit der Augmented Reality Brillen auf einem Virtual Reality System simuliert werden können. Damit soll eine Plattformübergreifende Entwicklung sowie die Auswahl des richtigen Systems vereinfacht werden.

Da die Simulation für mobile Endgeräte konzipiert werden soll, sollen möglichst realistische Umgebungen als Panorama vorgerendert werden können. Um diese auf Virtual Reality Systemen als stereoskopische Bilder darstellen zu können, werden verschiedene Verfahren zur Konvertierung in solche vorgestellt. Es wird ein Editor entwickelt, mit dem verschiedene Szenarien erstellt, Augmented Reality Systeme konfiguriert und schließlich auf einem Virtual Reality System angezeigt werden können. Abschließend wird untersucht, wie gut die Simulation gelungen ist und welche Verbesserungsmöglichkeiten es gibt.

Abstract

Augmented Reality has many areas of application. It can be used to simplify everyday life as well as working processes. However, since there are many manufacturers that offer greatly varying systems, choosing the correct system according to application as well as cross-platform development are difficult. This thesis attempts to develop an application which can be used to simulate Augmented Reality devices on Virtual Reality systems. This should simplify the processes of choosing a system as well as cross-platform development.

Since the simulation will be designed to run on mobile devices, it should be possible to render high quality, realistic environments in advance, using a panoramic image. On a Virtual Reality device, they need to be displayed as a stereoscopic image. To archive this, several methods are presented that can be used to perform this conversion. An editor will be created which will allow the creation of scenes, configuration of Augmented Reality devices and displaying them on a Virtual Reality system. For closing this thesis a test will be performed, to check the quality of the simulation as well as improvements that can be made.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation und Ziele

Augmented Reality (AR) und Virtual Reality (VR) sind zwei Begriffe, die aktuell nicht nur in Fachkreisen für viel Aufmerksamkeit sorgen. Insbesondere in dem Bereich der Virtual Reality ist mit der Veröffentlichung von gebrauchstauglichen und erschwinglichen Endverbrauchermodellen, wie der Oculus Rift und der HTC Vive, viel Bewegung gekommen. Im Vergleich dazu lassen entsprechende Geräte im Bereich der Augmented Reality noch auf sich warten. So stützt sich dieser derzeit zumindest bei Endverbrauchern primär auf Anwendungen, die Bilder oder Videos einer Kamera um virtuelle Objekte oder Informationen ergänzen. Während VR durch seine abgeschottete Natur nur ein eher eingeschränktes Anwendungsgebiet zulässt, sind dafür umso mehr Szenarien für den Einsatz von AR vorstellbar. Diese kann uns im Alltag wie Beruf unterstützen und viele Prozesse erleichtern.

Mit Produkten wie der Microsoft HoloLens oder Google Glass, die in absehbarer Zeit auch für den Endbenutzer erschwinglich und interessant werden könnten, sowie Produkten die bereits im professionellen Umfeld zum Einsatz kommen, lohnt es sich mittlerweile auch über die Entwicklung entsprechender AR-Anwendungen nachzudenken. Im Gegensatz zu VR-Brillen fallen die Gestaltung sowie die Fähigkeiten der AR-Brillen verschiedener Hersteller sehr unterschiedlich und vielfältig aus. So gibt es Beispielsweise große Unterschiede in der Größe des Sichtfeldes, der Bildschirmauflösung oder der Art und Weise, wie Informationen dargestellt werden. Im Rahmen dieser Arbeit soll deshalb geprüft werden, in wie weit es möglich ist die Entwicklung, insbesondere das Design, von AR-Anwendungen zu vereinfachen. Ein besonderes Augenmerkt soll hierbei darauf liegen verschiedene AR-Brillen vergleichbar zu machen, so dass man eine AR-Anwendung für verschiedene Systeme optimieren bzw. sich eine geeignete Plattform für die eigene Anwendung auswählen kann. Auch auf Probleme welche bei einer Simulation von AR-Brillen aufkommen können, soll im Rahmen dieser Arbeit eingegangen werden. Um mit dem Simulator auch auf schwächeren Endgeräten eine möglichst gute Darstellung erreichen zu können, wird außerdem die Verwendung vorgerenderter 360°-Panoramen untersucht. Zusätzlich werden verschiedene Verfahren zur Generierung stereoskopischer Bilder für ein VR-System vorgestellt, was insbesondere für die stereoskopische Darstellung vorgerenderter Panoramen notwendig ist.

Den Schwerpunkt und Kern dieser Arbeit bildet die Entwicklung einer Anwendung in der Unity Game Engine, welche die Simulation von AR-Brillen auf VR-Systemen ermöglicht. Die Anwendung ist in zwei Hauptbereiche unterteilt, der Darstellung der virtuellen Welt und die der darin eingebetteten AR-Informationen und -Objekte. Die virtuelle Welt soll entweder direkt über das Rendern von 3D Objekten in der Szene selbst oder die bereits er-

wähnten 360°-Panorama-Bilder dargestellt werden können. Die AR-Inhalte werden wie bei einer echten AR-Brille über den Hintergrund gelegt und können mit verschiedenen Parametern für einen möglichst akkuraten Eindruck konfiguriert werden. Abschließend erfolgt eine Expertenevaluation und eine Bewertung des Simulators. Im Rahmen dieser wird auch auf Probleme im Zusammenhang mit der Simulation von Augmented Reality eingangen, sowie ein Ausblick auf die zukünftige Möglichkeiten gegeben.

1.2 Aufbau der Arbeit

Die Arbeit ist in insgesamt sieben Kapitel gegliedert. Zuerst werden grundlegende Begriffe, die in dieser Arbeit verwendet werden, erklärt sowie ein Überblick über die verwendete bzw. zu simulierende Hardware gegeben. Danach folgt ein Überblick über vorausgegangene Arbeiten um einen ersten Eindruck zu vermitteln, welche Ergebnisse im Rahmen dieser Arbeit zu erwarten sind. In Kapitel 4 werden die Anforderungen an die Arbeit sowie die verwendeten Systeme dargestellt sowie das Konzept vorgestellt. Um auch auf schwächeren Endgeräten eine möglichst qualitativ hochwertige, stereoskopische Darstellung der Inhalte zu erzielen, werden anschließend mehrere Erfahren erläutert, mit denen aus vorgerenderten 360°-Panoramen entsprechende Umgebungen generiert werden können. Danach wird die im Rahmen dieser Arbeit entwickelte Software vorgestellt. Abschließend wird die durchgeführte Evaluation präsentiert und die Funktionalität des Simulators bewertet, gefolgt von einem Fazit und Ausblick.

2 Definitionen und Hardwareübersicht

2.1 Augmented Reality

Den Begriff *Augmented Reality* (AR) kann man mit verbesserter, erweiterter oder angereicherter Realität übersetzen. Es wird damit also eine Technologie beschrieben, welche die durch den Anwender wahrgenommene Realität mit zusätzlichen Informationen anreichert. In der einfachsten Form kann es sich dabei um reine Texteinblendungen handeln. Führt man das Konzept weiter, können Informationen auch so in die Umwelt eingebettet werden, dass der Anwender sie als Bestandteil davon wahrnimmt.

R. Azuma führte 1997 eine Studie zu AR durch [13] und stellte darin drei grundlegende Anforderungen auf. Demnach sollte AR

- reelles und virtuelles kombinieren,
- interaktiv sein und in Echtzeit arbeiten,
- sowie korrekt in den dreidimensionalen Raum eingebettet werden.

Milgram und Colquhoun [1] bauen 1999 darauf auf, verwenden allerdings eine stärkere Differenzierung. Dazu führen sie zunächst ein *Realitäts-Virtualitäts (RV) Kontinuum* ein. An dessen Enden befinden sich die ausschließlich reale Welt sowie die ausschließlich virtuelle Welt, wie es in Abbildung 1 dargestellt ist. Parallel dazu existiert ein *Extend of World Knowledge (EWK) Kontinuum*. Dieses gibt an, wie viel der Computer über die angezeigte Welt weiß. Während eine rein virtuelle Umgebung eine komplett modellierte Welt voraussetzt, liegt bei der rein realen Welt überhaupt keine Modellinformation vor.

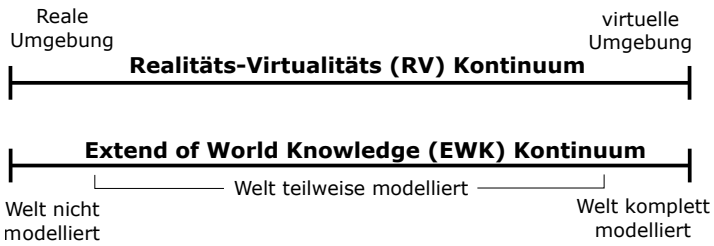

Abbildung 1: Realitäts-Virtualitäts Kontinuum, parallel zum Extend of World Knowledge Kontinuum [1]

Zwischen den Extrema kann eine Mischung zwischen realen und virtuellen Bildern statt finden. In der Mitte ist die Welt teilweise modelliert. Das heißt es kann Objekte geben, von denen man den Ort kennt an welchem sie sich befinden, aber nichts über das Objekt selbst. Sowie andere, bei denen man zwar viel über die Geometrie der Objekte weiß, jedoch nichts über ihren

Ort. Die linke Hälfte des RV bezeichnen Milgram und Colquhoun daher als Augmented Reality, die rechte als *Augmented Virtuality* (AV), wobei der Übergang flüssig ist. Bewegt man sich auf der RV Achse von der realen Umgebung in Richtung virtuelle Umgebung, stellt sich die Frage, ab wann man die reale Welt um virtuelle Objekte oder die virtuelle Welt um realen Daten erweitert.

Für einen einfacheren Umgang, ohne zwischen AR und AV entscheiden zu müssen, führen die Autoren daher den Begriff der *Mixed Reality* (MR) ein. Dieser deckt den gesamten Bereich des RV ab, schließt dabei die Endpunkte der reinen realen bzw. virtuellen Umgebung aber aus. Abbildung 2 zeigt den Zusammenhang zwischen AR, AV, MR und RV.

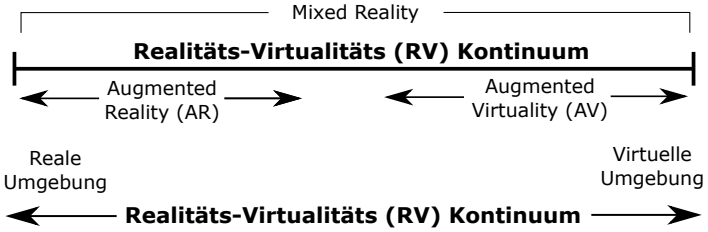

Abbildung 2: Definition der Mixed Reality im Kontext des Realitäts-Virtualitäts Kontinuum [1]

Die Darstellung von AR kann in zwei Kategorien unterteilt werden, *optisches see-through AR* sowie *Video see-through AR* [14]. Bei optischem see-through AR sieht man direkt die reale Welt. Virtuelle Inhalte werden mit einem Display, durch welches man durchschauen kann, eingeblendet. Für Video see-through AR wird mit einer Videokamera die reale Umgebung aufgezeichnet. Auf diese werden die virtuellen Inhalte gerendert und anschließend auf einem normalen Display angezeigt.

Während mit AR oft nur die visuelle Wahrnehmung assoziiert wird, deckt dieses Gebiet ein deutlich größeres Spektrum ab. Um die Immersion zu erhöhen, können auch räumlich korrekt positionierte akustische Signale und haptische Rückmeldung genutzt werden [14].

2.2 Virtual Reality

Virtual Reality (VR) lässt sich recht einfach mit "virtueller Realität" ins Deutsche übersetzen. Schlägt man den Begriff virtuell im Duden nach, wird dieser mit "nicht echt, nicht in Wirklichkeit vorhanden, aber echt erscheinend" [15] beschrieben. Kurz könnte man also VR also mit "Scheinrealität" übersetzen.

Sherman und Craig [16] definieren in ihrem Buch vier Schlüsselelemente von VR. Diese sind eine *virtuelle Welt*, *Immersion*, *sensorisches Feedback*

und *Interaktivität*.

Eine *virtuelle Welt* wird durch in ihr befindliche Objekte, deren Beziehungen zueinander sowie weiteren Regeln beschrieben. Dabei ist es unerheblich, ob diese Welt lediglich in den Gedanken ihres Erschaffers existiert, oder über ein für andere verständliches Medium geteilt wird. Ein Beispiel für eine virtuelle Welt ist ein von einem Autor erfundenes und definiertes Universium, welches in Form von Büchern oder Filmen verbreitet wird.

Immersion lässt sich mit vertiefen oder eintauchen beschreiben, dem Gefühl in einer bestimmten Umgebung zu sein. Hierbei kann es sich um eine alternative Realität, z.B. eine virtuelle Welt, aber auch lediglich einen anderen Blickpunkt auf die reale Umgebung handeln. In Hinblick auf VR ist anzumerken, dass diese Immersion über extrinsische Einflüsse statt findet. Es wird also etwas anderes wahrgenommen, als ohne diese Einflüsse. Entscheidend für ein hohes Maß an Immersion ist die Hemmung bzw. Aufhebung von Zweifeln, wodurch die präsentierten Inhalte real wirken. Sherman und Craig unterteilen zudem in physische und geistige Immersion. Bei VR steht am Anfang die physische Immersion. Hierbei geht es vornehmlich darum, menschliche Sinne unter Verwendung von Technologie zu stimulieren. Die Immersion kann sich dabei auf einzelne Sinne, beispielsweise das Sehen, beschränken. Das Einbinden von mehreren Sinnen erhöht die Immersion. Die geistige Immersion dagegen bezieht sich auf die präsentierten Inhalte. Dazu zählt geistig in die dargestellte Umgebung und Ereignisse vertieft zu sein, die bereits angesprochene Hemmung bzw. Aufhebung von Zweifeln sowie eingebunden zu sein.

Sensorisches Feedback beschreibt jegliche Form physikalischer Interaktion zwischen einem VR-System und seinem Benutzer. In seiner einfachsten Form kann es sich dabei um das Verfolgen von Kopfrotation oder auch der Position handeln, Bewegungen die in die virtuelle Welt übertragen werden. Auch das Abblilden von Bewegungen der Hände oder speziellen Gegenständen ist möglich. Fortgeschrittene Systeme können auch die wichtigsten und offensichtlichsten Gelenke des Körpers verfolgen. Neben der Übertragung von physikalischen Abläufen an das VR-System kann ein Feedback natürlich auch in umgekehrte Richtung erfolgen. Ein Beispiel hierfür wäre force feedback, wie es in vielen Spielecontrollern eingesetzt wird.

Als letztes gehen Sherman und Craig auf *Interaktivität* ein. Interaktivität beschreibt hierbei jede mögliche Form der Interaktion mit dem VR-System. Egal ob es lediglich das Verändern des Blickpunkts oder weitreichendere Interaktion mit der virtuellen Umgebung ist, wichtig ist die aktive Einflussnahme des Benutzers auf die virtuelle Welt. Dies steht im Gegensatz zu anderen Medien wie Filmen oder Büchern, bei denen Orte und Handlung vorgegeben sind. Eine besondere Form von Interaktivität ermöglichen *kollaborative Welten*. In diesen können mehrere Benutzer mit der gleichen virtuellen Welt interagieren. Die Repräsentationen der unterschiedlichen Teilnehmer werden als Avatare bezeichnet.

Auf diesen Ausführungen basierend stellen Sherman und Craig schließlich folgende, zusammengefasste Definition von VR auf:

"Virtual Reality ist ein Medium aus interaktiven Computersimulationen, welche Bewegungen und Aktionen der Teilnehmer erkennen, die Feedback an einen oder mehrere Sinne ersetzen oder erweitern und damit das Gefühl einer geistigen Immersion geben bzw. in der Simulation (einer virtuellen Welt) anwesend zu sein." [16]

2.3 AR-Brillen

Mittlerweile gibt es eine Fülle an AR-Brillen, die sich entweder noch in Entwicklung befinden oder bereits auf dem Markt verfügbar sind. An dieser Stelle wird daher ein kurzer Überblick über die Modelle gegeben, die seitens der Volkswagen AG für diese Arbeit gewünscht wurden.

2.3.1 Epson BT-200

Abbildung 3: Epson BT-200 [2]

Bei der Epson BT-200 handelt es sich um eine binokulare, optische see-through AR-Brille. Sie besitzt zwei transparente Brillengläser, in welche je ein Bildschirm eingebettet ist. Nach Epson [17] besitzen diese je eine Auflösung von 960 x 540 Pixeln und hat ein Sichtfeld von 23°.

Weiterhin besitzt sie eine externe Steuereinheit [18], auf der ein Android System läuft und über welches Anwendungen ausgeführt werden können. Die Navigation von AR-Anwendungen kann dabei auf zwei verschiedene Weisen erfolgen. Zum einen gibt es auf der Steuereinheit ein Touchfeld, dass Eingaben über Gesten wie bei einem normalen Smartphone entgegennimmt. Als zweite Eingabemöglichkeit dient ein in der Brille verbauter Motion-Tracker, der nicht nur Kopfrotationen, sondern auch Positionsveränderungen wahrnehmen kann. Die Epson BT-200 ist dazu in der Lage, sowohl einfache Einblendungen wie zum Beispiel das Abspielen von Videos zu machen, als auch

AR-Inhalte wie Annotationen an festen Positionen in der realen Welt anzuheften. Letzteres wird durch eine integrierte Kamera an der Vorderseite der Brille ermöglicht. Anhand der von ihr bereitgestellten Videos können über Marker oder die Erkennung von speziellen Texturen entsprechende Einblendungen vorgenommen werden.

2.3.2 Google Glass

Abbildung 4: Google Glass [3]

Bei der Google Glass handelt es sich um eine monokulare, optische seethrough AR-Brille. Im Gegensatz zu anderen AR-Brillen befindet sich das Display nicht direkt vor dem Auge im direkten Blickfeld, sondern leicht darüber. Die Einblendung der virtuellen Inhalte erfolgt über einen Prismaprojektor mit einer Auflösung von 640 x 360 Pixeln [19], dessen Sichtfeld rund 14° beträgt [20].

Vom Design her dient die Google Glass eher der Anzeige von zusätzlichen Informationen als dazu virtuelle Gegenstände in der realen Welt zu platzieren. Sie verfügt über Bewegungssensoren, durch die Kopfbewegungen von den aktiven Anwendungen berücksichtigt werden können. Dies kann beispielsweise dazu dienen, in Navigationsanwendungen die Karte korrekt zu rotieren. Über eine integrierte Kamera können Bilder und Videos aufgenommen werden, mit denen auch Funktionen wie Gesichtserkennung ausgeführt werden können. Die Google Glass kann per Bluetooth mit einem Smartphone verbunden werden, welches den Funktionsumfang erweitert. Hierüber werden unter anderem SMS Funktionen, Navigation und die Verbindung zum Internet ermöglicht [21].

2.3.3 Microsoft Hololens

Die Microsoft Hololens ist eine bikulare, optische see-through AR-Brille. Wie auch die Epson BT-200 verfügt sie über zwei transparente Brillengläser, in

Abbildung 5: Microsoft Hololens [4]

welche jeweils ein Bildschirm eingearbeitet ist. Das Sichtfeld wird auf 30° horizontal und 17, 5° vertikal geschätzt [22]. Die Bildschirme der Hololens lösen mit einer Auflösung von 1920 x 1080 Pixeln auf [23]. Durch die Verwendung von zwei Bildschirmen ist eine stereoskopische Darstellung der virtuellen Inhalte möglich.

Bisherige Demonstrationen zeigen die Verwendung der Hololens primär dazu, virtuelle Inhalte in die reale Welt zu projizieren. Das können zweidimensionale Inhalte wie Notizen oder Videos, sowie 3D-Objekte sein. Die Inhalte können fest in der Welt platziert, aber auch an die Hololens geheftet werden, so dass sie sich mit dieser zusammen bewegen oder rotieren. Die Hololens benötigt keine zusätzlichen angeschlossenen Geräte. Spatial Mapping ermöglicht das Platzieren von Hologrammen in der Welt, das heißt die Umgebung in der der Träger sich aufhält, wird vor dem Einsatz zuerst erfasst und gespeichert. Als Steuerung können Gesten, Sprache oder auch eine per Bluetooth verbundene Tastatur und Maus verwendet werden.

2.4 Samsung Gear VR

Abbildung 6: Samsung Gear VR [5]

Als VR-System für die Simulation der AR-Brillen wird eine Samsung

Gear VR verwendet. Die Gear VR ist ein Headset, welches in Kombination mit bestimmten Samsung Smartphones zu einer VR-Brille wird [24]. Dieses ermöglicht eine stereoskopische Bilddarstellung mit einem Sichtfeld von ca. 96°. Die Auflösung ist abhängig von dem verwendeten Smartphone und liegt derzeit bei maximal 1440 x 2560 Pixeln. Da auf dem Bildschirm die Bilder für beide Augen angezeigt werden müssen, steht jedoch nur maximal die halbe Auflösung pro Auge zur Verfügung. Diese wird durch die spezielle VR-Darstellung (Abbildung 7) zusätzlich eingeschränkt.

Abbildung 7: Anzeige von VR-Inhalten auf einem Display

Das Gear VR Headset für Samsung Smartphones stellt neben einer reinen Halterung und dem entsprechenden optischen System für die Anzeige von VR weitere Funktionen bereit [24]. Zur Steuerung von Anwendungen sind zum einen ein Touchpad sowie mehrere Tasten vorhanden. Zum anderen sind aber auch ein Beschleunigungssensor, ein Lagesensor und ein Näherungssensor verbaut. Mit Beschleunigungs- und Lagesensor können Kopfbewegungen an das Smartphone übertragen und zur Kamerasteuerung verwendet werden. Der Näherungssensor schaltet das Smartphone automatisch in Standby sobald die Brille abgesetzt wird.

3 Vorausggangene Arbeiten

Eine grundlegende Veröffentlichung zu Augmented Reality wurde 1997 von Ellis et al. [25] verfasst. In zwei Experimenten wurde unter anderem die Auswirkung von Latenz auf Tätigkeiten, welche eine hohe Präzision erfordern, untersucht. Eingesetzt wurde ein head-mounted Display (HMD) im see-through Modus. Über das Bild der realen Welt wurden AR-Objekte gelegt. Für die Auswirkung der Latenz mussten virtuelle Ringe unterschiedlicher Größe an verschiedenen virtuellen Pfaden entlang bewegt werden, ohne diese zu berühren. Die Versuchsreihe zeigte deutliche Auswirkungen von Latenz auf die Durchführung der Aufgabe. Dieses Experiment wird von späteren Arbeiten aufgegriffen und nachgestellt.

Die früheste zu findende Arbeit, in der mit Simulation von AR in einer VR-Umgebung gearbeitet wurde, ist 2006 von Gabbard et al. [26] veröffentlich worden. In dieser wurde untersucht, wie sich unterschiedliche Darstellungsweisen von Text auf einer AR-Brille auf die Verwendbarkeit von AR in Außenbereichen auswirkt. Dabei wurden insbesondere die durch die Brille wahrgenommene Hintergrundtextur, sowie die Lichtverhältnisse berücksichtigt. Für die Untersuchungen wurde eine Cave Automatic Virtual Environment (CAVE) verwendet, die AR-Inhalte wurden mit einer realen AR-Brille angezeigt. Ausschlaggebend für diese Entscheidung war vor allen Dingen, die volle Kontrolle über die simulierte Umgebung zu besitzen und die vielen unkontrollierbaren Faktoren der realen Welt auszuschalten. Das Verwenden einer CAVE, insbesondere unter Verwendung von Fotos, ließ außerdem ein hohes Maß an Immersion vermuten. Auch die korrekte Positionierung der AR-Inhalte wurde deutlich erleichtert. Insgesamt bewerteten Gabbard et al. dieses Verfahren als tauglich, um Innenräume sowie Nacht- und Dämmerungsszenarien nachzustellen. Für helle Tage, insbesondere bei Sonnenschein, sei es allerdings ungeeignet. Technische Einschränkungen ließen nicht die korrekte Simulation der Lichtverhältnisse zu, auch eine relative Anpassung der Lichtverhältnisse wurde als nicht auf reale Szenarien übertragbar bewertet.

In 2009 veröffentlichten S. Kim und A. K. Dey [27] eine Arbeit, in der die Verwendung eines AR-Windschutzscheibendisplays untersucht wurde. Dieses sollte insbesondere ältere Autofahrer unterstützen. Im Gegensatz zu Gabbard et al. wurde hier keine VR verwendet oder AR-Brille simuliert. Stattdessen wurde ein Fahrsimulator entwickelt, der die gewünschten AR-Inhalte einblendet und auf einem normalen TV-Bildschirm angezeigt wurde. In Tests mit Probanden wurde festgestellt, dass eine Einblendung von Informationen über ein Augmentiertes Windschutzscheibendisplay deutliche Vorteile gegenüber eines normalen Navigationssystems bringt. Die Veröffentlichung von Kim und Dey zeigt für meine Arbeit vor allen Dingen, dass sich Design und Auswirkungen von AR simulieren und auf reale Szenarien übertragen lassen.

Ebenfalls in 2009 untersuchten Ragan et al. [28] die vollständige Simulation von AR-Systemen in VR-Umgebungen. Vorteile seien die perfekte Kontrolle über Parameter, das Testen von noch nicht vorhandener Hardware, eine günstige Möglichkeit Designs zu überprüfen sowie Umgebungen zu testen, für die Real-Tests zu teuer oder gefährlich wären. Als Nachteile wurden fehlendes physisches Feedback, Bewegung und Tracking in der VR-Welt, die Limitierung der Helligkeit und eine unterschiedliche Raumwahrnehmung zwischen VR und Realität angeführt. Als Proof of Concept wurde das Experiment von Ellis nachgebildet, wobei eine CAVE zum Einsatz kam. Das Fazit war positiv, VR sei grundsätzlich geeignet zur Simulation von AR, allerdings mit Einschränkungen.

Ein Jahr später führten Lee et al. [29] weitere Untersuchungen in Bezug auf die Auswirkungen von Latenz auf die Gültigkeit von simulierter AR durch. Sie verwendeten dazu ein HMD um zuerst das Experiment von Ellis zu reproduzierten und anschließend den Effekt der VR-Latenz auf die simulierte AR zu prüfen. Dabei stellten sie fest, dass es sich scheinbar um einen additiven Effekt handelt und die Latenzen nicht miteinander interagieren. Damit kann simuliertes AR vergleichbare Ergebnisse wie reale AR-Tests liefern.

Aufbauend auf ihre vorangegangenen Untersuchungen, veröffentlichten Lee et. al [30] 2013 eine weitere Arbeit. In dieser wurden die Effekte der virtuellen Realität auf Suchaufgaben in einer Mixed-Reality (MR) geprüft. Dazu wurde wieder ein HMD verwendet, über welches die gleiche Szene in verschiedenen Varianten präsentiert wurde. Es gab drei rein virtuelle Darstellungen, welche einen unterschiedlich hohen Detailgrad besaßen sowie den Look-Through Modus als letzte Variante. In diesen Umgebungen mussten verschiedene Suchaufgaben durchgeführt werden. Die Auswertung zeigte, dass es insgesamt keine großen unterschiede durch die verschiedenen Detailgrade gab. Den größten Einfluss schienen Licht und Kontrast zu haben, was sich insbesondere im Look-Through Modus bemerktbar machte und dessen Verwendung auf Zeiten einschränkte, in welchen es keine starke Sonneneinstrahlung gab. Insgesamt wurde die Simulation von MR als eine geeignete Methode eingestuft, die allerdings vom Szenario und technischen Einschränkungen abhängig ist.

4 Anforderungen und Konzeption

4.1 Zielsetzungen

Primäres Ziel dieser Arbeit ist die Erstellung eines Demonstrators, mit dem verschiedene AR-Systeme simuliert werden können. Dieser soll als eine VR-Anwendung erstellt werden und ein möglichst authentisches Gefühl erzeugen. Die verschiedenen Inhalte, wie der Hintergrund und die AR-Objekte, werden als separate Ebenen behandelt.

Für die Darstellung der Umgebung ist wichtig, dass möglichst realistische und detailreiche 360°-Umgebungen in Form eines stereoskopischen Bildes angezeigt werden können. Während aktuelle Spitzen-Computersysteme ohne Probleme 3D-Szenen in der geforderten Qualität berechnen können, muss auf Grund der Systemvorgaben (beschrieben in Abschnitt 4.2) auch auf andere Verfahren zurückgegriffen werden. Auf diese Weise soll ermöglicht werden, bereits im Voraus erstellte 360°-Panorama-Bilder an Stelle einer 3D-Szene für die Darstellung der Umgebung zu verwenden. Je nach eingesetztem Verfahren können diese Panoramen nicht nur durch das Rendern von 3D-Szenen, sondern auch aus Fotos von realen Umgebungen erstellt werden. Die Möglichkeit 3D-Szenen direkt zu rendern soll weiterhin gegeben sein. Die eigentliche Erstellung von 360°-Panoramen ist nicht Teil dieser Arbeit.

Die AR-Inhalte werden wie bei einer richtigen AR-Brille über den Hintergrund gelegt. Wie bereits erwähnt sollen sie sich auf einer separat gerenderten Ebene befinden. Die Art und Weise auf die Inhalte eingefügt werden und worum es sich dabei handelt, soll variabel gehalten werden. Zweidimensionale Texte und Grafiken sollen genauso möglich sein wie das Einfügen von dreidimensionalen Objekten. Die Inhalte können statisch sowie dynamisch sein, fest in der virtuellen Welt platziert oder an der Kamera fixiert werden. Besonders wichtig in diesem Zusammenhang die eigentliche Darstellung der Inhalte, so dass sie den simulierten AR-Brillen möglichst nahe kommen. Diese soll über verschiedene Parameter konfiguriert werden können. Seitens der Volkswagen AG wurden Sichtfeld, Helligkeit und Auflösung als Parameter vorgegeben. Außerdem soll die Überlagerung optischem see-through AR gleichen, nicht dem video see-through AR.

Neben der Erstellung des Demonstrators sollen im Rahmen dieser Arbeit verschiedene Methoden zur Generierung stereoskopischer Bilder untersucht werden. Die Grundlage für diese bilden 360°-Panoramen. Drei mögliche Vorgehensweisen werden in Kapitel 5 näher erläutert.

Zum Schluss soll der Demonstrator evaluiert werden. Hierfür ist eine Expertenevaluation mit wenigen Versuchspersonen geplant. Gegenstand der Evaluation ist die Bewertung der Simulation selbst. Hierbei ist insbesondere die Rückmeldung der Probanten bezüglich der Authentizität der simulierten AR-Brillen wichtig. Weitere Auffälligkeiten, insbesondere die verwendete Hardware und die damit verbundenen Einschränkungen betreffend, werden

ebenfalls untersucht.

4.2 System und Einschränkungen

Wie seitens der Volkswagen AG gewünscht, kommt als VR-System ein Samsung Galaxy S6 in Verbindung mit einer Samsung Gear VR zum Einsatz. Der Demonstrator wird in der Unity Engine (Version 5.4) auf einem Windows-System entwickelt. Als Programmiersprache kommt daher `C#` zum Einsatz, für Shaderprogrammierung wird OpenGL bzw. CG verwendet.

Für die Simulation ergeben sich hieraus Hardwarebezogene Einschränkungen. Da die Anwendung für ein Smartphone entwickelt wird, ist die Rechenleistung natürlich nicht vergleichbar mit einem Spitzen-Computersystem. Während eine einfache 3D-Umgebung durchaus in Echtzeit auf dem Gerät gerendert werden kann, ist das für die geplante Komplexität der darzustellenden Szenen nicht mehr möglich. Daher wird zusätzlich mit vorgerenderten 360°-Panoramen gearbeitet.

Neben Einschränkungen durch die Plattform ergeben sich auch allgemeine technische Limitierungen durch den Einsatz von VR-Systemen, die bereits in Kapitel 3 angesprochen wurden. So können nicht alle Beleuchtungsverhältnisse dargestellt werden, da aktuelle Bildschirme nicht genug Helligkeit erzeugen können. Auch eine relative Helligkeitsdarstellung wurde als nicht Zielführend bewertet [26]. Vergleicht man die Gear VR mit den zu simulierenden AR-Brillen (Tabelle 1) zeigt sich außerdem, dass auch diese eine höhere Helligkeit erzeugen können. Neben der Helligkeit stellt die Auflösung ebenfalls ein Problem dar. Während z.B. eine HoloLens offensichtlich nicht in nativer Auflösung simuliert werden kann, dürfte dies auch für die Google Glass und die Epson BT-200 nicht funktionieren. Auch wenn diese eine niedrigere Auflösung besitzen, ist gleichzeitig auch das Sichtfeld deutlich geringer. Das heißt die Pixeldichte ist deutlich höher als bei einer Gear VR.

	Gear VR	Google Glass	HoloLens	Epson BT-200
Auflösung	2560x1440 (1280x1440)	640 x 360	1920 x 1080	960 x 540
Sichtfeld	96°	14°	30°	23°
Helligkeit	345 cd / m^2	$\sim$ 3000 cd / m^2	k.A.	$\sim$ 800 cd / m^2

Tabelle 1: Vergleich der Brillen. Daten aus Kapitel 2 und von der Volkswagen AG.

Insgesamt ist festzuhalten, dass eine korrekte Simulation der Brillen sowie der Umgebung nicht möglich sein wird. Daher wird versucht, das Gefühl der simulierten Brillen so gut wie möglich nachzuempfinden und eine möglichst hohe Immersion zu erreichen.

4.3 Konzept

Insgesamt werden zwei Unity-Projekte angelegt. Eines wird zu der Erstellung der 360°-Panoramen verwendet und ist eine Windows-Anwendung, während das andere der eigentliche Demonstrator ist und daher als Android-Anwendung entwickelt wird. Eine Unterteilung in zwei getrennte Projekte ist nötig, da Unity alle in dem Projekt verlinkten Ressourcen in die finale Anwendung übernimmt. Je nach Anzahl der Szenen sowie deren Komplexität könnte die auf Android zu übertragende Paketdatei damit auf die Größe von mehreren Gigabyte anwachsen, während die Größe der benötigten Dateien nur wenige Megabyte betragen. Der Demonstrator selbst enhält daher nur tatsächlich in den Szenen verwendete 3D-Modelle sowie die dazugehörigen HintergrundPanoramen.

Im Demonstrator können verschiedene Umgebungen in Form von einzelnen Szenen angelegt werden. Sämtliche Inhalte und Einstellungen sollen direkt im Unity-Editor vorgenommen werden können. Hierfür wird eine Basisszene bereitgestellt, in welcher die nötige Grundstruktur vorhanden ist. Dazu zählen die nötigen Kameras und ein grundlegender Szenengraph, außerdem sind die nötigen Scripte und Einstellungen bereits den zugehörigen Objekten zugeordnet.

Nativ unterstützt der Unity-Editor das Erstellen von 3D-Szenen. Damit ist es möglich, einfache virtuelle Umgebungen zu erstellen, die direkt auf der Gear VR gerendert werden können. Auf diese Weise kann beispielsweise Prototyping betrieben werden, bevor die finale Szene modelliert und als Stereo-Panorama gerendert wird. Zusätzlich soll wie bereits erwähnt wenigstens eins der in Kapitel 5 vorgestellten Verfahren umgesetzt werden, um ein solches Stereo-Panorama als virtuelle Umgebung darzustellen. Die Priorität liegt dabei auf der Umsetzung des Post-Rendering 3D Warpings. Da zu diesem Zeitpunkt jedoch noch nicht absehbar ist, ob dieses in ausreichender Qualität oder überhaupt auf einem Samsung Galaxy S6 läuft, wird jedoch zuerst das einfachere Stereo Panorama umgesetzt. Auf diese Weise ist sichergestellt, dass zumindest eines der Verfahren implementiert werden konnte.

Es sollen AR-Objekte unterschiedlichster Art unterstützt werden. Diese lassen sich grundsätzlich in 3D und 2D-Objekte unterteilen. 3D-Objekte sollen primär dazu dienen, reale Objekte zu ersetzen, können aber auch als zusätzliche Objekte in die Welt gesetzt werden. Bei 2D-Objekten kann es sich um Text, Bilder oder Videos handeln. AR-Objekte sollen entweder als Annotationen bzw. Überdeckungen in der Welt fixiert oder als Einblendungen fest im Sichtfeld der AR-Brille positioniert werden können. Abbildung 8 zeigt ein Mockup für 2D-Inhalte die fest in der Welt positioniert sind, der abgedunkelte Bereich stellt den Sichtbereich der AR-Brille dar. Da die Art der AR-Objekte und der Positionierung beliebig miteinander kombinierbar sein werden, lassen sich auf diese Weise sämtliche Darstellungsweisen von aktuellen AR-Brillen darstellen. Um Trackingfehler und generelle Verzögerung

bei der Darstellung von AR-Inhalten zu simulieren, kann für eine AR-Brille eine Latenz eingestellt werden.

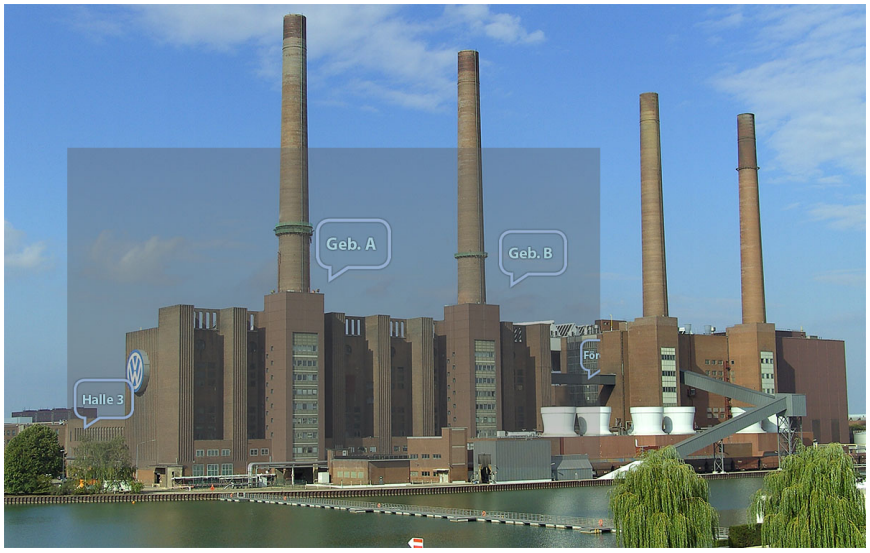

Abbildung 8: Mockup für 2D-Annotationen mit dem VW Wolfsburg Werk als Umgebung [6]

Die Steuerung des Demonstrators wird auf verschiedene Weisen möglich sein. Wie von einer VR-Anwendung zu erwarten wird die Kamera automatisch mit Kopfbewegungen rotiert. Es ist nicht vorgesehen sich in der Umgebung zu bewegen. Durch die Verwendung von verschiedenen Szenen ist es jedoch möglich die gleiche Umgebung aus unterschiedlichen Blickwinkeln zu betrachten. An die horizontale und vertikale Achse des Touchpads der Gear VR sollen über den Unity-Editor verschiedene Funktionen gebunden werden können. Dazu zählt beispielsweise das Wechseln von AR-Brille und Szene, aber auch die Möglichkeit einzelne Werte einer AR-Brille während der Laufzeit zu verändern. Das Menü der Anwendung wird mit einer Mischung aus Blick- und Touchpadsteuerung kontrolliert werden.

Über das Menü soll zum einen das Wählen von Szene und AR-Brille möglich sein, zum anderen die Einstellungen für die verschiedenen AR-Brillen angepasst werden können. AR-Brillen werden als verschiedene Voreinstellungen angezeigt, die sich editieren, löschen oder auch neu erstellen lassen. Um das Menü kompakt zu halten, wird es in verschiedene Untermenüs aufgeteilt. In Abbildung 9 ist das Menü in einem Mockup dargestellt.

Wichtig bei der Anzeige des Menüs ist, dass der Hintergrund stark genug abgedunkelt wird. Auf diese Weise soll klar erkennbar sein, wann sich der Benutzer im Menü befindet und wann in der Simulation.

Abbildung 9: Mockup für das Menü, im Hintergrund eine Szene mit AR-Inhalten [7]

Als mögliche Szenarien für die Simulation sind derzeit die Ansicht aus einem Fahrzeuginnenraums, eine Fabrikhalle (Abbildung 10) sowie eine Außenanlage geplant. Dementsprechend wird sich auch die Evaluation an diesen Szenarien orientieren.

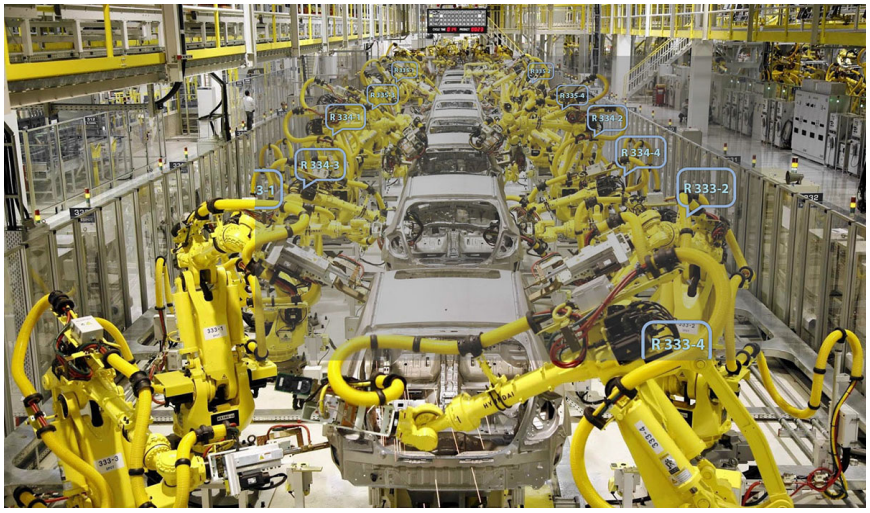

Abbildung 10: Mockup mit einer Fabrikhalle als Hintergrund [8]

5 Verfahren der Stereoskopischen Bilderstellung

Es gibt verschiedene Möglichkeiten um stereoskopische Bilder zu erstellen. Im einfachsten Fall verwendet man dazu, bei Fotos wie computergenerierten Bildern, eine stereoskopische Kamera. Verwendet man Panoramen zur Umgebungsdarstellung, ist eine stereoskopische Darstellung nicht mehr ohne Weiteres möglich. In diesem Fall müssen die Panoramen entweder auf eine spezielle Weise erstellt, oder die stereoskopische Umsetzung durch spezielle Renderverfahren ausgeführt werden. Im Nachfolgenden werden entsprechende Verfahren erläutert.

5.1 Stereo Panorama

Paul Bourke stellt in seiner Arbeit [9] ein Verfahren vor, mit welchem ein zylindrisches stereoskopisches Panoramabild aufgenommen werden kann. Erstellt man ein normales 360°-Panorama, wird eine (reale oder virtuelle) Kamera um ihre optische Achse gedreht und in regelmäßigen Abständen ein Bild gemacht. Für ein stereoskopisches Panorama reicht es aber nicht, die Kamera danach zu verschieben und ein zweites Panorama aufzunehmen. In Abbildung 11 (links) ist dieser Sachverhalt dargestellt. Sind die Sichtfelder der Kameras in der Ausgangsposition noch korrekt (schwarz, durchgezogen), existiert bei einer Rotation um 90° (rot, gestrichelt) keine Parallaxe mehr. Stattdessen müssen sich die Kameras um eine gemeinsame Rotationsachse drehen, von der beide gleich weit entfernt sind (Abbildung 11, rechts). Auf diese Weise besitzen alle aufgenommenen Bilder die gleiche Parallaxe. Der Abstand zwischen beiden Kameras wird als *interokularer Abstand* bezeichnet und wird abhängig von der Größe der Szene gewählt.

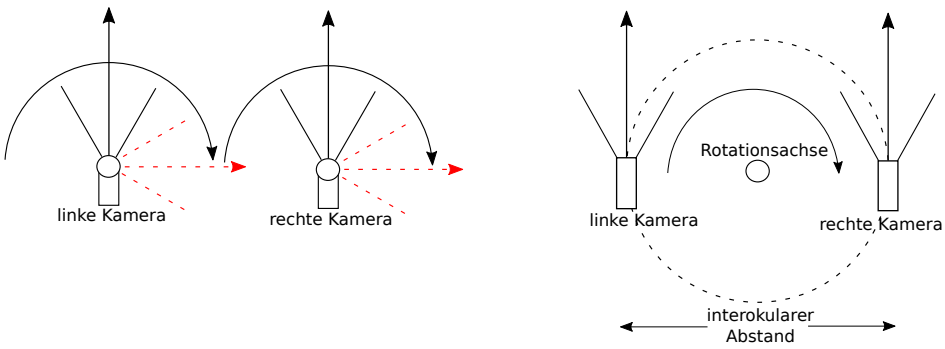

Abbildung 11: Falsches Stereo durch Verschiebung der Kamera (links) und korrektes Stereo Rig (rechts) [9]

Für ein normales 360°-Panorama reichen wenige Aufnahmen aus. Diese werden zusammengeschnitten, wobei Fehler durch die Kameraprojektion durch Überblendung reduziert werden. Die zusätzlichen Tiefeninformationen des Stereobildes verhindern jedoch eine solche Nachbearbeitung, da

auch geringe Fehler schnell auffallen. Daher müssen entsprechend mehr Bilder aufgenommen werden, die jeweils nur ein kleines Sichtfeld abdecken. Für gewöhnlich wird ein Sichtbereich von maximal 1° oder weniger verwendet um optimale Ergebnisse zu erzielen. Wird beispielsweise ein Sichtfeld von $0,5°$ verwendet, werden pro Kamera insgesamt 720 Bilder aufgenommen, die zusammengefügt das finale Panorama ergeben. Für jedes Bild werden die Kameras hierbei in gleichgroßen, diskreten Schritten rotiert, welche dem aufgenommenen Sichtfeld entsprechen. In Abbildung 12 wird dies repräsentativ für eine Kamera dargestellt.

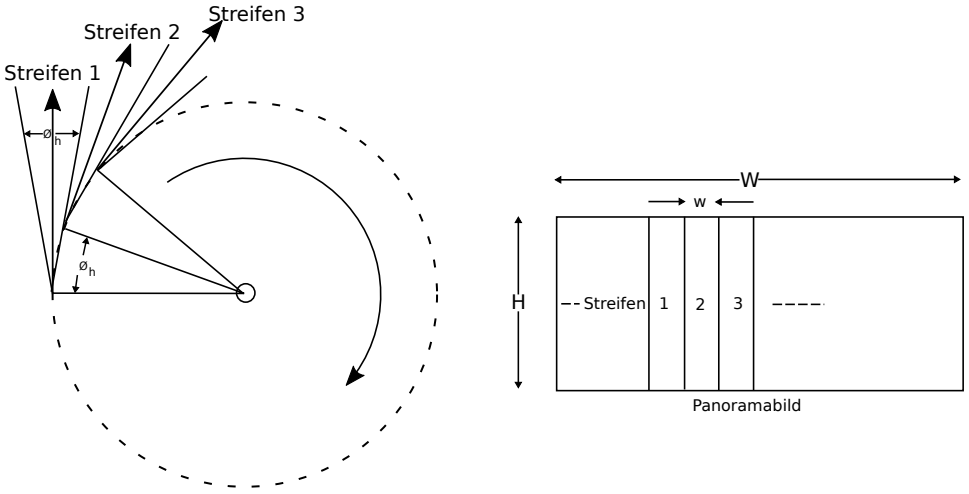

Abbildung 12: Die horizontale Sichtfeldgröße ϕ_h entspricht der Rotationsschrittweite, so dass im finalen Panorama keine Lücken entstehen [9]

Bei zylindrischen Panoramen stehen dessen Dimensionen in direkter Relation mit dem vertikalen Sichtfeld der Aufnahme. Dieses lässt sich durch die Gleichung

$$\phi_v = 2atan(H\pi/W) \tag{1}$$

berechnen. Hier bei stehen H und W für die Höhe und Breite des Panoramen. Für den Renderprozess muss das horizontale Sichtfeld ϕ_h so gewählt werden, dass es ein ganzzahliger Divisor von 360 ist. Nur so kann eine problemlose 360°-Abbildung gewährleistet werden. Die Breite w eines einzelnen Streifens berechnet sich dann durch

$$w = W * \phi_h/360 \tag{2}$$

wobei w ebenfalls eine Ganzzahl sein muss.

Wichtig für Stereoskopische Darstellung ist die so genannte *Null Parallaxe* f_0. Durch sie wird festgelegt, welche Entfernung Objekte in der Szene besitzen müssen, um später auf der Bildschirmebene zu liegen. Daraus folgend werden nähere Objekte vor dem Bildschirm (*negative Parallaxe*) und

weiter entfernte hinter dem Bildschirm angezeigt (*positive Parallaxe*). Mit der bisherigen, parallelen Positionierung der Kameras würde die Null Parallaxe im Unendlichen liegen. Um dies zu beheben, kann man eines der Bilder horizontal verschieben. Auf diese Weise ändert man den Zeitpunkt, an dem die Kamera eine bestimmte Position auf dem Kreis hat. Es entsteht eine Überschneidung der Kamera Blickrichtungen mit Winkel ϕ, wie es in Abbildung 13 zu sehen ist.

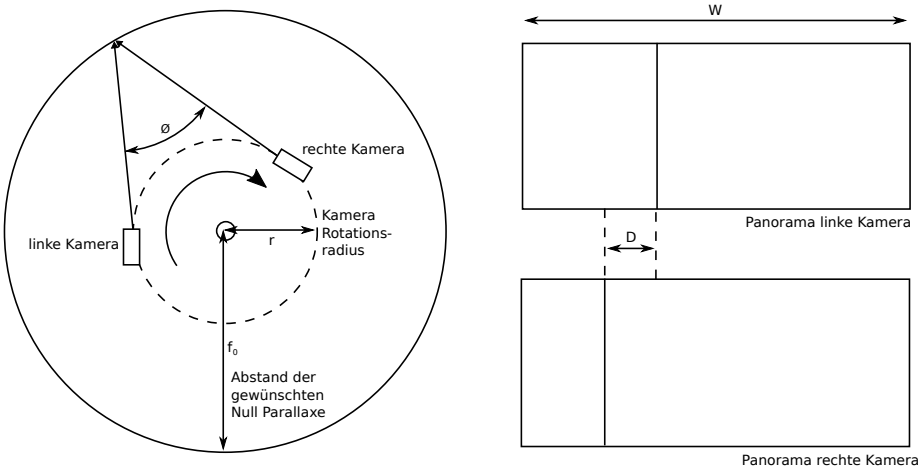

Abbildung 13: Geometrie für die Distanzberechnung, wie beide Panoramen verschoben sein müssen, um eine Null Parallaxe zu erreichen [9]

Die Anzahl der Pixel D um die man verschieben muss, kann man mit

$$D = W\phi/(2\pi) \tag{3}$$

berechnen, wobei ϕ durch

$$\phi = 2asin(r/f_0) \tag{4}$$

gegeben ist. Alternativ zur nachträglichen Panoramaverschiebung kann auch direkt mit richtig rotierten Kameras gerendert werden, dies wird als "*toe-in*" Kameras bezeichnet.

Während Paul Bourke sich auf zylindrische Panoramen beschränkt, übertragen Lee et al. [31] dieses Prinzip auf eine Kugel, um ein 360°-Sichtfeld in der Horizontalen wie Vertikalen zu ermöglichen. Zur Speicherung der Panoramen wird statt eines Zylinders nun eine spherische, auch *Equirectangular Map* genannt, verwendet. Alternativ dazu wäre auch eine *Cube Map* möglich, bei dem das Panorama auf die Seiten eines Würfels abgebildet wird.

Zusätzlich zu der vollständigen horizontalen Rotation wird abhängig vom vertikalen Sichtfeld die Kamera ebenfalls in der Vertikalen um einen Winkel θ_v rotiert. Dies geschieht von der Horizontalen ausgehend symmetrisch

in beide Richtungen, bis die Umgebung komplett abgedeckt ist. Die gerenderten Bildausschnitte werden dann an die entsprechende Position in die Equirectangular Map übertagen. Abbildung 14 zeigt ein fertiges Panorama im Equirectangular-Format.

Abbildung 14: Equirectangular Panorama

Zur Darstellung der Szene wird pro Auge eine Kamera verwendet. Um diese herum wird je eine Kugel erstellt, auf welche die Panorama-Textur für das entsprechende Auge gelegt wird. Auf diese Weise kann sich der Anwender mit korrektem stereoskopischen Eindruck umschauen.

Der Vorteil dieser Methode ist, dass der gesamte Aufwand der stereoskopischen Bilderstellung außerhalb der anzeigenden Anwendung anfällt. Lediglich das korrekte Rendern der Textur auf die Kugel fällt zur Laufzeit an. Zudem ist es auf diese Weise nicht nur möglich gerenderte Bilder zu erstellen, sondern auch reale Umgebungen zu fotografieren und entsprechend zu bearbeiten.

Als Nachteil ist zu erwähnen, dass das Neigen des Kopfes zum Verlust des stereoskopischen Eindrucks führt, was leicht nachvollziehbar ist. Bei der Generierung der Panoramen befanden sich die Kameras auf horizontaler Ebene nebeneinander. Eine Kopfneigung führt nun dazu, dass diese Kameraebene rotiert. Im Extremfall bei einer Kopfrotation von 90° steht sie senkrecht zu der Ebene, mit der die Aufnahmen gemacht wurden. Dadurch verändert sich auch die stereoskopische Wahrnehmung, was im Fall statischer stereoskopischer Panoramen nicht berücksichtigt wird.

5.2 Iterative Image Warping

Bowles et al. [10] stellen das *Iterative Image Warping* vor, dass nach dem Prinzip des *Gathering* (sammeln) arbeitet. Dieses Verfahren benötigt nur ein einzelnes Quellbild, für welches allerdings zusätzlich pro Pixel auch deren Tiefenwerte bzw. Weltkoordinaten bereitgestellt werden müssen. Das Verfahren dient primär dazu Kohärenzen zwischen aufeinanderfolgenden, geren-

derten Bildern auszunutzen und auf diese Weise den Rendervorgang zu beschleunigen. Anwendungsfälle hierfür sind beispielsweise die Erhöhung der Bildfrequenz, Tiefenunschärfe, Bewegungsunschärfe und das Erzeugen von stereoskopischen Bildern.

Ziel des Verfahrens ist es, ein Ausgangsbild I_s einer gerenderten Szene in ein Zielbild I_w zu überführen (*warpen*), welches einen anderen Blickwinkel oder eine zeitliche Veränderung darstellt, ohne die Szene dabei neu zu rendern. Zu diesem Zweck definieren Bowles et al. ein Vektorfeld $V : \mathbb{R}^2 \to \mathbb{R}^2$. Dieses beschreibt, wie jedes Pixel aus I_s verschoben werden muss um I_w zu erhalten. V kann unter Verwendung der Tiefenwerte sowie den Model-View-Projection-Matrizen von I_s und I_w vorab berechnet werden. Alternativ können die benötigten Werte auch direkt zur Laufzeit ermittelt werden.

Für einen beliebigen Punkt $\boldsymbol{x}_s$ aus I_s ist der transformierte Punkt $\boldsymbol{x}_w$ in I_w durch

$$\boldsymbol{x}_w = \boldsymbol{x}_s + V(\boldsymbol{x}_s) \tag{5}$$

gegeben, mit $\boldsymbol{x}_s, \boldsymbol{x}_w \in \mathbb{R}^2$. Da der Algorithmus für jedes Pixel in I_w ausgeführt und damit $\boldsymbol{x}_s$ gesucht wird, ist es sinnvoll die Funktion entsprechend umzuschreiben. Definieren wir also eine neue Funktion $G : \mathbb{R}^2 \to \mathbb{R}^2$ als

$$G(\boldsymbol{x}_s) = \boldsymbol{x}_w - V(\boldsymbol{x}_s), \tag{6}$$

kann man Gleichung 5 zu

$$\boldsymbol{x}_s = G(\boldsymbol{x}_s) \tag{7}$$

umschreiben. Da von I_w ausgegangen wird, sind anders als bei einer Transformation von Punkten aus I_s nach I_w keine Tiefeninformationen vorhanden. Daher kann keine einfache Rückprojektion von I_w nach I_s statt finden um die Farbe des Pixels zu bestimmen. Mittels *fixed point iteration (FPI)* wird nun ein Punkt $\boldsymbol{x}^*$ gesucht, der die Gleichung 7 mit $\boldsymbol{x}_s = \boldsymbol{x}^*$ erfüllt, wobei $\boldsymbol{x}^*$ einem fixen Punkt in G entspricht. Hierfür wird als Startpunkt $\boldsymbol{x}_0 = \boldsymbol{x}_w$ angenommen, also das Pixel, welches sich im Quellbild an der gleichen Position befindet, wie das aktuell bearbeitete im Zielbild. Ist das Pixel $\boldsymbol{x}_s$ für das Zielbild nicht verschoben worden, wird damit der FPI-Algorithmus gestartet.

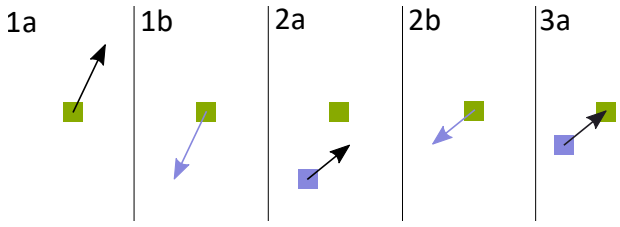

Abbildung 15: Ablauf FPI

Der Ablauf des FPI ist in Abbildung 15 visualisiert. Dabei entspricht das grüne Quadrat $\boldsymbol{x}_w$, das blaue $\boldsymbol{x}^*$. Für den Startwert x_0 wird dessen Verschiebung zwischen I_s und I_w berechnet bzw. in V nachgeschaut (1a). Mit

diesem Translationsvektor kann $\boldsymbol{x}^*$ berechnet werden, indem dieser von $\boldsymbol{x}_w$ abgezogen wird (1b). Für $\boldsymbol{x}^*$ wird nun erneut dessen Verschiebung berechnet bzw. nachgeschaut (2a). Endet die Verschiebung auf dem Pixel $\boldsymbol{x}_w$ des Zielbildes, ist die Lösung gefunden und es ist $\boldsymbol{x}_s = \boldsymbol{x}^*$ (3a). Wird das gefundene $\boldsymbol{x}^*$ an eine andere Stelle in I_w verschoben, wird auf die gleiche Weise weiter gesucht, bis ein passendes $\boldsymbol{x}^*$ gefunden wurde (2b).

Für das aktuelle Pixel $\boldsymbol{x}_w$ können verschiedene Szenarien zutreffen, bei denen sich der FPI unterschiedlich verhält. Diese sind in Abbildung 16 dargestellt.

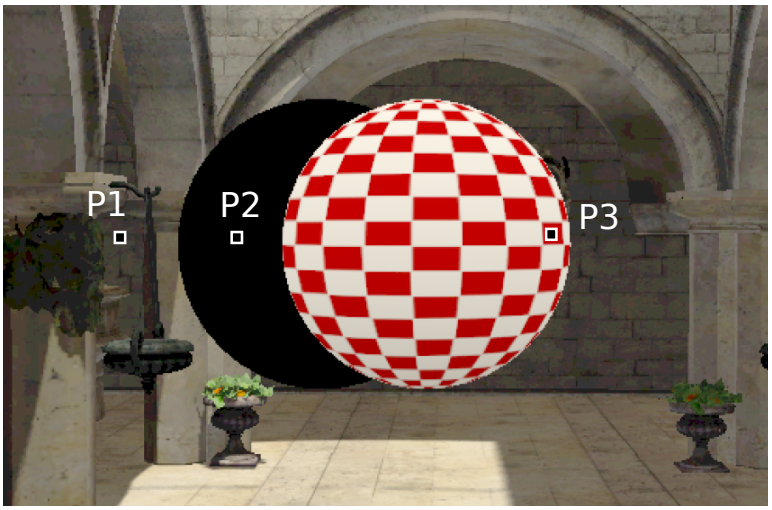

Abbildung 16: Beispiel für ein mit FPI transformiertes Bild [10]

Im Falle von P1 liegt $\boldsymbol{x}_w$ auf dem Hintergrund. In dieser Situation konvergiert der FPI gleichmäßig auf ein $\boldsymbol{x}^*$ und liefert ein eindeutiges Ergebnis. Die Farbe von $\boldsymbol{x}_w$ lässt sich somit ohne Probleme bestimmen.

An Punkt P2 ist die Situation eingetreten, dass ein Objekt durch Bewegen oder Änderung des Blickpunktes nicht mehr wie bisher den Hintergrund verdeckt (*disocclusion*). Sucht man nun an dieser Stelle nach einem geeigneten $\boldsymbol{x}^*$ wird der FPI keine Lösung finden, sondern nach einigen Iterationen zwischen zwei Punkten hin und her springen. Dies muss erkannt und der Algorithmus abgebrochen werden, da ansonsten eine Endlosschleife entsteht. Da keine Farbinformationen gewonnen werden können, ist der entsprechende Bereich hier schwarz eingefärbt.

Betrachtet man zuletzt Punkt P3, verdeckt hier nun ein Objekt den zuvor noch sichtbaren Hintergrund (*occlusion*). Führt man den FPI an dieser Stelle aus, besitzt dieser mehrere Ergebnisse. Welches er davon zurück gibt, hängt von der gewählten Startposition $\boldsymbol{x}_0$ ab. Bowles et al. schlagen an dieser Stelle vor eine vom Anwendungsfall abhängige Heuristik zu verwenden, mit der das vorderste Objekt erkannt und dessen Farbe zurückgegeben wird. Ansonsten kann es vorkommen, dass der Hintergrund das Objekt überdeckt

und abschneidet.

Um die undefinierten Stellen einer Disocclusion zu füllen, können verschiedene Ansätze gewählt werden. Eine Möglichkeit besteht darin, den von Solh und AlRegib vorgestellten *Hierarchical Hole Filling* Algorithmus zu verwenden [11].

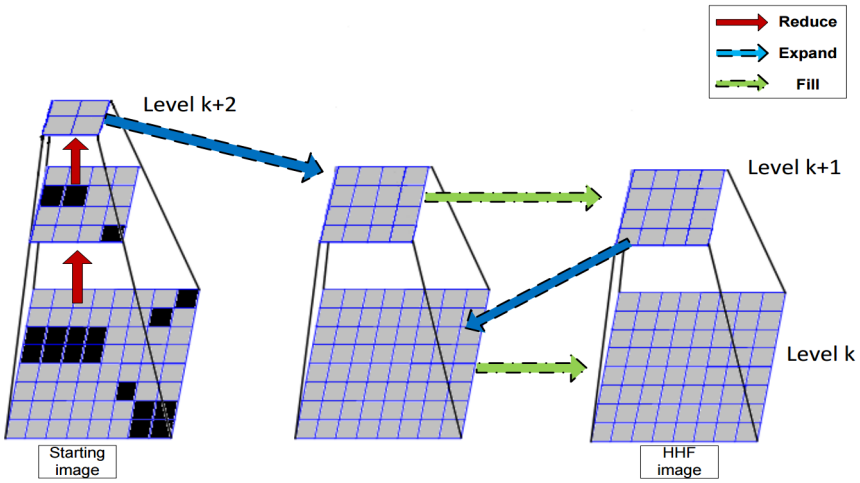

Abbildung 17: Hierarchical Hole Filling [11]

Wie in Abbildung 17 dargestellt, wird von einem Bild ausgegangen, welches undefinierte Stellen besitzt. Dieses wird schrittweise verkleinert, wobei immer mehrere benachbarte Pixel (beispielsweise 4) zu einem gemittelten Farbwert zusammengefasst werden. Dies wird so lange fortgeführt, bis alle undefinierten Stellen im Bild verschwunden sind. Im zweiten Schritt wird ausgehend vom kleinsten generierten Bild auf die nächsthöhere Stufe expandiert. Hierbei werden alle Pixel der höheren Stufe, die über keinen definierten Wert verfügen, mit den Durchschnittswerten der ihnen zugeordneten Pixel der unteren Stufe gefüllt. Es wird nun so lange abwechselnd expandiert und gefüllt, bis die höchste Ebene wieder erreicht wurde und alle undefinierten Stellen gefüllt sind.

Der Vorteil dieses Verfahrens ist, dass aus nur einem Bild mit wenigen Zusatzinformationen ein neuer Blickpunkt abgeleitet werden kann. Zudem arbeitet es sehr performant und ist vielseitig einsetzbar.

Als Nachteile sind ganz klar das Entstehen von undefinierten Stellen zu erwähnen, die mit speziellen Verfahren und Heuristiken gefüllt werden müssen. Auch die Stärke, um die sich ein Blickpunkt ändern, kann ist deutlich eingeschränkt, da eine stärkere Veränderung auch mehr undefinierte Stellen mit sich bringt.

5.3 Post-Rendering 3D Warping

Als letztes soll ein Verfahren von Mark et al. vorgestellt werden, welches nach dem *scattering* (Streuung) Prinzip arbeitet [12]. Wie bereits bei dem Iterative Image Warping wird hier lediglich ein Quellbild benötigt, für welches zusätzlich die Tiefenwerte oder die Weltkoordinaten der dargestellten Bildpunkte bereitgestellt werden müssen. Im Gegensatz zu dem Iterative Image Warping werden die Punkte von Quellbild I_s direkt ins Zielbild I_w transformiert, statt von I_w aus nach den Farbinformationen zu suchen. Das Verfahren ist primär zur Erhöhung der Bildfrequenz sowie zur Reduktion der Latenz für entfernte Rendersysteme entwickelt worden, eignet sich aber auch um Bilder in stereoskopisches 3D umzuwandeln.

Im einfachsten Fall wendet man auf I_s eine Reprojektion an. Dabei wird jeder Pixel $\boldsymbol{x}_s$ zuerst mit der für das Rendern verwendeten, inversen Model-View-Projection-Matrix $\boldsymbol{M}_{old}$ zurück in Weltkoordinaten transformiert. Danach werden alle dieser Punkte mit der aktuellen Model-View-Projection-Matrix $\boldsymbol{M}_{new}$ für I_w transformiert. Diese einfache Herangehensweise sorgt allerdings für undefinierte Stellen im Zielbild, wie in Abbildung 18 zu sehen ist.

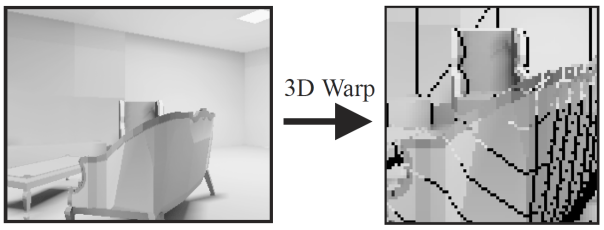

Abbildung 18: Durch Warping auftretende Löcher bei unterabgetasteten Oberflächen [12]

Um dieses Problem zu beheben, behandeln Mark et al. das Bild als Mesh. Das heißt, für jeden Pixel wird ein Vertex erzeugt, die zusammen das komplette Mesh ergeben. Auf die Vertices des Meshes werden dann die oben erwähnten Transformationen mit $\boldsymbol{M}_{old}$ und $\boldsymbol{M}_{new}$ angewendet. Den Vertices wird die Farbe des Pixels zugeteilt welches sie repräsentieren, dann wird die Szene gerendert. Auf diese Weise werden entstehende Löcher von dem Mesh abgedeckt und die Farbwerte zwischen den Vertices interpoliert.

Ein Problem welches auf diese Weise ensteht ist die Entstehung von Falten. Dies geschieht, wenn sich durch die Änderung der Perspektive mehrere Vertices zusammen schieben, so dass sie entweder deutlich näher zusammenfallen als zuvor oder sogar auf dem gleichen Pixel des Zielbildes liegen. In diesem Fall wird zuerst versucht über die Tiefe der Vertices den geeigneten Farbwert zu finden. Ist dies nicht möglich weil sie aufeinander liegen, wird ein sogenannter Zuversichtswert (*confidence value*) berechnet. Dieser besteht aus dem Verhältnis der Raumwinkel des Pixels in Ausgangs- und Zielbild,

welche abhängig von der Orientierung der Oberfläche sind zu der das Pixel gehört.

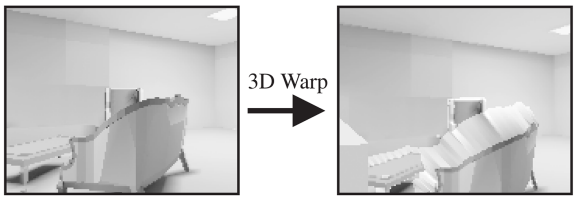

Abbildung 19: Durch reine Meshtransformation auftretende Gummibänder zwischen Vorder- und Hintergrund [12]

Ein weiteres Problem durch die Transformation eines Meshs sind die so genannten Gummibänder. Da es sich um ein einziges Mesh handelt, sind Objekte im Vordergrund an ihren Kanten mit dem Hintergrund verbunden. Ändert sich nun der Standpunkt in der Szene, wird vorher verdeckter Hintergrund sichtbar. Da diese Stellen undefiniert sind, sieht man nun die Verbindung zwischen Vordergrund- und Hintergrundobjekten (Abbildung 19) Um diese Problemstellen zu erkennen, wird eine Verbundenheit (*connectedness*) berechnet. Im Gegensatz zu dem Zuversichtswert für Faltenerkennung wird diese pro Dreieck berechnet. Hierbei fließen die Oberflächennormale sowie die Abstände der aufspannenden Vertices in die Berechnung ein. Auf diese Weise können stark abknickende Oberflächen erkannt und als potentielle Fehler in der neuen Projektion eingestuft werden. Damit nicht auch tatsächlich stark abknickende Oberflächen wie z.B. eine Tischkante in diese Kategorie fallen, werden zudem die Tiefenwerte der Vertices verglichen. Liegen diese nahe beieinander ist anzunehmen, dass sie trotz starker Abknickung zu der selben Oberfläche gehören.

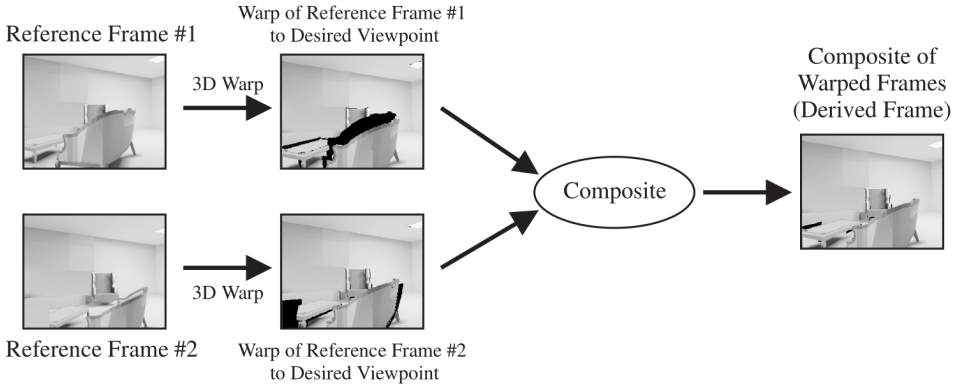

Abbildung 20: Warping und Kompositing zweier Referenzbilder [12]

Nachdem Gummibänder erkannt wurden, müssen entsprechende Stellen im Zielbild mit sinnvollen Farbinformationen gefüllt werden. Hier für sind

verschiedene Vorgehensweisen möglich. Mark et al. bauen ihre Arbeit darauf auf, dass zwei verschiedene Perspektiven der Szene gerendert werden, die z.B. durch das Verschieben der Kamera bei einer Bewegung durch die Szene enstehen. Das Verfahren soll nun zwischen diesen Referenz- oder Schlüsselbildern einen Übergang schaffen, um eine höhere Bildfrequenz zu erzielen oder die Latenz zu reduzieren. Mark et al. konstruieren daher aus beiden Schlüsselbildern die aktuelle Sichtweise auf die Szene, wobei die beiden dadurch abgeleiteten Bilder miteinander kombiniet werden. Da durch die unterschiedlichen Ausgangsblickpunkte unterschiedliche Stellen verdeckt bzw. sichtbar sind, können große Teile der undefinierten Stellen mit korrekten Bildinformationen ersetzt werden (Abbildung 20). Die verbleibenen Stellen werden durch die Verwendung einer Heuristik gefüllt. In dieser wird angenommen, dass der am weitesten entfernt liegende Vertex zu der Oberfläche gehört, die an dieser Stelle den Hintergrund bildet. Die Oberfläche wird dann fortgesetzt, indem die Farbe dieses Vertex für alle nicht definierten Stellen verwendet wird (Abbildung 21).

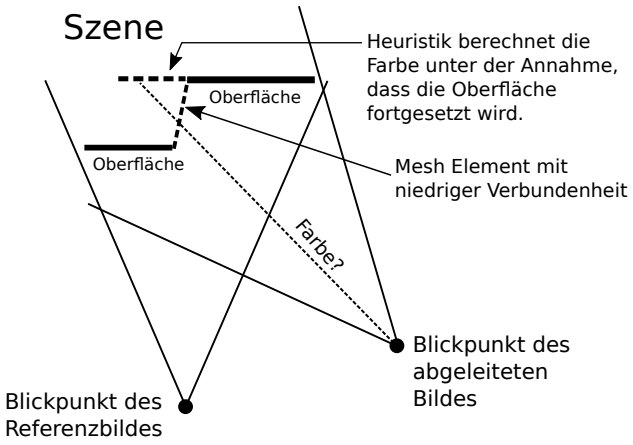

Abbildung 21: Verwendung einer Heuristik zur Farbermittlung von Dreiecken mit niedriger Verbundenheit [12]

Übertragen auf die Verwendung zur Generierung eines stereoskopischen Bildes, lässt sich dieses Verfahren als Ergänzung zu den Stereo-Panoramen implementieren. So könnte man die beiden Panoramen für linkes und rechtes Auge als die beiden Referenzbilder verwenden. Da die Änderung des Blickwinkels nur gering ist, könnte das bereits ausreichen, um entstandene undefinierte Stellen im finalen Bild zu füllen. Eine Anwendung auf Stereo-Panoramen würde deren Nachteil beheben, dass der Stereo-Effekt bei zu starker Kopfneigung verschwindet. Dies führt allerdings auch dazu, dass auf dem Anzeigegerät ein deutlich aufwendigerer Algorithmus implementiert werden muss, der auch eine entsprechende Rechenleistung benötigt.

Denkbar ist zudem, durch Gummibänder entstehende undefinierte Stellen

wie beim Iterative Image Warping anschließend mittels Hierarchical Hole Filling zu schließen. Hierdurch werden wie bereits erwähnt aber insbesondere größere undefinierte Stellen allerdings verwaschen wirken.

Vorteile dieses Verfahrens sind, dass es sich vorwiegend um die Transformation eines Meshs handelt, eine Standardaufgabe für jede Grafikpipeline. Da mit einem durchgehenden Mesh gearbeitet wird, entstehen keine undefinierten Stellen. Das Problem der Verbindung von Vorder- und Hintergrundobjekten kann erkannt und mittels einfacher Heuristiken behoben werden.

Ein ganz klarer Nachteil der Methode ist die enorme Größe des entstehenden Meshs. Da im Idealfall dessen Anzahl an Vertices der Auflösung des Bildes entspricht, müssen bei heutigen Bildschirmauflösungen schnell mehrere Millionen Vertices auf diese Weise erzeugt und verarbeitet werden. Dies wird insbesondere auf mobilen Endgeräten mit eingeschränkter Rechenleistung Probleme bereiten. Wählt man die Auflösung des Meshs niedriger, erzeugt also beispielsweise nur für jeden zweiten Pixel einen Vertex, werden insbesondere Probleme wie die Gummibänder deutlich verstärkt und werden im Endergebnis vermutlich deutlich stärker auffallen. Wie bei allen Reprojektionsverfahren ist das Maß der Blickpunktänderung natürlich stark eingeschränkt, da ansonsten zu viele undefinierte Stellen auftreten.

6 Aufbau eines AR-Brillen-Simulators

Im Nachfolgenden Kapitel wird der entstandene Demonstrator zur Simulation von AR-Brillen vorgestellt. Dieses wird in drei Abschnitte eingeteilt. Zuerst wird die Erstellung von (Stereo-)Panoramen erläutert. Darauf folgt die Vorstellung des Editors und dessen Möglichkeiten verschiedene Szenen, Konfigurationen für AR-Brillen und AR-Inhalte zu erstellen und konfigurieren. Abschließend wird die Verwendung des Demonstrators auf dem Zielsystem gezeigt. Alle Anwendungen wurden in Unity 5.4.0 entwickelt.

6.1 Erstellung von (Stereo-)Panoramen als Szenenhintergrund

Für die Erzeugung eines 360°-Panoramen wurde ein eigenständiges Projekt in Unity erstellt um zu verhindern, dass große VR-Szenen im eigentlichen Demonstrator liegen und ebenfalls auf das Smartphone geladen werden. Da die Erstellung der Umgebungen eigentlich nicht Bestandteil dieser Arbeit war, wurde nach bestehenden Lösungen hierfür recherchiert. Dabei wurde ein kostenfreies Tool im Unity Asset Store gefunden [32], welches diese als einfache sowie stereo-Variante ausgeben kann.

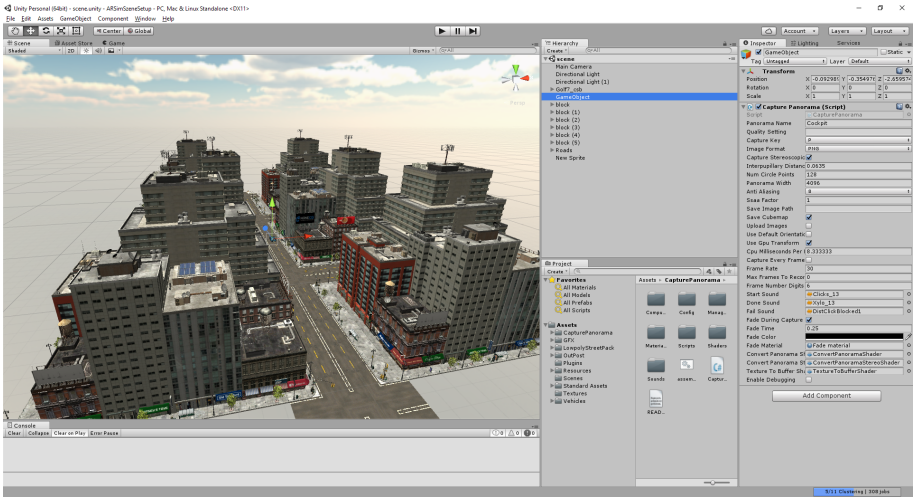

Abbildung 22: Erstellung von 360°-Panoramen

Für den Aufbau der Szene wird der standard Unity-Editor verwendet (Abbildung 22). Über diesen können die benötigten Ressourcen wie gewohnt importiert und in der Szene platziert werden. Die Hauptkamera der Szene muss an die Stelle positioniert werden, von der aus das Panorama erstellt werden soll. Auf eines der Objekte muss das “Capture Panorama”-Script gelegt werden. Dieses wird automatisch ausgeführt sobald die Anwendung in Unity gestartet wird. Um die Erstellung neuer Szenen zu vereinfachen, wurde das Script auf ein separates Objekt gelegt, welches als so genanntes

Prefab im Projektordner abgelegt wurde. Erstellt man eine neue Szene, kann das Prefab einfach in die Szene gezogen werden, wodurch das Objekt samt fertig konfiguriertem Script darin erstellt wird.

Wählt man das Object mit dem Script aus, kann man Einstellungen für die Panoramaerstellung vornehmen (Abbildung 22, rechte Spalte). Dazu gehören unter Anderem das Bildformat, die Auflösung, ob ein normales oder stereoskopisches Panorama erstellt werden soll sowie diverse Qualitätseinstellungen. Derzeit ist die maximal mögliche Breite auf 4096 Pixel für ein stereo und auf 8192 Pixel für ein einfaches Panorama beschränkt.

Sind Szene und Script fertig gestellt, started man die Unity-Anwendung über den Play-Button. Sobald die Szene geladen ist, kann man über das Drücken einer (ebenfalls einstellbaren) Taste die Erstellung des Panoramen starten. Für diese Dauer wird das Szenenfenster komplett schwarz gezeigt. Wie in Kapitel 5.1 erklärt werden nun zuerst die benötigten Einzelbilder gerendert und schließlich zu dem fertigen Panorama zusammengesetzt. Einzelbilder sowie Panorama werden im Hauptverzeichnis des Unity-Projekts abgelegt, die Einzelbilder können nach Abschluss des Vorgangs direkt gelöscht werden. Ein Beispiel für ein fertiges Panorama zeigt Abbildung 23.

Abbildung 23: Fertiges stereo Panorama, oben: linkes Auge, unten: rechtes Auge

Um ein generiertes Stereopanorama im Demonstrator verwenden zu können, müssen die Bilder noch mit einem Bildbearbeitungsprogramm separiert werden. Die obere Hälfte enthält die Darstellung für das linke, die untere für das rechte Auge. Die fertigen Panoramen können anschließend in den

Demonstrator importiert werden.

6.2 Aufbau und Konfiguration des Demonstrators

Wie bereits bei der Erstellung der Hintergrund Panoramen wird auch der eigentliche Demonstrator direkt im Unity-Editor angepasst. Deswegen wird an dieser Stelle zuerst eine kurze Einführung in das von Unity verwendete Vokabular gegeben.

Abbildung 24: Die Oberfläche des Editors für den Demonstrator

Abbildung 24 zeigt den Editor wie er für den Demonstrator angelegt wurde. In Fenster 1 befindet sich der *Szeneneditor,* in welchem die Objekte positioniert werden können. In Unity werden alle Elemente einer Szene als *Game Objects* bezeichnet. Ein Game Object kann verschiedene Komponenten besitzen und verfügt zumindest über eine Transformationskomponente. Alle in einer Szene vorhandenen Game Objects werden in der *Hierarchy* (2) aufgelistet, die den Szenengraphen darstellt. Die Objekte können hier beliebig angeordnet und verschachtelt werden. Wählt man ein Objekt aus, bekommt man im *Inspector* (3) alle daran gebundenen Komponenten angezeigt. Hier kann man dem Objekt neue Komponenten hinzufügen sowie bestehende bearbeiten oder löschen. Über das *Project* Fenster (5) lassen sich alle für die Anwendung benötigten Ressourcen, genannt *Assets*, verwalten. Eine Besonderheit sind die bereits erwähnten *Prefabs*. Unity bietet die Möglichkeit, Game Objects, die sich in der Szenenhierarchie befinden, per Drag & Drop zurück in die Assets zu ziehen. Dadurch wird ein Ressourcenpaket erstellt, welches das Game Object selbst, alle ihm untergeordneten Game Objects sowie alle darauf liegenden Komponenten samt derer aktuellen Konfiguration enthält. Fügt man ein Prefab in eine Szene ein, werden alle darin

enthaltenen Game Objects samt Komponenten wiederhergestellt. Auf diese Weise lassen sich auch komplexere Objekte einfach zwischen Szenen oder Objekten austauschen.

6.2.1 Struktur einer Szene

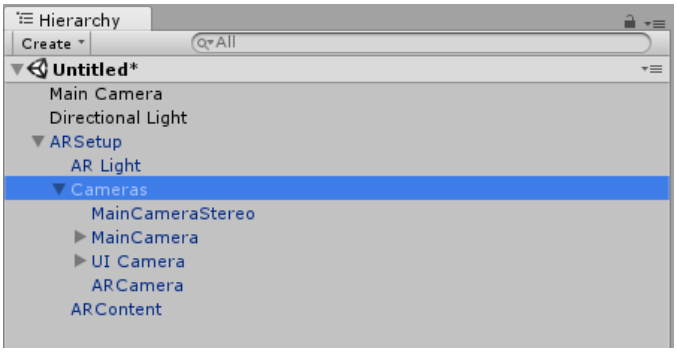

Abbildung 25: Der grundlegende Szenengraph

Beginnt man eine neue Szene, fügt man als erstes das ARSetup Prefab hinzu. Damit werden alle benötigten Objekte samt Komponenten automatisch erstellt (Abbildung 25). Alle für die AR-Simulation benötigten Objekte sind zur besseren Übersicht unter dem "ARSetup" Game Object gruppiert. Dazu gehört zumindest eine Lichtquelle, verschiedene Kameras sowie die eigentlichen AR-Inhalte. Sollte eine VR-Umgebung direkt gerendert werden statt ein Panorama zu verwenden, können die verwendeten Objekte beliebig außerhalb des AR-Setups angeordnet werden.

Es wird zumindest ein Licht zum Rendern der AR-Objekte benötigt. Dafür können auch die bereits in der Szene vorhandenen Lichter verwendet werden. Um eine höhere Flexibilität zu gewährleisten, wurde die Möglichkeit einer separaten AR-Beleuchtung eingeführt. Damit diese nur AR-Objekte betrifft, muss die Culling-Maske der dazugehörigen Lichtquellen auf ausschließlich "ARElements" eingestellt werden. Bei "ARElements" handelt es sich um eine Ebene (*Layer*), der alle AR-Inhalte zugeordnet werden müssen. Bei Lichtquellen, die beispielsweise für VR-Inhalte benötigt werden und AR-Inhalte nicht betreffen sollen, muss diese Ebene aus der Culling-Maske genommen werden. Es können beliebig viele AR-Lichtquellen erzeugt und einzeln konfiguriert werden.

6.2.2 AR-Inhalte

AR-Objekte können beliebig mit dem Editor in der Szene platziert werden. Hier bei kann es sich um jegliche Art von Objekten handeln, die Unity in einer Szene darstellen kann. Um eine logische Grundstruktur zu schaffen wurde ein

Game Object "ARContent" angelegt, dem alle Inhalte untergeordnet werden können. Dieses befindet sich in der Hierarchie unter dem "ARSetup".

Je nach Art der Simulation können die Objekte verschieden in der Hierarchie platziert werden. Sollen die Objekte an festen Punkten in der Szene sitzen, kann die bisherige Struktur beibehalten und die AR-Objekte dem "ARContent" untergeordnet werden. Sollen sich die AR-Objekte mit der Kamera mitbewegen, also statische Einblendungen in der Brille zu sehen sein, kann man den "ARContent" in der "ARCamera" unterordnen. Beides kann beliebig kombiniert werden, um eine Mischung aus getrackten und statischen Elementen zu erhalten.

Wichtig bei der Erzeugung der AR-Objekte ist, dass sie der Ebene "AR-Elements" zugeteilt werden. Zum einen kann dies wie bereits erwähnt für die Beleuchtung interessant sein, zum anderen wird darüber gekennzeichnet, dass diese Objekte nicht über die normale Kamera, sondern die AR-Kamera gerendert werden.

6.2.3 Kameras

Unter dem Game Object "Cameras" sind alle benötigten Kameras gruppiert. Insgesamt werden vier verschiedene Kameras verwendet. Zwei für die stereoskopische Darstellung der VR-Welt, eine zum Rendern der AR-Inhalte und zuletzt eine für die UI. Es werden zuerst die beiden Hauptkameras gerendert, gefolgt von der AR-Kamera und zuletzt der UI-Kamera.

Kameras werden in Unity standardmäßig automatisch gerendert. Die Reihenfolge kann über die Kameratiefe im Inspector eingestellt werden. Besitzen mehrere Kameras die selbe Tiefe, wird nach der Anordnung in der Szenenhierarchie vorgegangen, von oben nach unten. Es ist möglich Kameras zu deaktivieren, wodurch sie nicht mehr automatisch gerendert werden. Der Rendervorgang kann jedoch in einem Script manuell angestoßen werden.

Die Hauptkameras sowie die UI-Kamera werden automatisch gerendert, die AR-Kamera ist deaktiviert. Das Rendern der AR-Inhalte erfolgt in einen so genannten *Image Effect*, über welchen Post-Processing in Unity durchgeführt wird. Hierbei handelt es sich um ein Script, das auf den beiden Hauptkameras liegt und in Kapitel 6.2.7 näher erläutert wird. Diese Vorgehensweise ist nötig, da Kameras direkt auf die Ausgabetextur gerendert werden und damit eine nachträgliches Post-Processing ausschließlich auf den AR-Inhalten nicht möglich ist. Es wäre zwar möglich die AR-Kamera in eine eigene Rendertextur rendern zu lassen, deren Größe muss jedoch bereits im Editor festgelegt werden und ist nicht variabel, zudem lassen sich nicht alle benötigten Effekte wie z.B. die Simulation von Latenz über diese Methode umsetzen.

Die Steuerung aller Kameras erfolgt automatisch durch das Tracken der Kopfrotation. Sobald in Unity VR aktiviert ist, wird vor jedem Rendervorgang die Rotation aller Kameras an die von der Gear VR gemeldeten Kopf-

rotation angepasst. Da dies unmittelbar vor dem Rendervorgang geschieht, ist es nicht möglich diese Werte manuell zu überschreiben. Das Bewegen innerhalb der Szene ist derzeit nicht vorgesehen und wurde nicht umgesetzt. Sollte dies in Zukunft gewünscht sein, beispielsweise in einer direkt auf dem Endgerät gerenderten 3D-Szene, reicht es aber den "Input Manager" (Kapitel 6.2.4) zu erweitern, so dass dieser bei entsprechenden Nutzereingaben das "Cameras" Objekt verschiebt.

Den beiden Hauptkameras ist eine Skybox-Komponente zugeordnet. Über diese können für jedes Auge ein eigenes Panorama zugewiesen werden. Auf diese Weise lässt sich leicht die Anzeige des stereoskopischen Panoramen realisieren. In den Kameras ist zudem das Zielauge eingestellt. Bei Erstellung einer neuen Kamera rendert diese standardmäßig für beide Augen. Da aber separate Bilder auf den Hintergrund gerendert werden müssen, ist je eine auf ausschließlich das linke und rechte Auge eingestellt.

6.2.4 Benutzereingaben

Um Benutzereingaben und die Anzeige von Informationen zu handhaben, wurde der "Input Manager" entwickelt (Abbildung 26). Da dieser eine globale Funktion übernimmt, liegt er als Komponente auf dem "Cameras" Game Object. Neben voreingestellten Werten für die Anzeige von Informationen können hier verschiedene Einstellungen bezüglich Benutzereingaben vorgenommen werden.

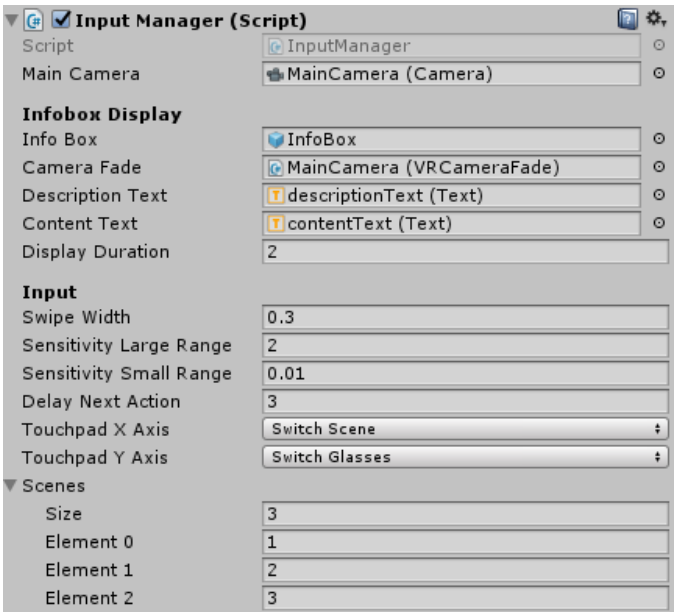

Abbildung 26: Der Input Manager

Die Steuerung innerhalb einer Szene erfolgt primär über das Touchpad

der Gear VR. Dieses stellt eine x und y-Achse bereit. Benutzereingaben sollen in Form von *Swipes* erkannt werden, also indem der Benutzer entweder horizontal oder vertikal über das Touchpad wischt. Neben den Swipes existiert noch ein "zurück"-Button auf der Gear VR, welcher dazu verwendet wird das Menü aufzurufen.

Das Touchpad der Gear VR ist etwas ungenau, weshalb die übertragenen Eingabewerte nicht direkt in Aktionen umgesetzt werden können. Stattdessen wird der Startpunkt eines Swipes registriert und eine Aktion erst dann ausgeführt, wenn auf einer Achse eine bestimmte Strecke zurückgelegt wurde. Wie groß diese ist und damit die Sensitivität lässt sich über die "Swipe Width" einstellen. Welche Aktion bei einem erfolgreichen Swipe ausgeführt wird, lässt sich den einzelnen Achsen zuweisen. Je nach Aktion werden weitere Einstellungen berücksichtigt. Erfolgt eine Änderung von Werten einer AR-Brille, wird abhängig von dem Wertebereich eine konfigurierbare Schrittweite verwendet. Soll die Szene oder die AR-Brille gewechselt werden, kann der nächste Wechsel verzögert werden, damit nicht aus Versehen mehrere Wechsel direkt hintereinander erfolgen.

Um zwischen Szenen wechseln zu können, müssen an zwei Stellen Anpassungen vorgenommen werden. Zuerst müssen alle Szenen in die Unity *Build settings* aufgenommen werden. Danach kann man die IDs der Szenen unter "Scenes" im "Input Manager" eintragen. Das Menü muss hierbei immer an erster Stelle (Szenen-ID 0) stehen, damit es bei Programmausführung als erste Szene geladen wird. Daher kann es automatisch angesprochen werden und wird nicht in die Szenen eingetragen. Da der "Input Manager" derzeit pro Szene arbeitet, muss diese Anpassung in sämtlichen Szenen vorgenommen werden. Im Menü müssen die Szenen ebenfalls einmalig eingetragen werden, was in Kapitel 6.2.8 beschrieben wird.

6.2.5 Verwaltung der AR-Konfigurationen

Der "AR Configuration Loader" dient der Verwaltung der simulierbaren AR-Brillen. Da dieser eine globale Funktion übernimmt, liegt er als Komponente auf dem "Cameras" Game Object. Er benötigt hierfür Zugriff auf die Hauptkameras sowie spezielle Materialien zum Rendering der AR-Inhalte, die im Inspector referenziert werden müssen (Abbildung 27). Er übernimmt alle Funktionen, die zum Laden, Ändern, Speichern, Löschen und Wechseln der AR-Konfiguration nötig sind. Die aktuell verwendete Konfiguration wird szenenübergreifend gespeichert und bleibt daher auch bei einem Szenenwechsel erhalten.

Beim Starten einer Szene lädt er die verfügbaren AR-Konfigurationen. Ein Vorgang der davon abhängig ist, ob das Programm im Unity-Editor oder der Gear VR ausgeführt wird.

Bei der Ausführung aus dem Editor werden die Einstellungen von der "Main Camera" geladen. Dort liegen sie in Form von Scripten, je eins pro

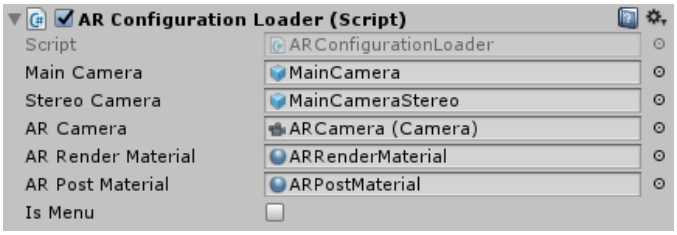

Abbildung 27: Der AR Configuration Loader

Konfiguration. Außerdem können im Editor mit der Tastenkombination Alt+s die Konfigurationen für die spätere Ausführung auf der Gear VR gespeichert werden. Hierbei wird eine XML-Datei erzeugt, welche in den Assets als *Streaming Asset* abgelegt wird (Listing 1). Streaming Assets sind Dateien, auf die später unter Android ein normaler Lesezugriff erfolgen kann. Dies ist nötig um das XML verarbeiten zu können, da Unity außer über `C#`-Programmierung (Linq) keine Möglichkeit dazu bietet. Man kann die Datei also nicht als Standard Asset einem Script über den Inspector verlinken. Wird die Anwendung auf einer Gear VR ausgeführt, wird statt den Scripten von der Kamera das XML geladen. Da über das Menü Änderungen an den Konfigurationen vorgenommen werden können, wird eine lokale Kopie angelegt. Eine direkte Änderung von Streaming Assets ist nicht möglich. Bei jedem Ladevorgang wird über einen im XML vorhandenen Timestamp geprüft, ob die Version in den Streaming Assets neuer ist als die lokale. Ist dies der Fall, wird das lokale XML mit dem neuen überschrieben.

```
<root>
  <version>1474119601</version>
  <config name="ARGlasses1">
    <brightness>0</brightness>
    <contrast>0</contrast>
    <transparency>1</transparency>
    <darkening>0.8</darkening>
    <resolutionx>1920</resolutionx>
    <resolutiony>1080</resolutiony>
    <topcornerx>0.2</topcornerx>
    <topcornery>0.2</topcornery>
    <width>0.5</width>
    <height>0.5</height>
    <latency>0</latency>
  </config>
</root>
```

Listing 1: XML zur Speicherung der AR-Konfigurationen

Unabhängig davon wo das Programm ausgeführt wird, erfolgt neben dem Laden der Daten eine Initialisierung der Szene. An dieser Stelle werden zuerst alle AR-Rendering-Scripte von Haupt- und Stereokamera entfernt, bevor auf beide Kameras ein einzelnes gelegt wird. An diesem werden beim Umschalten

der AR-Brillen lediglich die Einstellungen angepasst.

6.2.6 Rendern der Szene

Um den Rendervorgang der VR-Welt zu beeinflussen, wurde ein "Render Manager" entwickelt, der als letztes Script auf dem "Cameras" Game Object liegt (Abbildung 28).

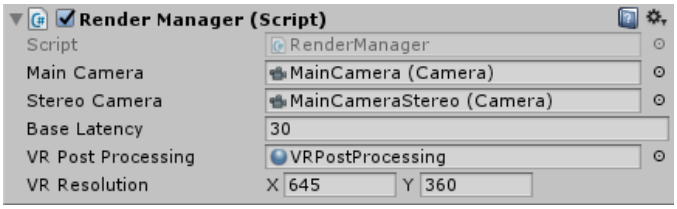

Abbildung 28: Der Render Manager

Derzeit verfügt er über zwei Funktionen, eine künstliche Latenz für den VR-Hintergrund zu erzeugen sowie dessen Auflösung zu reduzieren. Da wie bereits erwähnt eine direkte Manipulation der Kamerarotation bei Verwendung von VR nicht möglich ist, muss diese in Unity eingebaute Sicherheit umgangen werden. Dies lässt sich bewerkstelligen, indem ein übergeordnetes Objekt statt der Kamera selbst manipuliert wird. Statt also lediglich die Rotation der Kamera etwas zu verzögern, muss die durchgeführte Rotation über ein gegengesteuerte Rotation des Elternobjekts ausgeglichen werden.

```
// add rotation of current frame to queue
rotQueueEntry newRotation;
newRotation.timestamp = Time.time;
newRotation.rotation = InputTracking.GetLocalRotation(VRNode.
    CenterEye);
rotationQueue.Enqueue(newRotation);

// now counter rotate the parent to compensate for the head
    rotation.
// this will simulate an input lag
rotQueueEntry rot = rotationQueue.Peek();

Quaternion inputRot = InputTracking.GetLocalRotation(VRNode.
    CenterEye);
Quaternion relative = Quaternion.Inverse(inputRot) * rot.
    rotation;
this.transform.rotation = relative;

if (rot.timestamp < (Time.time - delay))
{
   rotationQueue.Dequeue();
}
```

Listing 2: Gegenrotation zur Erzeugung künstlicher Latenz

Dazu wird in zwei Schritten vorgegangen (Listing 2). Zuerst wird die aktuelle Rotation der Kamera abgerufen und zusammen mit einem Timestamp in einer Queue gespeichert. Ist beispielsweise eine Latenz von 30ms eingestellt, befinden sich in dieser Queue alle Rotationen die innerhalb der letzten 30ms statt gefunden haben, bevor diese abgerufen und tatsächlich ausgeführt werden.

Als zweiter Schritt muss das Elternobjekt soweit zurückrotiert werden, dass die Kameraposition der entspricht, wie sie zum Zeitpunkt der Verzögerung gewesen wäre. Dazu wird der oberste Eintrag der Queue abgerufen, der diese Information als Quaternion enthält. Das Inverse des Quaternions der aktuellen Rotation multipliziert mit der zurückliegenden Rotation liefert den relativen Wert, um den das Elternobjekt entgegengesetzt rotiert werden muss. Zuletzt muss der oberste Eintrag der Queue entfernt werden, sobald die gewünschte Latenz erreicht wurde.

Die künstliche Reduktion der Auflösung wird nicht direkt im "Render Manager" durchgeführt, da dieser vor dem Rendervorgang läuft. Stattdessen wird es als Post-Processing Schritt in dem Image Effect durchgeführt, der auch die AR-Inhalte rendert (Kapitel 6.2.7). Der "Render Manager" setzt lediglich die benötigten Informationen für den Image Effect.

6.2.7 Rendern der AR-Inhalte

Das Rendern der AR-Inhalte erfolgt im Post-Processing-Schritt als Image Effect. Nachdem der Rendervorgang einer Kamera abgeschlossen wurde, wird von allen auf dieser liegenden Scripten die `OnRenderImage()` Funktion aufgerufen, insofern sie existiert. Zu diesem Zeitpunkt ist das fertig gerenderte Bild bereits vorhanden und kann weiterverarbeitet werden. Das Post-Processing läuft in mehreren Schritten ab, die nachfolgend erläutert werden und verschiedene Bildebenen repräsentieren.

Zuerst erfolgt das verbleibende Post-Processing, welches nicht direkt im "Render Manager" (Kapitel 6.2.6) durchgeführt wurde. Aktuell handelt es sich dabei lediglich um die Anpassung der Auflösung. Zu diesem Zweck wird ein Shader (Listing 3) über das Bild laufen gelassen. Dieser verwendet einen standard Vertex-Shader, so dass lediglich der Fragment-Shader noch Berechnungen vornimmt. Der Shader bekommt das gerenderte Bild als Textur sowie die gewünschte Auflösung übergeben. Mit der Auflösung werden die UV-Koordinaten neu berechnet. Dabei werden die Werte des Augangsbildes auf das Zentrum der am nächsten liegenden Pixel abgebildet und die entsprechende Farbe zurückgegeben.

```
CGPROGRAM
#pragma vertex vert_img
#pragma fragment frag

#include "UnityCG.cginc"
```

```
uniform sampler2D _MainTex;
uniform float4 _Pixels;

float4 frag(v2f_img i) : COLOR
{
  // pixelation effect to emulate lower resolution displays
  float2 pixeledUV = round(i.uv * _Pixels.xy + 0.5f) / _Pixels.
      xy;

  // get color from textures
  float4 result = tex2D(_MainTex, pixeledUV);

  return result;
}
ENDCG
```

Listing 3: VR Post-Processing Shader

Als nächster Schritt erfolgt die Simulation der Latenz. Diese erfolgt analog zu der im "Render Manager". Da die AR-Kamera manuell gerendert wird, kann die gewünschte Rotation direkt in die Kamera geschrieben werden und es muss nicht auf eine Gegenrotation zurückgegriffen werden. Es wird also zuerst die korrekte Rotation passend zu der gewünschten Latenz gesetzt und danach der Rendervorgang angestoßen. Das Bild wird dabei zuerst in eine Rendertextur geschrieben.

Nachfolgend wird zuerst die Abdunklung der AR-Brille auf das Bild berechnet und anschließend die AR-Inhalte darüber gelegt. Für die Abdunklung muss der auszugebende Farbwert lediglich mit einem Wert zwischen 0 und 1 multipliziert werden (Listing 4), wobei 1 für keine und 0 für voll Abdeckung steht. Beim Kopieren der AR-Inhalte auf das fertige Bild werden neben der Anpassung der Auflösung ebenfalls Helligkeit, Kontrast sowie Transparenz angepasst(Listing 5).

```
CGPROGRAM
#pragma vertex vert_img
#pragma fragment frag

#include "UnityCG.cginc"

uniform sampler2D _MainTex;
uniform float _Darkening;
uniform float _TopX;
uniform float _TopY;
uniform float _BotX;
uniform float _BotY;

float4 frag (v2f_img i) : COLOR
{
  float4 c = tex2D(_MainTex, i.uv);
  float4 result = c;

  // only apply effect on AR display area
```

```
  if (i.uv.x > _TopX && i.uv.x < _BotX && i.uv.y < (1.0f - _TopY
      ) && i.uv.y >(1.0f - _BotY))
  {
    result.rgb *= _Darkening;
  }
  return result;
}
ENDCG
```

Listing 4: AR Post Processing Shader

```
CGPROGRAM
#pragma vertex vert_img
#pragma fragment frag

#include "UnityCG.cginc"

uniform sampler2D _MainTex;
uniform sampler2D _ARTex;
uniform float _bwBlend;
uniform float4 _Pixels;
uniform float _Brightness;
uniform float _Contrast;
uniform float _TopX;
uniform float _TopY;
uniform float _BotX;
uniform float _BotY;

float4 frag(v2f_img i) : COLOR
{
  // pixelation effect to emulate lower resolution displays
  float2 pixeledUV = round(i.uv * _Pixels.xy + 0.5f) / _Pixels.
      xy;

  // get color from textures
  float4 mainColor = tex2D(_MainTex, i.uv);
  float4 arColor = tex2D(_ARTex, pixeledUV);
  float4 result = mainColor;

  // show AR objects only in defined areas
  if (arColor.a > 0.5f && i.uv.x > _TopX && i.uv.x < _BotX && i.
      uv.y < (1.0f - _TopY) && i.uv.y >(1.0f - _BotY))
  {
    arColor.rgb = ((arColor.rgb - 0.5f) * _Contrast) + 0.5f;
    arColor.rgb *= _Brightness;
    result.rgb = lerp(mainColor.rgb, arColor.rgb, _bwBlend);
    result.a = arColor.a;
  }
  return result;
}
ENDCG
```

Listing 5: AR Render Shader

Sind all diese Vorgänge abgeschlossen, ist das Bild fertig und kann ausgegeben werden. Ein Beispiel für eine fertig gerenderte Ansicht ist in Abbildung 29 zu sehen.

Abbildung 29: Die fertig gerenderte Ausgabe

6.2.8 Menü

Im Gegensatz zum Konzept musste das Menü überarbeitet werden. Zum einen sind Auflösung und Platz in der VR-Darstellung zu stark eingeschränkt, zum anderen konnten Umsetzung sowie Handhabung durch den Nutzer dadurch vereinfacht werden. Statt das Menü einzublenden wurde nun eine eigenständige Menüszene geschaffen.

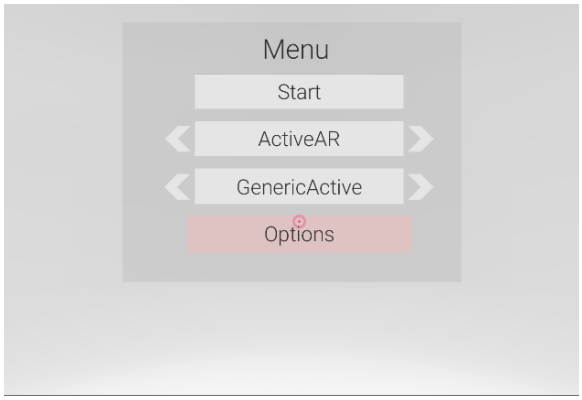

Abbildung 30: Das überarbeitete Menü

Diese ist der Ausgangspunkt nachdem die Anwendung gestartet wurde und der Benutzer kann jeder Zeit dorthin zurück kehren. Das neue Menü ist

in Abbildung 30 zu sehen und wurde in seiner grundlegenden Funktionalität den Unity VR-Beispielen entnommen.

Über das Menü können eine Startszene sowie Start-AR-Brille gewählt und die Simulation gestartet werden. Außerdem lassen sich die AR-Brillen konfigurieren. Hierbei ist es möglich neue Konfigurationen anzulegen und bestehende zu ändern sowie zu löschen. Änderungen werden wie in Kapitel 6.2.5 beschrieben lokal auf dem Gerät abgelegt.

Die Steuerung des Menüs erfolgt über Kopfbewegungen und Touchpad. Per Raycaster wird ein Strahl in die Mitte des Bildes geschossen. Dort wird ein roter Punkt als Cursor angezeigt. Schneidet der Strahl des Raycasters ein interaktives Element, wird dieses rot eingefärbt, vergrößert und um den Cursor wird ein Kreis angezeigt. Der Benutzer kann seine Auswahl bestätigen, indem er auf das Touchpad drückt. Um eine versehentliche Selektion zu verhindern, muss es eine kurze Zeitspanne gedrückt bleiben. In dieser Zeit läuft ein Fortschrittsbalken in dem Kreis. Sobald der Kreis komplett gefüllt ist, wird der Button aktiviert. Die Dauer des Gedrückthaltens ist konfigurierbar.

Wie bereits in Kapitel 6.2.4 angesprochen, müssen auch im Menü die Szenen angepasst werden. Dazu müssen in der Hierarchy die Game Objects "MenuItemScenesPref" und "MenuItemScenesNext" unter "GUI" - "MainMenu" ausgewählt werden (Abbildung 31). Diese verfügen über eine "Menu Button Main"-Komponente, in welcher die verfügbaren Szenen eingetragen werden müssen. An dieser Stelle muss der tatsächliche Name der Szene eingetragen werden, damit dieser im Menü angezeigt wird.

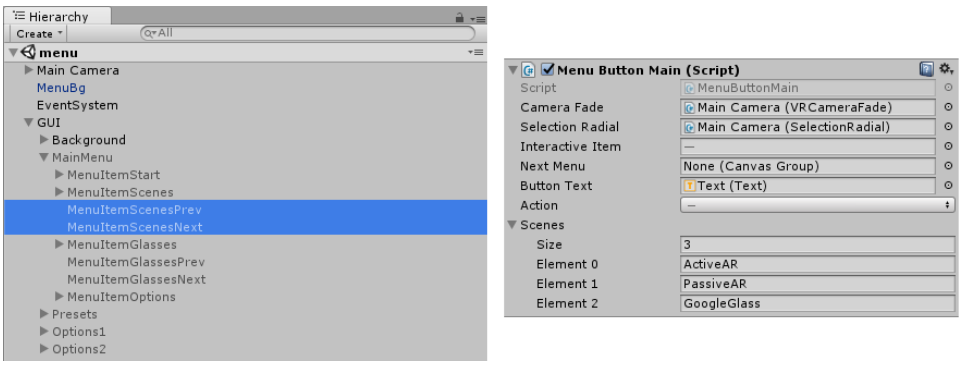

Abbildung 31: Konfiguration der Szenen im Menü

6.2.9 Post Rendering 3D Warp

Das zuvor präferierte Verfahren des Post Rendering 3D Warps konnte im Rahmen dieser Arbeit leider nicht komplett umgesetzt werden. Hierbei spielten zeitliche und insbesondere technische Einschränkungen eine wesentliche Rolle. In diesem Abschnitt soll kurz darauf eingegangen werden, welcher Status bei der Entwicklung erreicht werden konnte.

Es wurde ein Script “BackgroundMesh.cs” geschrieben, welches dynamisch ein Mesh inklusive UV-Koordinaten in beliebiger Auflösung generiert. Zudem wurde ein Shader geschrieben, der es ermöglicht dieses als Screen-Filling-Quad anzuzeigen und eine Textur im Equirectangular-Format darauf zu projizieren. Diese Funktionalität konnte in einem Test auch bestätigt werden. Bereits in diesem Rahmen sind allerdings bereits erste technische Einschränkungen und Probleme aufgefallen.

Eine Bildschirmhälfte der Gear VR verfügt über eine Auflösung von 1280x1440 Pixel. Für ein optimales Ergebnis müsste also ein Mesh erstellt werden, welches über ca. 1,8 Millionen Vertices verfügt. Durch diese hohe Anzahl an Vertices war eine flüssige Darstellung nicht mehr möglich. Auch bei Reduzierung auf die Hälfte der Vertices waren insbesondere bei schnellen Bewegungen noch Ruckler wahrnehmbar. Da das Verfahren im Idealfall mit maximal möglicher Auflösung ausgeführt wird, sind entsprechende Qualitätseinbußen zu erwarten.

Neben den Einschränkungen durch die Hardware sind allerdings auch softwareseitige aufgefallen. So kann eine Tiefentextur, die im Idealfall direkt die Weltpositionen für jeden Pixel enthält, nicht direkt als solche gespeichert werden. Eine normale RGBA-Textur verfügt nicht über einen ausreichenden Werteraum, Floatingpoint-Texturen wurden bei Tests nicht durch Android unterstützt. Damit bleiben nur in der Umsetzung aufwendige Alternativen über. So könnten die Tiefenwerte in mehrere RGBA-Texturen aufgeteilt werden, die später im Shader entsprechend auch wieder zusammengesetzt werden müssten. Alternativ dazu könnten die Tiefenwerte in einem eigenen Dateiformat gespeichert und ausgelesen werden. Bei beiden Möglichkeiten ist allerdings nicht absehbar, ob der auf einer Gear VR verfügbare Grafikspeicher für derartige Operationen ausreicht. Derzeit wird als empfohlende maximale Texturgröße für Android-Endgeräte eine Auflösung von 2048 x 2048 angegeben. Um die Umgebung möglichst hochauflösend darstellen zu können, wurde bereits mit zwei 4096 x 2048 Texturen gearbeitet. Unity selbst erlaubt Texturen mit maximal 4096 x 4096 Pixeln.

7 Evaluation

Zum Abschluss der Entwicklung wurde eine Evaluation durchgeführt. Gegenstand dieser war die Frage danach, ob die Simulation von AR-Brillen authentisch umgesetzt werden konnte. Daher konzentriert sie sich auf die Verwendung der finalen Anwendung durch einen Benutzer, nicht auf die Erstellung und Konfiguration von Inhalten im Editor. Wie bereits bei den Zielsetzungen erwähnt handelt es sich um eine Expertenevaluation, in der insgesamt sieben Personen in verschiedene Szenarios versetzt und nach ihrem Subjektiven Eindruck befragt wurden. Im Nachfolgenden werden erst der Aufbau des Tests, danach die Durchführung und letztendlich die Auswertung vorgestellt.

7.1 Aufbau

Für die Evaluation wurden mehrere Szenarien entworfen, die verschiedene Anwendungsfälle der Simulation aufgreifen. Hierbei handelt es sich um die zwei allgemeinen Fälle, Video see-through AR und generisches optisches see-through AR, sowie die Simulation einer Google Glass und Microsoft HoloLens als spezifische Anwendungsfälle.

Die Darstellung der allgemeinen Fälle wurden aus mehreren Gründen gewählt. Zum einen stellen sie die freieste Form der Simulation dar und können die generelle Funktionalität dieser demonstrieren, ohne sich an die Vorgaben eines existierenden technischen Systems festlegen zu müssen. Zum anderen zeigen sie die Möglichkeit, dass man jede beliebige vorstellbare AR-Brille simulieren kann. Damit ist es nicht nur möglich, augmentierte Inhalte an eine spezielle Hardware anzupassen, sondern auch eine optimale AR-Brille für den gewünschten Zweck zu finden und mit diesen Informationen die am besten geeignete Hardware auszuwählen. Außerdem kann durch eine generische Simulation verschiedener AR-Szenarien die Testperson an das Thema heran geführt werden, bevor diese einen direkten Vergleich durchführen muss.

Die spezifischen Anwendungsfälle wurden gewählt, damit die gezielte Simulation existierender Systeme geprüft werden kann. Im Gegensatz zu den allgemeinen Fällen steht hier nicht die allgemeine Machbarkeit einer AR-Simulation im Vordergrund, sondern der direkte Vergleich. Da Google Glass und die Microsoft HoloLens die bekanntesten Vertreter ihrer Art sind und außerdem durch ihre unterschiedliche Funktionsweise ein großes Anwendungsspektrum abdecken, wurde sich für diese als Testsysteme entschieden.

Sinnvolle Szenarien in denen die AR-Konfigurationen getestet werden konnten wurden bereits im Konzept vorgestellt und orientieren sich an dem Anwendungskontext, in welchem diese Arbeit erstellt wurde. Dies sind die Cockpit-Ansicht eines Autos, das Innere einer Fabrikhalle sowie eine Außenansicht.

Im Rahmen der Tests wurde sich aus verschiedenen Gründen für die Umsetzung der Cockpit-Ansicht entschieden. Zum einen konnte VW ein detail-

liertes 3D Modell des Golf 7 bereitstellen. Des weiteren ließ sich mit Hilfe des Unity Asset Stores eine gute Umgebung zusammenstellen, ohne dass zeitaufwendig alles selbst modelliert werden musste. Außerdem stellt das Cockpit einen guten Anwendungsfall dar, für den sich sinnvolle AR-Einblendungen generieren ließen. Auf Grund zeitlicher Einschränkungen wurde sich auf dieses eine Testszenario beschränkt. Dies stellt jedoch keinen Nachteil dar, da es eine bessere Vergleichbarkeit der AR-Konfigurationen gewährleistet und den zeitlichen Aufwand für die Probanden auf einen überschaubaren Rahmen beschränkt.

Nachdem die zu testenden Systeme und Szenario ausgewählt wurden, konnte der Demonstrator entsprechend vorbereitet werden. Hierbei wurde zuerst eine möglichst detailreiche virtuelle Umgebung erstellt und als stereo Panorama vorgerendert. Danach wurden die dazugehörige AR-Inhalte erstellt, sowie die Konfiguration der AR-Brillen durchgeführt. Ein Beispiel für einen fertig gerenderten Szenenausschnitt wie ihn der Proband zu sehen bekommt, ist in Abbildung 32 dargestellt. Als AR-Inhalt wird ein *Head-up Display* (HUD) eingefügt. Nachfolgend werden die einzelnen Szenarien sowie deren Einstellungen kurz erläutert.

Abbildung 32: Konfiguration der Szenen im Menü

Um Video look-through AR zu simulieren, wird die Auflösung der VR-Umgebung reduziert und diese zusätzlich mit einer leichten Latenz versehen von 30ms. Damit soll der Blick auf einen Bildschirm simuliert werden statt direkt die Realität zu sehen. Da die Gear VR bereits diese Einschränkung aufweist, ist die finale Darstellung also überspitzt, um einen Vergleich mit optischem see-through AR zu ermöglichen. Für die AR-Inhalte gibt es kein einschränkendes Sichtfeld und sie werden in der selben Auflösung wie die virtuelle Umgebung dargestellt. Außerdem enthalten sie nur eine minimale Latenz von 10ms um leichte Verzögerungen beim Tracking zu simulieren. Da

bei Video look-through die Inhalte direkt über das Bild gelegt werden, wird keine Transparenz und Abdunklungseffekt verwendet.

Für generisches optisches see-through AR wird die volle Auflösung für die VR-Umgebung verwendet und diese ohne Latenz angezeigt, da die reale Welt simuliert werden soll. Für die AR-Inhalte wird ein kleines Display mit einem geringen Sichtfeld von ca. 25° und einer Auflösung von 200 x 100 verwendet, um ein deutlich eingeschränktes Display zu simulieren. Das Sichtfeld ist mittig platziert, hat eine Transparenz von 60% und Abdunklung von 20%. Die AR-Inhalte selbst erhalten eine Latenz von 30ms, um ein schlechteres Tracking zu simulieren.

Wie auch für das generische optische see-through AR wird für die Google Glass die VR-Umgebung in voller Auflösung und ohne Latenz dargestellt. Das Display ist nochmals kleiner mit ca. 20° Sichtfeld, verfügt aber über eine deutlich höhere Auflösung von 640 x 360. Zudem befindet es sich oben rechts im Bild, um die Platzierung über dem rechten Auge zu simulieren. Das Sichtfeld wurde etwas größer gewählt, um zwei Dingen entgegen zu wirken. Zum einen fällt bei einem zu kleinen Sichtfeld die geringe Auflösung des VR-Systems noch stärker ins Gewicht, zum anderen machen sich durch die Positionierung oben rechts zusätzlich Linseneffekte bemerkbar. Die Transparenz der AR-Inhalte wurde auf 40% und die Abdunklung auf 55% gesetzt. Diese starke Abdunklung wurde der Einfachheit halber als Ersatz für den Hintergrund der dargestellten Anwendung, einer Navigation, gewählt. Die AR-Inhalte besitzen zu dem keine Latenz und sind im Sichtfeld der Kamera fixiert, da die Google Glass als statisches Overlay agiert. Der dargestellte Kartenausschnitt rotiert zudem mit der Kopfbewegung mit.

Zuletzt wird für die Microsoft HoloLens wie auch für die anderen optischen see-through AR-Lösungen eine voll aufgelöste VR-Umgebung ohne Latenz verwendet. Das Display ist das Größte der Testkonfigurationen und beträgt ca. 30° mit einer Auflösung von 1920 x 1080. Die Transparenz ist auf 30% eingestellt mit einer minimalen Abdunklung von 5%. Da in Demovideos keine keine Latenz der AR-Inhalte wahrzunehmen war, wurde auf diese verzichtet. In Tabelle 2 sind die Konfigurationen nochmals aufgelistet. In Abbildung 33 sind diese gegenübergestellt.

	Video AR	Generisches AR	Google Glass	HoloLens
Auflösung	1280 x 1440	200 x 100	640 x 360	1920 x 1080
Sichtfeld	96°	25°	20°	30°
Latenz VR	30 ms	0 ms	0 ms	0 ms
Latenz AR	10 ms	30 ms	0 ms	0 ms
Transparenz	0%	60%	40%	30%
Abdunklung	0%	20%	55%	5%

Tabelle 2: Gegenüberstellung der AR-Testkonfigurationen.

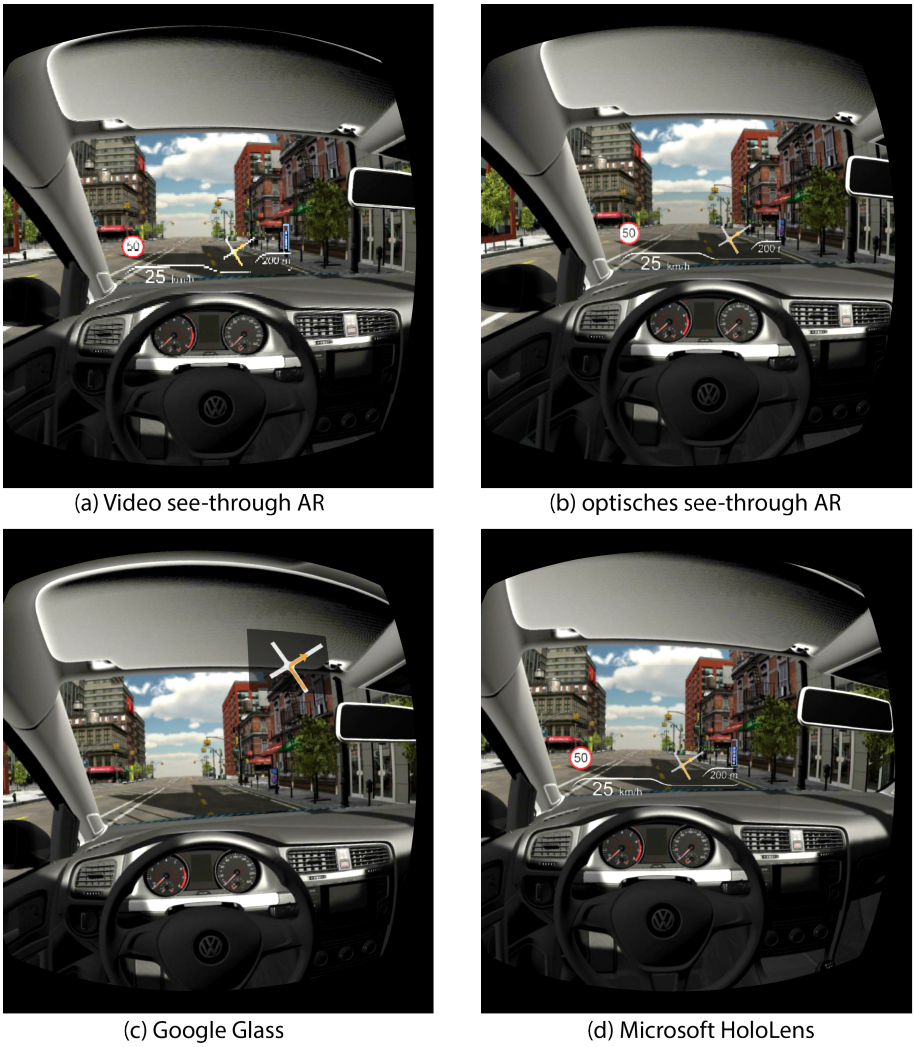

Abbildung 33: Gegenüberstellung der verschiedenen Szenarien der Evaluation

7.2 Durchführung

Für die Durchführung wurden die Probanden einzeln in einem ruhigen Raum befragt. Es handelte sich hierbei nicht um einen komplett eigenständigen Test, sondern sie wurden durch einen Versuchsleiter durch die einzelnen Szenarios geführt. Der dabei enstandende Dialog und die gegebenen Antworten wurden mit einem Diktiergerät aufgezeichnet, um diese später besser auszuwerten und vergleichen zu können. Dies hatte zudem den Vorteil, dass sich Versuchsleiter wie Testperson ohne Ablenkung auf Simulation und Versuchsdurchführung konzentrieren konnten. Der Versuchsleiter hat Szenarien und Begriffe erklärt, die Szenarien eingestellt und zuletzt die Fragen gestellt. Die Probanden mussten nur noch die Gear VR aufsetzen, sich umschauen und

dann ihre Einschätzung abgeben. Im Nachfolgenden wird nun der Ablauf des Tests skizziert.

Zuerst erfolgte die Frage nach dem Einverständnis der Testperson, dass das Gespräch aufgezeichnet wird. Danach wurde der Ablauf erklärt und eine kurze Einweisung in die Gear VR gegeben, falls diese dem Probanden noch nicht bekannt war. Dann wurde das Diktiergerät aktiviert und der eigentliche Test begonnen.

Als erstes Szenario wurde das Video see-through AR-Szenario durchgenommen. Dazu wurde der Proband zuerst gefragt, ob er weiß, wobei es sich bei diesem handelt. Verneinte dieser die Frage, folgte eine entsprechende Erklärung und es wurde zudem ein entsprechendes Video gezeigt. Danach konnte sich der Proband die Gear VR aufsetzen und umschauen. Nach kurzer Zeit und noch während des Umschauens wurde die Frage gestellt, ob sich das Dargestellte wie Video see-through AR anfühlt, ob etwas besonderes auffällt und ob in der Simulation noch etwas fehlt. Hatte der Proband seine Einschätzung abgegeben, konnte er die VR-Brille wieder absetzen und der Versuchsleiter hat das nächste Szenario eingestellt.

Das Vorgehen für die restlichen Szenarien war analog zu dem hier vorgestellten. Als Reihenfolge wurde zunächst das generische optische see-through AR, gefolgt von der Google Glass und der Microsoft HoloLens gewählt.

Waren alle vier Szenarien abgeschlossen, wurde das Diktiergerät abgeschaltet und der Proband konnte den Raum verlassen.

7.3 Auswertung

Im Rahmen der Auswertung wird im Nachfolgenden wird zuerst auf allgemeine Auffälligkeiten eingegangen, die mehrere bis alle Szenarien betreffen. Danach werden Besonderheiten die einzelne Szenarien betreffen durchgegangen.

7.3.1 Allgemeines

Die Vorkenntnisse der Probanden waren insgesamt sehr ähnlich. Lediglich eine Person hatte bereits Kontakt mit einer Gear VR. Bis auf zwei konnten die restlichen Probanden außerdem die Begrifflichkeiten des Video und optischen see-through AR nicht direkt zuordnen. Nach einer kurzen Erklärung stellte sich jedoch bei allen heraus, dass sie bereits über entsprechende Systeme gehört (optisches see-through AR) bzw. bereits damit Kontakt hatten (Video see-through AR). Insbesondere die Google Glass und die Microsoft HoloLens war allen Probanden ein Begriff, allerdings hatte noch niemand eine verwendet.

Bis auf eine Testperson wurde von allen angemerkt, dass die Auflösung insgesamt sehr gering sei, so dass Details nicht mehr erkennbar sind. Dies bezog sich insbesondere auf die Darstellung der virtuellen Welt, aber auch auf

die der AR-Inhalte. Die Ursachen hierfür liegen primär an zwei Eigenschaften der Simulation.

Zum einen wird im Falle der VR-Umgebung die Szene nicht direkt gerendert, sondern ein vorgerendertes 360°-Panorama verwendet. Durch die Entwicklung für Android ist dessen Auflösung stark eingeschränkt. Allgemein wird derzeit eine maximale Auflösung von maximal 2048 x 2048 Pixeln empfohlen, Unity erlaubt bis zu 4096 x 4096 Pixel. Das Panorama wurde daher mit der maximalen Auflösung von 4096 x 2048 pro Auge erstellt. Dieses wird nun auf die als Skybox fungierende Kugel projiziert und verteilt sich auf deren vollen 360°. Dies bedeutet bei einem Sichtfeld der Gear VR von 96°, dass auf deren maximale Auflösung von 1440 Pixel nur rund 1092 Pixel des Panoramas projiziert werden. Eine Lösung hierfür könnte das Aufteilen des Panoramas auf mehrere große Texturen sein, die beim Rendern entsprechend zusammengeschnitten werden.

Abbildung 34: Vergleich eines hochaufgelösten Renderings (links) mit der niedrigeren Auflösung der Gear VR (rechts)

Ein ebenso großes Problem stellt jedoch die Beschränkung durch die native Auflösung der Gear VR dar. Dies fällt besonders stark dadurch auf, dass sich diese auf ein verhältnismäßig großes Sichtfeld recht nahe vor den Augen verteilt. Hierdurch ist es bereits beim Erstellen der Szene, insbesondere der AR-Inhalte, zu Problemen gekommen. Darzustellende Objekte müssen eine recht hohe Mindestgröße besitzen, insbesondere Texte sind schnell zu klein um noch durch die geringe Auflösung dargestellt werden zu können. Das heißt insbesondere Details, die eine reale AR-Brille je nach eigener Auflösung ohne Probleme darstellen könnte, können nicht simuliert werden, da die Pixeldichte der Gear VR viel zu gering dafür ist. Derzeit lösen aber auch Varianten die direkt an einen PC angeschlossen werden, wie die Oculus

Rift und HTC Vive, dieses Problem leider nicht. Deren Auflösung ist sogar noch geringer und würden das Problem verstärken. In Abbildung 34 ist ein Vergleich eines hochauflösend gerenderten Bildes und der tatsächlichen Darstellung abgebildet, um die Problematik zu verdeutlichen.

Ein weiteres Problem welches während des Tests aufgefallen ist, sind fehlende Vergleichsmöglichkeiten. Während dies weniger die Szenarien für Video see-through AR und generisches optisches see-through AR betrifft, fällt es umso deutlicher in den Szenarien für die Google Glass und die Microsoft Hololens auf. Da diese Geräte im Rahmen dieser Arbeit leider nicht zur Verfügung standen, musste auf Präsentationsvideos der Hersteller sowie Videos von Erfahrungsberichten zurückgegriffen werden, um einen möglichst guten Eindruck zu vermitteln. Optimal wäre an dieser Stelle gewesen, hätten die Probanden die entsprechende Hardware selbst ausprobieren und damit einen direkten Vergleich zwischen realer Briller und Simulation ziehen können.

7.3.2 Video see-through AR

Insgesamt empfanden die Testpersonen das Video see-through AR Szenario als authentisch. Von vier Personen wurden noch zusätzliche Anmerkungen gemacht, die allerdings nicht die Simulation selbst, sondern die Darstellung im Szenario betreffen.

Zwei Personen merkten an, dass das eingeblendete HUD für reale Umstände etwas groß und zu ablenkend beim Fahren sei. Die Größe hängt unmittelbar mit dem bereits erwähnten Problem der geringen Auflösung zusammen, so dass es in der verwendeten Form nicht mehr kleiner angezeigt werden konnte. Man könnte es dahin gehend optimieren, dass man weniger und andere Einblendungen vornimmt, damit es weniger prominent wirkt. Insgesamt hatte dieser Punkt allerdings keinen Einfluss auf den Gesamteindruck der Simulation.

Eine weitere Anmerkung von einer Person war, dass es nur eine globales Latenz über VR- und AR-Inhalte geben sollte. Da es sich hierbei allerdings lediglich um eine vorhandene Einstellung im Simulator handelt und zudem an Hand kommerzieller Produkte gezeigt werden konnte, dass auch bei diesen eine gewisse Latenz auf den AR-Inhalten möglich ist, fällt dieser Punkt für den Zweck dieser Evaluation nicht weiter ins Gewicht.

Eine letzte Anmerkung war, dass man die Simulation von Video see-through AR generell anders gestalten könnte. Es wurde vorgeschlagen, ein Display in der virtuellen Welt simulieren, welches die AR-Einblendungen vornimmt. Während dies eine durchaus interessante Idee ist, liegt der Fokus der Arbeit jedoch auf der Simulation von AR-Brillen, nicht Video see-through AR und kann derzeit in dieser Form noch nicht umgesetzt werden. Es wäre allerdings durchaus denkbar, den Simulator in Zukunft um eine entsprechende Möglichkeit zu erweitern.

7.3.3 Generisches optisches see-through AR

Auch die Darstellung einer generischen optischen see-through AR-Brille wurde als insgesamt authentisch bewertet. Da an dieser Stelle zum ersten Mal die Umgebung in voller Auflösung angezeigt wurde, wurden hier auch zum ersten Mal die Anmerkungen zur generellen Auflösung und nicht mehr wahrnehmbaren Details gemacht.

Ein weiteres Problem, welches allerdings nur zwei Personen aufgefallen ist, war, dass die Bildwiederholrate im Falle der VR-Umgebung Einbrüche verzeichnete und nicht ganz flüssig dargestellt wurde. Während die Ursache nicht genau geklärt werden konnte, wurde jedoch festgestellt, dass dies erst auftritt, wenn zu der reinen Darstellung der VR-Umgebung das AR-Postprocessing dazugeschaltet wird. Da derzeit mehrere hintereinander geschaltete Shader angewendet werden werden um das fertige Bild zu erzeugen, könnte versucht werden diese falls möglich weiter zu kombinieren und damit die wiederholte Verarbeitung des Bildes zu verringern. Ob dies das Problem löst ist derzeit allerdings nicht bekannt.

Die Umsetzung der AR-Darstellung, insbesondere die hinzugefügte Latenz sowie geringe Auflösung und Sichtfeld, wurde ansonsten sehr positiv bewertet.

7.3.4 Google Glass

Bei der Umsetzung der Google Glass wurde von allen Teilnehmern am wenigsten angemerkt und die Simulation als authentisch und sehr gut bewertet.

Durch die besondere Positionierung des Sichtfeldes einer Google Glass oben rechts im Bild ist zwei Personen eine Chromatische Aberration aufgefallen. Diese entsteht durch die in der Gear VR verwendeten Linsen und ist damit leider nicht softwaremäßig korrigierbar. Interessant wäre an dieser Stelle ein Vergleich mit anderen VR-Brillen, ob dieser Effekt dort ebenfalls in ähnlicher Stärke auftritt oder sich anders verhält. Insgesamt wir die Simulation nicht hierdurch eingeschränkt, allerdings nimmt die Immersion etwas ab.

7.3.5 Microsoft HoloLens

Auch die Microsoft HoloLens wurde insgesamt positiv und als authentisch bewertet. Im Gegensatz zur Google Glass gab es hier allerdings noch kleinere Anmerkungen.

Zum einen wurde festgestellt, dass das im Test verwendete Sichtfeld etwas zu groß bzw. das der Gear VR im Vergleich zu gering sein könnte. Außerdem könnte auch die Transparenz etwas zu gering eingestellt sein. In dieser Hinsicht fehlt leider die Möglichkeit eines direkten Vergleichs, wodurch sich die Testerstellung sowie der Proband lediglich auf Werbematerial, Erfahrungsberichten und technischen Daten stützen konnte. Da es sich hierbei aber le-

diglich um Einstellungen handelt, stellen diese Anmerkungen kein Problem für die Simulation an sich dar.

Eine ähnliche Anmerkung wurde von einer weiteren Person gemacht. Hierbei ging es darum, dass die AR-Inhalte noch über eine zumindest minimale Latenz verfügen sollten statt fest im Raum zu stehen. Leider war in dieser Beziehung kein Informationsmaterial zu bekommen. In der Aufnahme die für den Erfahrungsbericht gemacht wurde, war ebenfalls keine bemerkbare Latenz vorhanden. Sollte eine geringe Latenz für die korrekte Simulation einer HoloLens nötig sein, kann diese jedoch auch jeder Zeit hinzugefügt werden und stellt damit ebenfalls kein Problem dar.

7.3.6 Abschluss

Insgesamt wurde die Simulation der AR-Brillen von den Probanden als authentisch empfunden und positiv bewertet.

Die Testszenarien könnten an einigen aufgezeigten Stellen noch optimiert werden, waren aber insgesamt gut und konnten den gewünschten Eindruck bei den Probanden erzeugen. Für die Google Glass und Microsoft HoloLens hätte eine direkte Vergleichsmöglichkeit mit den tatsächlichen Brillen noch eine deutliche Verbesserung dargestellt, ließen sich aber vom grundsätzlichen Eindruck her ebenfalls ohne Probleme darstellen.

Die meisten Probleme entstanden durch technische Einschränkungen, die derzeit leider noch nicht gelöst werden können. Bei der Verwendung eines gut ausgestatteten PCs würden zwar Einschränkungen wie zu niedrig aufgelöste Texturen entfallen, das Problem der geringen nativen Auflösung von VR-Brillen bleibt jedoch bestehen. In dieser Beziehung muss auf bessere Endgeräte gewartet werden. Weitere Störfaktoren wie eine Chromatische Aberration an den Rändern des Sichtfeldes der Gear VR lassen sich nur durch entsprechend teurere Optiken verbessern, welche für Endverbraucherprodukte wie eine Gear VR aber vermutlich zu teuer sind.

8 Fazit und Ausblick

In dieser Arbeit wurde gezeigt, dass die glaubhafte Simulation von AR-Brillen in einer VR-Umgebung möglich ist. Dazu wurden zuerst verschiedene Verfahren zur Erstellung von stereoskopischen Bildern untersucht und vorgestellt, um dies auch auf Endgeräten mit deutlichen Einschränkungen in Hard- und Software in einer möglichst hohen Qualität zu ermöglichen. Anschließend wurden zwei Programme entwickelt, eines zur Erstellung von (stereoskopischen) 360°-Panorama-Bildern und der eigentliche Simulator. Als Stereo-Verfahren wurde das Stereo-Panorama eingesetzt.

Die Erstellung von Panoramas ist auf eine maximal mögliche Breite von 4096 Pixel für ein stero und 8192 Pixel für ein einfaches Panorama beschränkt. Da bei größeren Auflösungen nicht mehr genug Speicher für die resultierenden Bilder angefordert werden kann (es erfolgt eine entsprechende Meldung in der Unity-Konsole), ist dies vermutlich eine Einschränkung durch Unity. Außerdem wird derzeit noch kein Tiefenbild des Panoramas erstellt, weshalb sich das Programm noch nicht zum Einsatz für die anderen vorgestellten Verfahren eignet. Diese Funktionalität müsste bei zukünftigen Weiterentwicklungen des Simulators noch ergänzt werden.

Die Entwicklung des eigentlichen Simulators konnte erfolgreich abgeschlossen werden und im Test überzeugen. Damit liegt ein voll funktionstüchtiges Programm zur Simulation von AR-Brillen auf einem VR-Gerät vor. Im Rahmen der Vorbereitung und Durchführung der Evaluation wurden verschiedene Funktionalitäten angesprochen, die in Zukunft weiter ausgebaut und optimiert werden können um den Simulator weiter zu verbessern. Dazu gehören Optimierungen in Bezug auf die Durchführung der Konfiguration von Szenen, beispielsweise dass szenenübergreifende Einstellungen wie das Szenenmanagement global bearbeitet werden können. Außerdem kann eine weitere Optimierung der Performance für schwächere Endgeräte erfolgen, zum Beispiel durch das Zusammenlegen von Shadern, soweit dies möglich ist. Nicht zuletzt kann der Simulator um weitere stereoskopische Bilderstellungsverfahren erweitert werden. Wie bereits erwähnt kommt aufgrund der verfügbaren Systemleistung dafür eher das Iterative Image Warping als das Post-Rendering 3D Warping in Frage.

Wie insbesondere beim Expertentest festgestellt wird die Simulation von AR-Brillen auf VR-Geräten vor allen Dingen durch hardwareseitige Einschränkungen limitiert. Hierunter fallen viele Faktoren, die prominentesten davon sind Bildschirmauflösung, Helligkeit und Sichtfeld. Mit der Weiterentwicklung der VR-Geräte könnten in Zukunft diese Limitierungen deutlich verringert werden bis komplett entfallen. Auf diese Weise könnte in Zukunft eine deutlich bessere Simulation verschiedener Umgebungen, Gegebenheiten und eine höhere Immersion erreicht werden.

Literatur

[1] P. Milgram and H. Colquhoun, *Mixed Reality - Merging Real and Virtual Worlds*. ISBN 978-3-642-87514-4: Springer, 1999.

[2] Epson, "Moverio bt-200 smart glasses." `http://www.epson.com/alf_upload/images/products/bt200_fca-s_396x264.jpg`. (Stand: 12.10.2016).

[3] T. Reckmann, "Google glass explorer edition (2014)." `https://de.wikipedia.org/wiki/Google_Glass#/media/File:Google_Glass_Main.jpg`. (Stand: 12.10.2016).

[4] Microsoft, "Microsoft hololens." `https://compass-ssl.surface.com/assets/13/b5/13b57736-9875-4332-af9f-dca0accbc331.jpg?n=HoloLens_Homepage_Audiences1_1920_1001.jpg`. (Stand: 16.10.2016).

[5] Samsung, "Gearvr." `http://www.samsung.com/in/consumer-images/product/wearables/2016/SM-R322NZWAINU/features/SM-R322NZWAINU-372765-0.jpg`. (Stand: 16.10.2016).

[6] F. U. Contrast, "Vw wolfsburg." `https://upload.wikimedia.org/wikipedia/commons/0/01/VW_Wolfsburg.JPG`. (Stand: 30.10.2016).

[7] F. motortrend.com, "Vw golf gte cockpit." `http://st.motortrend.com/uploads/sites/5/2014/02/Volkswagen-Golf-GTE-cockpit.jpg`. (Stand: 30.10.2016).

[8] F. robots.com, "Industrial robots." `https://www.robots.com/images/Robot%20Integration.jpg`. (Stand: 30.10.2016).

[9] P. Bourke, "Synthetic stereoscopic panoramic images," in *Interactive Technologies and Sociotechnical Systems*, pp. 147–155, Springer, 2006.

[10] H. Bowles, K. Mitchell, R. W. Sumner, J. Moore, and M. Gross, "Iterative imagewarping," in *Computer Graphics Forum*, vol. 31, p. 237–246, John Wiley & Sons Ltd, May 2012.

[11] M. Solh and G. AlRegib, "Depth adaptive hierarchical hole-filling for dibr-based 3d videos," in *SPIE Proceedings, Three-Dimensional Image Processing (3DIP) and Applications II*, vol. 8290, SPIE, February 2012.

[12] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3d warping," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 7–16, ACM, April 1997.

[13] R. Azuma, "A survey of augmented reality," *Presence*, vol. 6, pp. 335–385, August 1997.

[14] R. Behringer, "Augmented reality." https://www.academia.edu/2977942/Augmented_reality. (Stand: 17.10.2016).

[15] Duden, "virtuell." http://www.duden.de/rechtschreibung/virtuell#Bedeutungb. (Stand: 20.10.2016).

[16] W. R. Sherman and A. B. Craig, *Understanding virtual reality*. ISBN 978-1-55860-353-0: Morgan Kaufmann, 2003.

[17] Epson, "Moverio bt-200 smart glasses specs." http://www.epson.com/cgi-bin/Store/jsp/Product.do?sku=V11H560020&UseCookie=yes#key-features. (Stand: 12.10.2016).

[18] Epson, "Moverio bt-200 smart glasses brochure." http://www.epson.com/alf_upload/pdfs/brochure_moverio.pdf. (Stand: 12.10.2016).

[19] Google, "Google glass tech specs." https://support.google.com/glass/answer/3064128?hl=en&ref_topic=3063354. (Stand: 18.04.2013).

[20] J. Han, J. Liu, X. Yao, and Y. Wang, "Portable waveguide display system with a large field of view by integrating freeform elements and volume holograms," *Opt Express*, vol. 23, February 2015.

[21] Google, "Google glass help - myglass app on android." https://support.google.com/glass/answer/3068035?hl=en. (Stand: 16.10.2016).

[22] doc ok.org, "On the road for vr: Microsoft hololens at build 2015, san francisco." http://doc-ok.org/?p=1223. (Stand: 16.10.2016).

[23] Microsoft, "Hardware details." https://developer.microsoft.com/en-us/windows/holographic/hardware_details. (Stand: 16.10.2016).

[24] Samsung, "Gearvr." http://www.samsung.com/global/galaxy/gear-vr/. (Stand: 17.10.2016).

[25] S. R. Ellis, F. Bréant, B. Menges, R. Jacoby, and B. D. Adelstein, "Simulation of augmented reality systems in purely virtual environments," in *Virtual Reality Conference*, pp. 287–288, IEEE, March 2009.

[26] J. L. Gabbard, J. E. S. II, and D. Hix, "The effects of text drawing styles, background textures, and natural lighting on text legibility in outdoor augmented reality," *Presence*, vol. 15, February 2006.

[27] S. Kim and A. K. Dey, "Simulated augmented reality windshield display as a cognitive mapping aid for elder driver navigation," in *Proceedings*

of the SIGCHI Conference on Human Factors in Computing Systems, pp. 133–142, ACM, April 2009.

[28] E. Ragan, C. Wilkes, D. A. Bowman, and T. Höllerer, "Simulation of augmented reality systems in purely virtual environments," in *Virtual Reality Conference*, pp. 287–288, IEEE, March 2009.

[29] C. Lee, S. Bonebrake, T. Höllerer, and D. A. Bowman, "The role of latency in the validity of ar simulation," in *Proceedings of the 2010 IEEE Virtual Reality Conference*, pp. 11–18, IEEE, 2010.

[30] C. Lee, G. A. Rincon, G. Meyer, T. Höllerer, and D. A. Bowman, "The effects of visual realism on search tasks in mixed reality simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, April 2013.

[31] H. Lee, Y. Tateyama, and T. Ogi, "Panoramic stereo representation for immersive projection display system," in *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*, pp. 379–382, ACM, 2010.

[32] eVRydayVR, "360 panorama capture." `https://www.assetstore.unity3d.com/en/#!/content/38755`. (Stand: 05.11.2016).