VISUS

Masterarbeit

Immersive Kartendarstellungen für AR-Brillen

Armin K.

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Prof. rer. nat. Dr. techn. h. c. Dr.-Ing. E. h. Thomas Ertl
Betreuer/in:	Dipl.-Inf. Christoph Müller, Michael Becher, M.Sc.
Beginn am:	3. September 2019
Beendet am:	29. Juni 2020

Kurzfassung

Digitale Karten, wie sie in Autos und Mobiltelefonen genutzt werden, sind aus dem heutigen Alltag nicht mehr weg zu denken. Allerdings sind die Visualisierungsmöglichkeiten dieser Systeme durch ihre Limitierung auf zwei Dimensionen stark eingeschränkt. Mithilfe von Augmented-Reality-Geräten, wie beispielsweise der Microsoft Hololens, ist man in der Lage, die reale Welt mit dreidimensionalen virtuellen Objekten zu ergänzen. Diese Arbeit präsentiert mehrere Ansätze für Minimaps, welche es dem Benutzer einer solchen Augmented-Reality-Brille ermöglichen sollen, sich durch eine virtuelle Karte zu bewegen. Weiterhin werden verschiedene Verzerrungsalgorithmen entwickelt, welche dem Anwender die Lesbarkeit der Minimap erleichtern, indem sie die Sichtbarkeit von Elementen in unmittelbarer Nähe des Nutzers verbessern. Anschließend wird der Entwurf einer Studie präsentiert, deren Ziel darin besteht, zu testen, inwiefern diese Minimaps den Anwender bei der Navigation innerhalb der realen Welt unterstützen können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ARTK Augmented Reality Toolkit. 25

FOV Field of View. 15

FPS Frames per Second. 22

HLSL High-Level Shading Language. 32

HMD Head Mounted Display. 17

MRTK Mixed Reality Toolkit. 7

SDK Software Development Kit. 25

VISUS Visualisierungsinstitut der Universität Stuttgart. 8

1 Einleitung

Karten (engl.: Maps), sind aus unserem Alltag inzwischen nicht mehr wegzudenken. Sie finden sich nicht nur in verschiedenen Navigationssystemen, wie beispielsweise in Automobilen oder Applikationen wie Google Maps, sondern auch in vielen verschiedenen Computerspielen. Der Nutzen dieser (Mini-) Maps ist allerdings immer der gleiche: Sie sollen bei der Orientierung und Navigation innerhalb der Welt unterstützen, sei sie nun real oder fiktiv. All diese Kartenvarianten sind zweidimensional. Zwar gibt es über Programme wie Google Street View bereits die Möglichkeit, eine räumliche Ansicht der Umgebung zu erhalten, jedoch fehlte dieser Anwendung bislang ein entsprechendes Medium, um alle Vorzüge einer dreidimensionalen Ansicht zu vermitteln. Des Weiteren wird bei dieser dreidimensionalen Ansicht die eigene Position selten berücksichtigt, was jedoch bei der Unterstützung der Orientierung unabdingbar ist. Das Ziel von Augmented Reality besteht darin, die reale Welt mit digitalen Zusatzinformationen zu verknüpfen. Dies geschieht, indem mithilfe eines entsprechenden Mediums, wie beispielsweise einer Augmented-Reality-Brille oder einem Beamer virtuelle Gegenstände in die reale Welt projiziert werden. Somit kann einem Nutzer beispielsweise durch eine solche Brille nicht nur die reale Welt gezeigt werden, sondern auch ein virtueller Richtungspfeil, welcher ihm die Richtung seines Ziels vorgibt. Im Zuge dieser Masterarbeit soll mit Hilfe von Augmented Reality eine dreidimensionale Darstellung eines Gebäudes erstellt und ermittelt werden, inwiefern die virtuellen Konstrukte bei der Navigation innerhalb dieses Gebäudes in der realen Welt unterstützen können. Hilft es dem Besucher eines fremden Gebäudes, sich darin zurecht zu finden, wenn ihm während der Suche eines bestimmten Raumes ein dreidimensionales Modell dieses Gebäudes angezeigt wird? Welche Darstellungsvariante für ein solches virtuelles Modell ist dabei die effizienteste? Dies sind zwei der Forschungsfragen, welche es mit dem hier entwickelten System zu klären gilt. Als Gerät zur Darstellung der virtuellen Objekte wird die von Microsoft entwickelte Augmented-Reality-Brille Hololens 1 verwendet. Das System wird in der Spiele-Engine Unity entwickelt, da diese über einige Schnittstellen zu der Microsoft Hololens verfügt. Bei der Entwicklung der Nachbildung des betroffenen Gebäudes muss nicht nur darauf geachtet werden, es so akkurat wie möglich abzubilden, sondern auch, dass beim Traversieren der Umgebung keine wichtigen Details vom Anwender verdeckt werden. Darüber hinaus kann es vor allem bei komplexeren Karten schnell passieren, dass für den Benutzer wichtige Gebiete zu unübersichtlich werden und es ihm dadurch schwer fällt, für ihn relevante Informationen aus der Karte zu erhalten. Um dem Benutzer das Extrahieren die für ihn interessanten Informationen aus der Karte zu erleichtern, werden verschiedene Verzerrungsmethoden mithilfe von Shadern entwickelt. Diese werden anschließend auf unterschiedliche Aspekte hin miteinander verglichen. Dazu zählen beispielsweise Nutzerfreundlichkeit oder auch wie aufwendig die Berechnung der Verzerrung für die Hardware ist. Da Augmented-Reality-Anwendungen eine Bildrate von mindestens 60 Bildern pro Sekunde benötigen, um ein realitätsnahes Anwendungsgefühl zu gewährleisten, muss hierauf ein besonderes Augenmerk gelegt werden. Aber auch der eingeschränkte Field of View (FOV) der Hololens muss berücksichtigt werden. Einen Raum in seiner normalen Größe abzubilden ist daher beispielsweise nicht möglich. Das liegt daran, dass ein durchschnittlich großer Raum schon zu groß wäre, um ihn auf einmal mit der Hololens betrachten zu können. Darüber

hinaus besteht das Ziel der virtuellen Karte darin, dem Nutzer einen Überblick über die Umgebung zu liefern. Würde man die Karte in Originalgröße visualisieren, würde die Übersicht verloren gehen. Somit muss eine geeignete Minimierung der Gesamtumgebung gefunden werden, damit man nicht nur Details, sondern auch das große Ganze im Blick behalten kann. Es soll zwischen zwei Karten Varianten unterschieden werden. Zum einen gibt es eine 3D Minimap, in welcher der Nutzer über einen Positionsindikator innerhalb der Karte angezeigt wird. In der anderen Variante wird eine große, fixe dreidimensionale Karte um den Nutzer herum aufgebaut, in welcher der Nutzer selbst der Positionsindikator ist. Somit muss gewährleistet werden, dass die virtuellen Räume mit deren Position in der realen Welt überein stimmen. In Kapitel 2 werden umfassend bisherige Kenntnisse über die für diese Masterarbeit relevanten Themen untersucht. Darunter fällt ein Verzerrungsalgorithmus namens "Fisheye", der für diese Arbeit von der 2D Variante in 3D überführt wird, Ansätze zum Thema Spatial Mapping von Augmented Reality in der realen Welt und die Untersuchung bereits implementierter AR-Navigationstools.

In Kapitel 3 werden die verschiedenen Grundlagen, welche für das Verständnis dieser Arbeit notwendig sind, geklärt. Dazu gehört eine kurze Einführung in die Soft- und Hardware, welche für diese Arbeit verwendet wurden und grundlegende Informationen über Shader.

Kapitel 4 bildet den Hauptteil dieser Arbeit. Hier werden die verschiedenen Verzerrungsvarianten und Minimaps erläutert und aufgezeigt, was die unterschiedlichen Probleme bei deren Implementierung waren. Darüber hinaus wird hier eine Studie skizziert, mit deren Hilfe getestet werden kann, welche der implementierten Minimaps am Besten bei der Orientierung unterstützen kann.

Abschließend wird in Kapitel 5 ein kurzer Rückblick über meine Arbeit geliefert. Außerdem werden einige Richtungen aufgezeigt, in die das entwickelte System weiter entwickelt werden kann.

2 Verwandte Arbeiten

Um eine Grundlage zu haben, auf der ich bei der Entwicklung meines Systems aufsetzen kann, war es zu Beginn wichtig in Erfahrung zu bringen, welche mit meinem Thema verwandte Forschungen bereits durchgeführt wurden. Dabei stehen für diese Ausarbeitung insbesondere diejenigen Arbeiten im Fokus, welche sich mit unterschiedlichen Verzerrungsmethoden beschäftigen (Abschnitt 2.3). Darüber hinaus werden Arbeiten betrachtet, welche sich bereits mit dem Thema User Interfaces innerhalb der realen Welt beschäftigt haben. Insbesondere sind diejenigen Arbeiten hierfür von Belang, welche mithilfe von Augmented Reality die virtuelle Welt mit der realen Welt in irgendeiner Weise miteinander verknüpft haben. Da virtuelle Karten, ob nun zwei- oder dreidimensional, in erster Linie für Aufgaben geeignet sind, welche mit Navigation in Verbindung stehen, wird außerdem auf Arbeiten eingegangen, welche dem Anwender mithilfe von virtuellen Systemen bei der Navigation in der realen Welt unterstützen sollten.

2.1 Augmented Reality

Bei einer Augmented-Reality-Anwendung handelt es sich um eine Echtzeit Ansicht der physischen Welt, ergänzt durch computergenerierte virtuelle Objekte [CFA+11]. 1968 entwickelte Ivan Sutherland das erste Head Mounted Display (HMD) und bildete somit eine Grundlage für die Augmented-Reality-Geräte der heutigen Zeit. Azuma et al. differenzieren hier zwischen drei verschiedenen Geräten [ABB+01]:

1. Augmented-Reality-Brillen: Diese werden von einem Nutzer auf dem Kopf getragen und erzeugen das gewünschte Bild direkt vor den Augen. Diese Brillen können entweder komplett durchsichtig sein, wobei der Nutzer die reale Welt direkt durch die Linsen sieht, oder undurchsichtig, wobei die Sicht des Anwenders von einer Video Kamera aufgezeichnet und auf den Display in der Brille projiziert wird. In beiden Fällen werden die virtuelle Objekte auf dem Display innerhalb der Brille angezeigt, wodurch der Eindruck entsteht, dass diese mit der realen Welt koexistieren. Bei der Microsoft Hololens, welche in dieser Arbeit verwendet wird, handelt es sich um den ersten der beiden Typen.
2. Tragbare Displays: Bei dieser Variante trägt der Anwender ein kleines Augmented-Reality-Display in der Hand. Dieses fungiert wie eine Art Fenster, welches mithilfe von Kameras die Welt dahinter aufzeichnen und diese, ergänzt mit virtuellen Objekten, auf dem Display wiedergeben.
3. Projektionen: Hier ist kein spezifischer Display vonnöten. Die virtuellen Objekte werden mithilfe von Projektoren in die reale Welt projiziert und erwecken damit den Anschein, als wären sie ein Teil von dieser.

Bei der Nutzung solcher Augmented-Reality-Geräte muss man besonders auf deren Einschränkungen achten, wenn man ein nutzerfreundliches System entwickeln möchte. Augmented-Reality-Brillen verfügen zum Beispiel mit ihren durchsichtigen Displays nicht über genügend Helligkeit, um sie problemlos im Außenbereich, wenn die Sonne scheint, zu benutzen. Darüber hinaus stellt die Auflösung und der eingeschränkte FOVs weitere Herausforderungen dar [ABB+01].

Diese Einschränkungen werden jedoch mit dem stetigen technologischen Fortschritt immer geringer. Darüber hinaus sinken die Kosten für Augmented-Reality-Geräte stetig, was sie für eine immer größer werdende Spannweite von Menschen zugänglich macht. Daher macht es Sinn zu untersuchen, wofür Augmented Reality in der realen Welt sinnvoll eingesetzt werden kann. Höllerer et al. haben ein experimentelles mobiles Augmented Reality System (MARS) entwickelt, welches verschiedene User Interfaces zur Verfügung stellt mit Informationen, welche in der realen Welt veranktert sind [HFT+99]. Die folgenden vier User Interfaces standen dem MARS System zur Verfügung:

1. Für den Außenbereich, der Nutzer bewegt sich in einem vorgeschriebenem Areal und sieht mithilfe einer Augmented-Reality-Brille und einem GPS Tracking System auf dem Rücken verschiedene virtuelle Elemente innerhalb der realen Welt. Hier waren es Namen verschiedener Gebäude.
2. Ebenfalls für den Außenbereich geeignet, eine zweidimensionale Karte auf einem tragbaren Display, um Informationen auf Abruf über die verschiedenen Gebäude zu erhalten.
3. Für den Innenbereich, eine dreidimensionale Karte des Gebietes, auf dem der Anwender bestimmte Gebiete markieren und annotieren kann und somit mit einem Nutzer, der gerade draußen unterwegs ist, interagieren kann.
4. Eine dreidimensionale virtuelle Karte des Gebiets, welche auf einen Tisch projiziert wird. Mithilfe von Handtrackern kann man mit den einzelnen Elementen interagieren (siehe Abbildung 2.1 [HFT+99]).

Die Idee von MARS besteht darin, dass Nutzer, welche das System kollaborativ benutzen, egal welche UI sie gerade verwenden, miteinander interagieren können. Beispielsweise kann ein Anwender, welcher die vierte UI benutzt, einen Marker in dem 3D Modell platzieren (so wie es in Abbildung 2.1 zu sehen ist) und ein Nutzer, der gerade außerhalb unter Verwendung der ersten UI unterwegs ist, kann diese Markierung in Echtzeit sehen.

Besonders die vierte UI, welche in Abbildung 2.1 zu sehen ist, ist für diese Arbeit von großem Interesse. Auch hier soll eine dreidimensionale virtuelle Karte eines real existierenden Gebäudes konstruiert werden. Der Unterschied liegt jedoch darin, dass in meiner Arbeit zusätzlich verschiedene Verzerrungsmethoden implementiert werden sollen, welche bei der Betrachtung eines Bereichs von Interesse unterstützen sollen. Des Weiteren werden, neben einer wie in Abbildung 2.1 zu sehenden, stationären Karte, auch noch andere Kartenvarianten implementiert und miteinander verglichen.

2.2 Navigationshilfen

Navigation ist eine Aufgabe, die jeder aus dem Alltag kennt. Es passiert regelmäßig, dass man an einen Ort möchte, an dem man noch nie war, und man auf irgendeine Art von Navigationshilfe angewiesen ist, um diesen zu finden. Früher hat man sich mit physischen Karten Abhilfe geschafft,

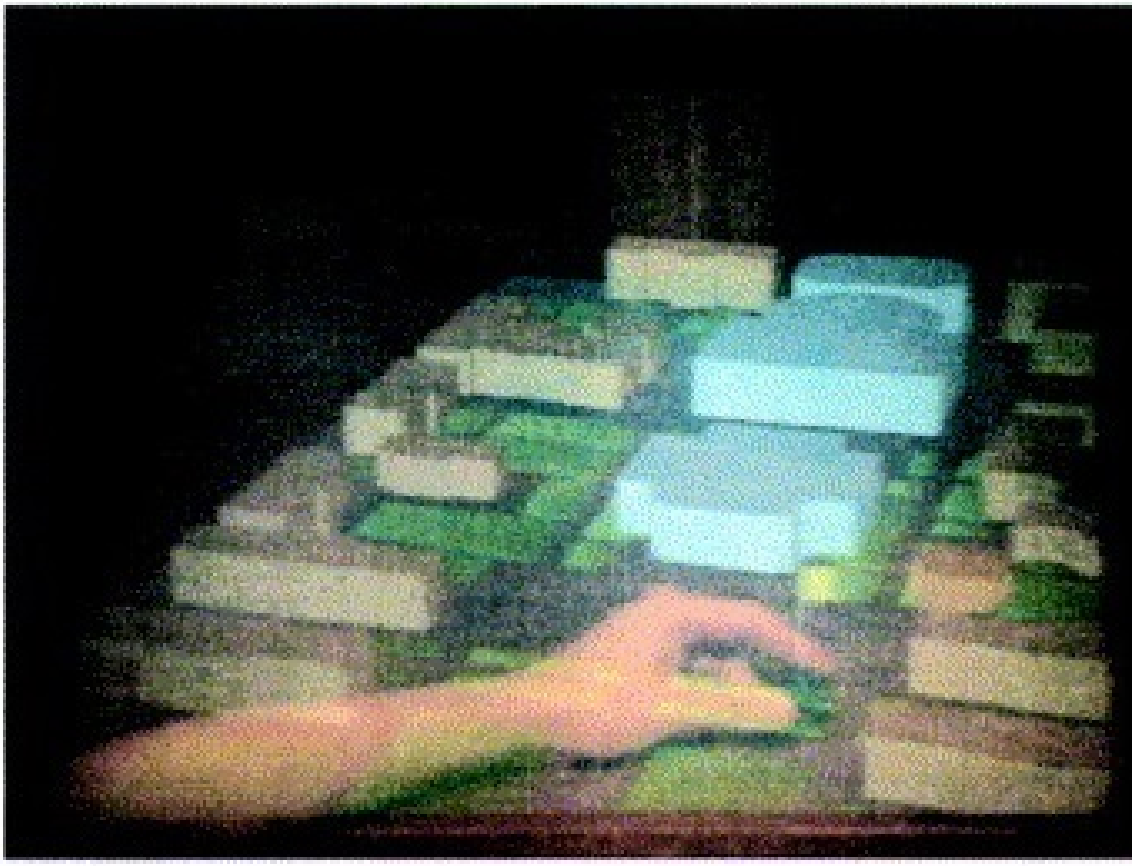


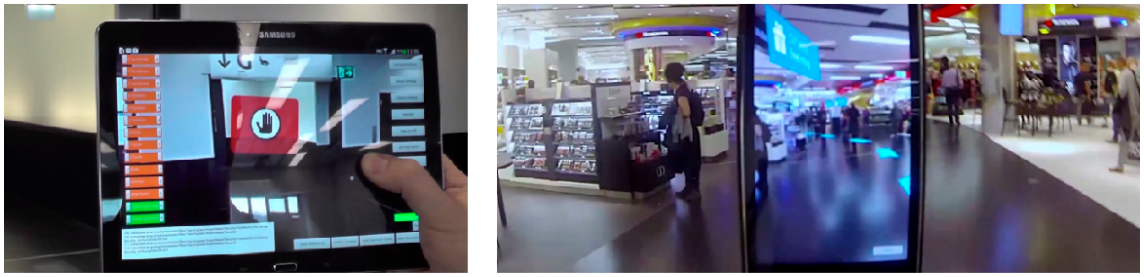
Abbildung 2.1: Graphische Darstellung der vierten UI des MARS [HFT+99].

im heutigen Zeitalter sind jedoch praktischere Navigationshilfen allgegenwärtig. Applikationen wie beispielsweise Google Maps können die eigene Position und die des gewünschten Ziels mittlerweile bis auf den Meter genau lokalisieren. Diese Systeme sind jedoch auf bestimmte Lokalisierungshilfsmittel angewiesen. Ein Navigationsgerät für das Auto beispielsweise funktioniert nicht ohne GPS. Daher haben Narzt et al. ein System entwickelt, welches Informationen zu Positionierung und Orientierung aus einer beliebigen Sensorquelle bezieht und mithilfe von Video und 3D Grafiken eine Augmented Reality Ansicht liefert [NPF+03]. Dieses System nennt sich INSTAR (Information and Navigation System Through Augmented Reality). Es bezieht Informationen zur Positionierung aus unterschiedlichen Systemen, wie beispielsweise GPS oder WLAN und Orientierung aus im System implementierten Gyrometern. Routing-Informationen erhält es wiederum aus herkömmlichen Navigationsgeräten. Mithilfe von Augmented Reality wird dem Nutzer auf einem Display die Strecke visuell dargestellt. In Abbildung 2.2 wird illustriert, wie man sich ein Augmented-Reality-Navigationssystem vorstellen kann. Man hat sich hier für einen semi-transparenten gelben Pfad entschieden, welcher dem Anwender den Weg weist. Gerade in komplizierten Verkehrslagen, wenn es mehrere mögliche Abfahrten gibt, kann eine solche visuelle Repräsentation des Weges definitiv von Nutzen sein.

Abbildung 2.2: INSTAR Simulation. Man sieht deutlich, welche Strecke der Betrachter nehmen muss und sieht diese sogar in der realen Welt visuell dargestellt [NPF+03].

INSTAR stellt ein System dar, welches es dem Anwender ermöglicht, ob per Auto oder zu Fuß, den gewünschten Weg mithilfe von Augmented Reality gezeigt zu bekommen. Beide Möglichkeiten sind jedoch auf die Außenwelt beschränkt. Es gibt aber auch Fälle, in denen es wünschenswert wäre, Navigationsunterstützung innerhalb eines Gebäudes zu bekommen. Dies ist vor allem dann besonders hilfreich, wenn man in einer komplizierten Gebäudestruktur mit vielen Gängen und Räumen unterwegs ist. Da sich mein System ebenfalls als eine potentielle Navigationshilfe innerhalb von Gebäuden anbietet, sind etwaige Forschungen in diese Richtung von besonderem Interesse.

Gerstweiler et al. haben sich mit diesem Thema auseinander gesetzt und ein System namens HyMoTrack (kurz für Hybrid Motion Tracking) entwickelt, welches bei der Orientierung in Gebäuden unterstützen soll. Herkömmliche Trackingmethoden, wie beispielsweise GPS, sind in Gebäuden leider nicht präzise genug einsetzbar. Aus diesem Grund wird in dem hier vorgestellten System mithilfe eines hochentwickelten Computer-Vision-Kamera-Systems die initiale Position bestimmt, indem das Gebiet nach klar sichtbaren, eindeutigen Eigenschaften, wie Werbungen oder Firmenlogos abgesucht wird. In einem zweiten Schritt wird durch den SLAM-(simultaneous localization and mapping) Algorithmus die Position des Anwenders getrackt [GVK16]. Der Algorithmus hat die Eigenschaft, dass er mithilfe von Scannern die Umgebung untersucht und daraus eine Referenzkarte erstellt, mit deren Unterstützung navigiert werden kann [BD06]. In dem von Gerstweiler et al.



(a) (b)

Abbildung 2.3: Die Smartphone Ansicht des HyMoTrack Systems. Links sieht man ein Warnsignal, wenn der Anwender kurz davor ist, einen falschen Pfad zu wählen. Auf der rechten Seite sieht man den richtigen Pfad in dem Smartphone mithilfe von weißen Streifen dargestellt [GVK16].

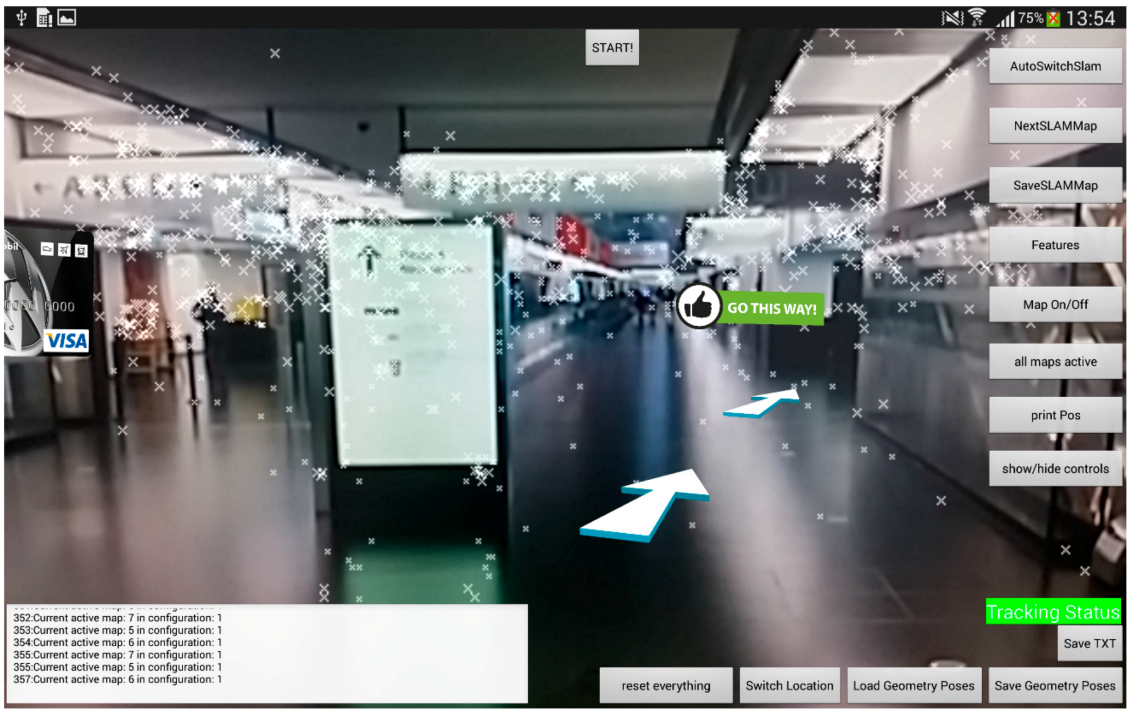


Abbildung 2.4: Die Ansicht des User Interfaces des HyMoTrack. In diesem Beispiel signalisieren Pfeile den richtigen Pfad und eine Textzeile beschreibt darüber hinaus, dass man sich auf dem richtigen Pfad befindet. Über die Menüansicht an den Seiten kann man zwischen unterschiedlichen Visualisierungs- und Tracking-Modi wechseln [GVK16].

vorgestellten System werden die visuellen Features, die über den SLAM-Algorithmus identifiziert werden, mit einer 2D-Karte des entsprechenden Gebäudes überlagert, sodass die Features einen räumlichen Zusammenhang erhalten. Diese Operationen sind sehr rechenintensiv, weshalb sie auf einem separaten Hochleistungsrechner getätigt werden und nur das Endresultat, also die fertige Karte wird dann auf das zur Navigation verwendete Smartphone geschickt. Um das System zu testen wurde eine Studie in einem Flughafen durchgeführt, in der ein Kunde verschiedene Orte, unter Zuhilfenahme des entwickelten Systems, finden musste. Ein wichtiger Faktor bei diesem Testverfahren war die Echtzeit-Reaktion des Systems. Wenn die Pfadsuche zu lange dauert oder das System auf 15 Frames per Second (FPS) herunter fährt, so sinkt auch die Akzeptanz des Nutzers gegenüber dem System. In Abbildung 2.3 und Abbildung 2.4 wird dargestellt, wie HyMoTrack den Anwender bei dem Finden des richtigen Weges unterstützt.

2.3 Fisheye-Verzerrung

Verzerrungen sind besonders dann nützlich, wenn man es dem Betrachter einer komplexen Struktur ermöglichen möchte, Details in Erfahrung zu bringen, ohne den Überblick über die komplette Struktur zu verlieren. Einer der häufigeren Anwendungsfälle hierfür ist das Betrachten eines Graphen. Graphen mit Hunderten von Knoten oder anderen komplizierten Strukturen sind mittlerweile keine Seltenheit mehr, insbesondere in Computerwissenschaften. Bei solchen großen Schaubildern muss man sich meistens entscheiden, ob man lieber globale Informationen, oder lokale Details betrachten möchte. Sarkar und Brown [SB94] haben eine Möglichkeit untersucht, wie man den für den Betrachter interessanten Part eines Graphen in lokalem Detail hervorheben kann, ohne die globale Übersicht über den restlichen Graphen zu verlieren. Die von ihnen entwickelte Fisheye-Linse funktioniert so, dass der relevante Bereich vergrößert dargestellt wird, während die Gebiete um ihn herum sukzessiv von dem Bereich weg geschoben und mit wachsendem Abstand verkleinert werden. In Abbildung 2.5 ist der Fisheye-Algorithmus in Aktion zu beobachten. Beide Bilder zeigen einen Graphen mit jeweils 134 Knoten und 338 Kanten. Die Knoten repräsentieren Städte der Vereinigten Staaten und die Kanten einen Pfad zwischen benachbarten Städten. Es fällt auf, dass es selbst bei einer solch überschaubaren Anzahl an Knoten und Kanten aufgrund des begrenzten Platzes und Kanten- und Knotenüberschneidungen schwer fällt, lokale Informationen zu beziehen. Im rechten Bild sieht man nun unter Anwendung des Fisheye Algorithmus, wie lokale Details in den Vordergrund gerückt werden. Die Stadt von Interesse ist hier St. Louis, weshalb dieser Knoten besonders groß dargestellt wird (er steht quasi im Zentrum der Fisheye Linse). Alle anderen Städte werden von St. Louis weggeschoben, jedoch werden auch die Städte, welche eine direkte Verbindung zu St. Louis besitzen, größer dargestellt. Je weiter eine Stadt von dem fokussierten Gebiet entfernt ist, desto kleiner wird sie, sodass teilweise der Name nicht einmal mehr lesbar ist. Diese Darstellung ist beispielsweise dann besonders hilfreich, wenn die Städte in der Umgebung eines bestimmten Gebietes von Interesse sind.

Reinhard et al. entwickelten ein Tool, welches auf den Grundlagen des Fisheye-Algorithmus beruht. Ihnen zufolge liegt der Hauptfokus des Fisheye-Effekts darauf, den Bereich von Interesse hervorzuheben. Allerdings ist es ebenso wichtig, dass die Struktur des Graphen dabei erhalten bleibt [RMS+08]. Dies liegt daran, dass der Betrachter beim ersten Anblick der Karte bereits einen ersten Eindruck gewinnt, welcher nicht durch Deformation zerstört werden darf.

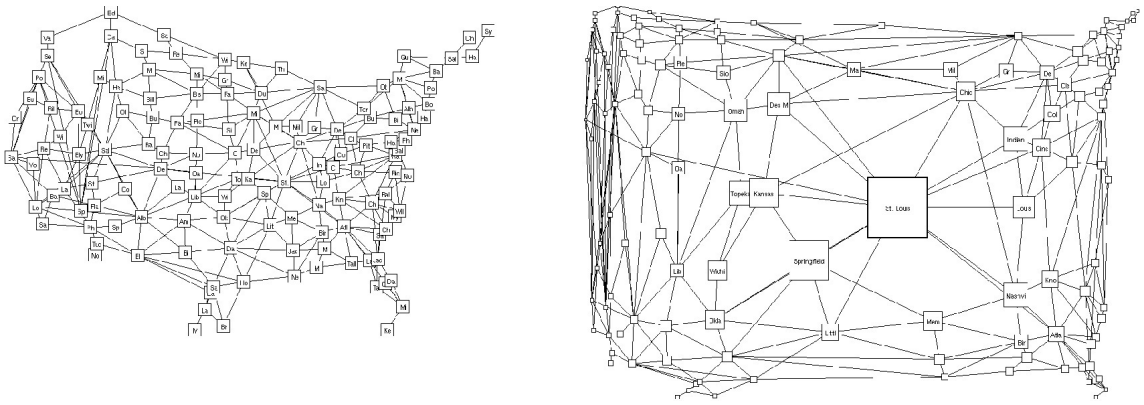


Abbildung 2.5: Eine Darstellung der Funktionsweise der Fishey-Linse. Auf der linken Seite wird eine unübersichtliche Karte der amerikanischen Staaten illustriert. Auf der rechten Seite sieht man dieselbe Karte, allerdings unter Anwendung des Fisheye Algorithmus [SB94].

Zunächst interessiert uns die intuitive Berechnung der Position und die Größe eines Knoten nach der Anwendung des Fisheye-Algorithmus. Diese lässt sich wie folgt bestimmen:

$$P_{feye}(\upsilon, f) = F_1(P_{norm}(\upsilon), P_{norm}(f)) \tag{2.1}$$

$$S_{feye}(\upsilon, f) = F_2(S_{norm}(\upsilon), P_{norm}(v), P_{norm}(f), API(\upsilon)) \tag{2.2}$$

P_{feye} stellt die berechnete Position und S_{feye} die Größe eines Knoten nach Anwendung des Fisheye-Algorithmus dar. Hierbei stehen υ für den Knoten und f für den Fokus, also den für den Betrachter relevanten Bereich. In Gleichung 2.2 ist noch ein weiterer Wert, die API (a priori importance) aufgeführt. Diese ist ein einfacher Zahlenwert, welche die relative Wichtigkeit des Knoten im globalen Kontext widerspiegelt. Sarker und Brown [SB94] haben in ihrer Arbeit außerdem eine Formel für die Menge an Details, welche ein Knoten im Fisheye-View beinhaltet und ein für den visuellen Wert betrachtet, jedoch sind diese beiden Formeln für den weiteren Verlauf dieser Arbeit zu vernachlässigen.

$$P_{feye} = \langle G(\frac{D_{normx}}{D_{maxx}} D_{maxx} + P_{focus_x}, G(\frac{D_{normy}}{D_{maxy}} D_{maxy} + P_{focus_y} \rangle \tag{2.3}$$

wobei

$$G(x) = \frac{(d+1)x}{dx+1} \tag{2.4}$$

In Gleichung 2.3 zeigt die ausformulierte Formel F_1, welche in Gleichung 2.1 zu sehen ist. Hierbei bezeichnen D_{normx} den horizontalen Abstand zwischen dem Fokuspunkt und dem Punkt, der transformiert wird und D_{maxx} den horizontalen Abstand zwischen der Grenze des Graphen und dem Fokuspunkt. Die Variable d in Gleichung 2.4 steht für den Verzerrungsfaktor. Je größer dieser gewählt ist, desto höher fällt der Zoom-Effekt in dem betrachteten Bereich aus.

Diese Formeln werden für die Implementierung des Fisheye-Effekts in dieser Masterarbeit eine große Rolle spielen. Jedoch muss berücksichtigt werden, dass diese Arbeit nur für zweidimensionale Graphen ausgelegt sind. Es bleibt zu untersuchen, welche Auswirkungen eine Überführung der Algorithmen in den dreidimensionalen Raum nach sich zieht.

3 Grundlagen

Da es sich bei dem entwickelten System um eine Augmented-Reality-Anwendung handelt, muss im Vorfeld die Art und Weise geklärt werden, mit der die virtuellen Gegenstände visualisiert werden sollen. Zum einen können realistisch wirkende 3D-Objekte mithilfe von Beamern oder Bildschirmen in die Welt projiziert werden. Zum anderen kann dies unter Verwendung von Augmented-Reality-Brillen geschehen. Für die Variante mit den Beamern und Bildschirmen spricht die Tatsache, dass die Qualität der virtuellen Objekte hochwertiger ist als bei den Brillen, da die Auflösung von Bildschirmen der von Brillen stark überlegen ist. Des weiteren ist die Nutzung einer Brille auf einen Anwender beschränkt und kann auf Dauer, vor allem wenn der Nutzer die Verwendung von Augmented-Reality-Brillen nicht gewohnt ist, sehr anstrengend werden. Allerdings hat die Nutzung von Brillen auch einige Vorteile. Beispielsweise ist man hier um einiges flexibler, da man nicht an die Positionierung der Bildschirme oder Beamer gebunden ist. Darüber hinaus ist der visuelle Effekt, welcher in dieser Arbeit mithilfe der Verzerrung der Räume hervorgerufen werden soll, abhängig von der Position des Nutzers. Da diese ohne Weiteres bei Augmented-Reality-Brillen möglich ist, und man keine aufwendige Tracking-Methode implementieren muss, wird in dieser Arbeit eine Augmented-Reality-Brille, nämlich die Microsoft Hololens 1, verwendet.

Auch für die Programmierung des Systems gibt es unterschiedliche Ansätze. Einerseits kann man direkt über Direct 3D für die Hololens entwickeln. Andererseits kann man aber auch die Spiele-Engine Unity dafür nutzen. Der Vorteil bei der Nutzung von Unity liegt darin, dass es bereits einige entwickelte Komponente gibt, welche für die Entwicklung mit der Hololens sehr hilfreich sind. So stellt beispielsweise das MRTK ein Software Development Kit (SDK) dar, welches die Entwicklung von Mixed-Reality Anwendungen in Unity ermöglicht, oder das Augmented Reality Toolkit (ARTK) eine Möglichkeit, mithilfe von Markern virtuelle Objekte an fixe Positionen in der realen Welt zu platzieren. Darüber hinaus verfügt Unity über eine eigene Shader-Entwicklungssprache, welche in Kombination mit der Echtzeit-Übersetzung des Programms in die Entwicklungsumgebung das Debuggen sehr nutzerfreundlich und effizient macht.

3.1 Microsoft Hololens

Die Hololens 1 [Micd] ist eine im Januar 2015 auf den Markt gekommene Augmented-Reality-Brille der Microsoft Corporation. Sie verfügt über eine eigene Rechnereinheit, was es dem Anwender ermöglicht, Programme auf der Hololens zu benutzen, auch wenn diese nicht an einen Computer angeschlossen ist. Da man sich in der für diese Arbeit entwickelten Szenerie frei bewegen können möchte, ist dies ein sehr wichtiger Faktor. Sie verfügt über eine Tiefen-Kamera, wodurch sie in der Lage ist, die Umgebung zu scannen und Objekte zu erkennen. Dies kann beispielsweise für Verdeckungen verwendet werden. Mit einem FOV von 52° gehört sie für damalige Verhältnisse zum Durchschnitt.

Betriebssystem	Windows 10
Prozessor	Intel, unterstützt von der Hololens Processing Unit (HPU)
Speicher	64 Gigabyte Flash
RAM	2 Gigabyte
Gewicht	579 Gramm
Video	2 Kameras, je 2,4 beziehungsweise 1,1 Megapixel

Tabelle 3.1: Technische Details zur Microsoft Hololens 1

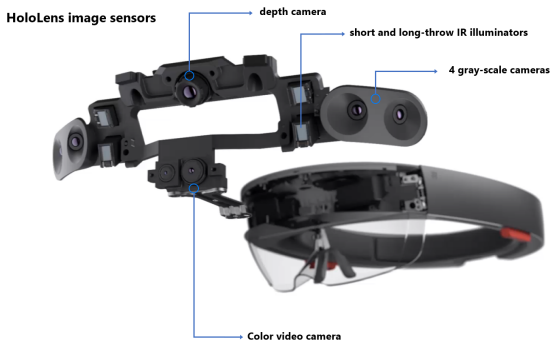


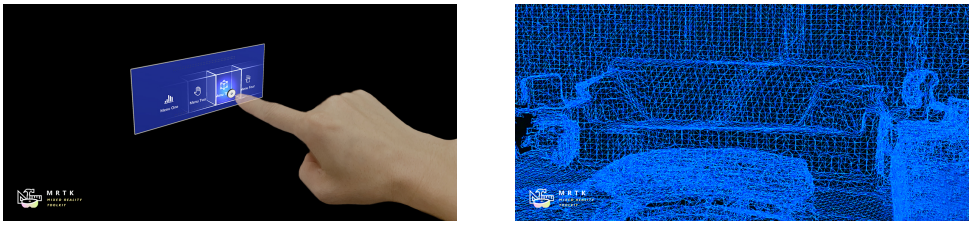
Abbildung 3.1: Bild der Microsoft Hololens 1, in dem die relevante Hardware markiert ist. [Mica]

Die Hololens ermöglicht es dem Anwender auch, mit der Umgebung zu interagieren, wenn es die jeweilige App entsprechend zulässt. Beispielweise ist man mit der Hololens in der Lage, virtuelle Objekte mithilfe von Gesten-Steuerung zu rotieren, skalieren oder zu verschieben. Darüber hinaus ist die Hololens in der Lage, mithilfe der integrierten Mikrofone auf bestimmte Spracheingaben zu reagieren. In Abbildung 3.1 sind die für diese Interaktionen relevanten Sensoren und Kameras markiert. In der unten stehenden Tabelle Tabelle 3.1 sind die technischen Details zu der Hololens 1 aufgelistet [Micf].

3.2 Unity

Bei Unity handelt es sich um eine Spiele-Engine des Unternehmens Unity Technologies [Uni]. Auch wenn Unity hauptsächlich für die Entwicklung von 2D- und 3D-Spielen für Computer, Konsolen und Mobiltelefone genutzt wird, ist man dank der Entwicklung spezieller SDK in der Lage, Unity für die Entwicklung von Augmented- und Virtual-Reality-Programme zu nutzen. Dadurch ist es möglich, virtuelle Objekte in einer Unity Szene zu erstellen und diese dem Anwender mithilfe der Brille visuell anzeigen zu lassen. Diese sogenannten Gameobjects verfügen über einige Komponente, welche für diese Arbeit relevant sind:

1. Transform: Diese Komponente definiert die Dimensionen des Objekts und gibt somit die Skalierung, Orientierung und Position an.
2. Rigid Body: Ermöglicht physikalische Kräfte, wie beispielsweise Gravitation auf die Objekte wirken zu lassen. Ist außerdem notwendig, wenn man die Kollision mehrerer Objekte erfassen möchte.



(a) **(b)**

Abbildung 3.2: Eine Illustration zweier Eigenschaften des MRTK. Links sieht man die Interaktion mit Hologrammen anhand eines Button-Klicks, rechts wird das „Spatial-Awareness“ Feature des MRTK demonstriert, mit dessen Hilfe man die Hologramme mit der physischen Umgebung interagieren lassen kann.

3. Mesh Filter und Mesh Renderer: Notwendig, um die Oberfläche eines Objekts zu manipulieren.
4. Material: Materialien sind Bestandteil des Mesh Renderers. Mit ihnen kann man das Aussehen eines Gameobjects verändern. Die Eigenschaften eines Materials, wie beispielsweise die Farbe oder Textur, können mithilfe von Skripten oder Shadern nach Belieben verändert werden.
5. Box Collider: Eine ungefähre Abschätzung der Grenzen eines Gameobjects, ebenfalls notwendig um Kollisionen zu erfassen.

Gameobjects müssen nicht zwangsweise statische Objekte sein, welche einfach in eine Szene platziert werden. Ihnen können auch Skripte zugewiesen werden, wodurch sie zur Laufzeit auf entsprechende Events, wie beispielsweise Eingaben des Nutzers, reagieren können. Diese Skripte können entweder in C#, JavaScript, oder Boo programmiert werden. Um die Programmierung mit Unity zu erleichtern, wird dem Anwender eine Basis-Klasse namens „MonoBehaviour“ zur Verfügung gestellt, von dem jedes Skript ableiten kann. Damit stehen dem Programmierer hilfreiche Funktionen, wie beispielsweise die Start()-Funktion, welche zu Beginn, oder die Update()-Funktion, welche jeden Frame aufgerufen wird, zur Verfügung.

Um die Entwicklung für die Microsoft Hololens zu ermöglichen, wurde in dieser Arbeit das MRTK verwendet [Mice]. Dadurch wird die Entwicklungsumgebung von Unity um einige Features erweitert, welche von Beginn an nicht unterstützt würden. Darunter fallen unter anderem die Verwendung von Gesten und Audioeingaben, oder auch die Möglichkeit, die Umgebung nach Gegenständen zu scannen um mögliche Verdeckungen sinnvoll zu implementieren (siehe Abbildung 3.2b). Eine weitere wichtige Eigenschaft, welche durch die Nutzung des MRTK ermöglicht wird, ist die Interaktion mit Hologrammen (siehe Abbildung 3.2a).

Wenn keine weiteren Modifikationen an einer Unity-Szene vorgenommen werden, wird diese, wenn sie auf der Hololens gestartet wird, abhängig von der Position und der Ausrichtung des Anwenders instantiiert. Somit bildet die Position, an der sich der Nutzer zu Beginn befindet, den Ursprung der virtuellen Welt und somit auch des Koordinatensystems, welches von dieser Stelle aus aufgezogen wird. Unity arbeitet nativ mit einem rechtshändigen Koordinatensystem, in dem die positive x-Richtung nach rechts, die positive y-Richtung nach oben und die positive z-Richtung nach vorne orientiert sind.

Da in einem Modus des entwickelten Systems die Räume der realen Welt mit denen der virtuellen Karte überlagert werden sollen, müsste dem Nutzer seine genaue Position und Orientierung während des Startens des Systems vorgeschrieben werden. Da dies jedoch nicht wünschenswert ist, schon allein weil eine minimale Fehlpositionierung zu großen Abweichungen in der Überlagerung der Räume führen kann, wird für die Platzierung der virtuellen Karte in der realen Welt das von Long Qian entwickelte ARTK für die Hololens verwendet [AQNK18]. Dieses ermöglicht die Erkennung von vorgefertigten Markern (siehe Abbildung 4.9)), was es möglich macht, das virtuelle Gebäude abhängig von der Positionierung der Marker in der realen Welt zu generieren. Folglich müssen nur die Marker richtig hingelegt und von der Hololens erkannt werden und das System funktioniert unabhängig davon wie der Nutzer während der Erstellung der Szene steht.

3.3 Shader

Bei Shadern handelt es sich um Programme, welche darüber entscheiden, wie die vom Anwender gewünschten Primitive auf dem Bildschirm angezeigt (gerendert) werden sollen. Da es sich bei der Hololens um eine Augmented-Reality-Brille von Microsoft handelt, macht es Sinn, dass als Programmierschnittstelle das Microsoft angehörige Direct 3D verwendet wird. In Abbildung 3.3 wird der Datenfluss durch die Direct3D Pipeline dargestellt, von der Eingabe durch die verschiedenen Stufen hin bis zur Ausgabe. Im Folgenden wird jedoch nur auf diejenigen Stufen eingegangen, die für die Entwicklung mit Unity relevant sind.

3.3.1 Vertex-Shader

In dieser Stufe werden die einzelnen Knoten aus dem Assembler verarbeitet, indem Operationen, wie beispielsweise Transformation, auf ihnen ausgeführt werden. Der Vertex-Shader arbeitet immer an genau einem Knoten und gibt auch wieder genau einen Knoten zurück. Der Vertex-Shader wird immer auf allen Knoten eines Primitives ausgeführt, was ebenfalls benachbarte Knoten mit einschließt, sollten solche Informationen aus der vorangehenden Phase vorliegen. Durch die Manipulation der Koordinaten-Informationen eines Knoten können weitreichende Veränderungen des gesamten Objekts herbeigeführt werden, wie beispielsweise Translation, Skalierung oder Rotation. Um die Geometrie eines Objekts zu verformen muss die Position bestimmter Knoten entsprechend verschoben werden. Daher findet im Vertex-Shader in dieser Arbeit die Berechnung der Verzerrungsalgorithmen statt.

3.3.2 Tesselierung

Tesselierung bezeichnet den Vorgang, in dem eine Oberfläche mit wenig Detailreichtum so unterteilt wird, dass es mit mehr Details gerendert werden kann. Dies ist besonders dann hilfreich, wenn Modelle aus einer geringen Anzahl aus Polygonen bestehen, für eine realistischere oder schönere Ansicht jedoch eine höhere Anzahl vonnöten wäre. Dies wird in Abbildung 3.4 illustriert. Um eine Kugel realistisch darzustellen, genügt ein so geringer Tesselierungs-Faktor, wie er oben links zu sehen ist, nicht aus. Je mehr man jedoch die Oberfläche unterteilt, desto klarer nimmt das Objekt die Form einer Kugel an. Die Tesselierung wird in Direct3D mithilfe von drei Komponenten umgesetzt:

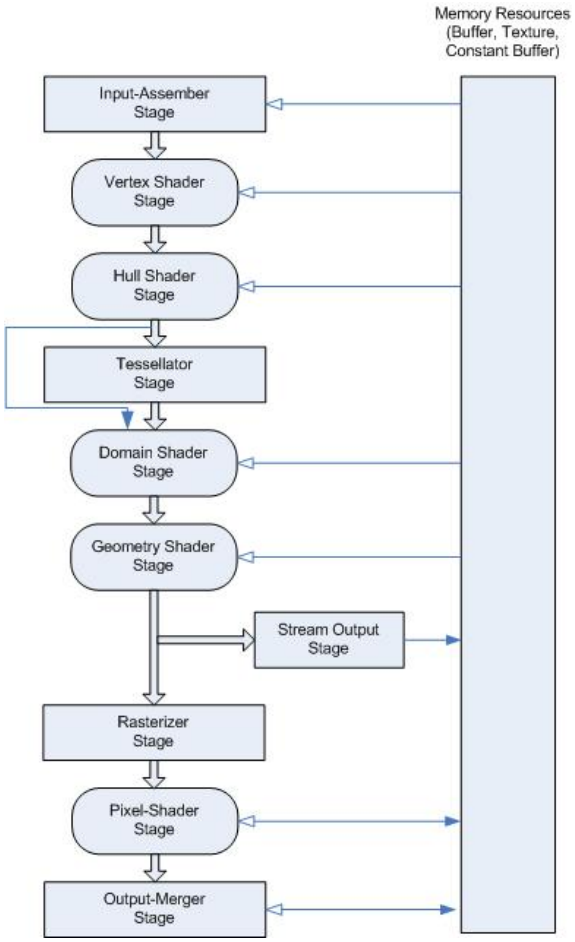


Abbildung 3.3: Die programmierbare Grafik Pipeline von Direct3D 11. Hier sieht man den Datenfluss von der Eingabe über jede eigene Stufe bis hin zur Ausgabe. [Micc]

1. Hull Shader: Hier wird ein Primitiv in einen Patch konvertiert, welcher eine bestimmte Anzahl an Kontrollpunkten besitzt. Abhängig von diesen Kontrollpunkten wird ein Tesselierungs-Faktor bestimmt.
2. Tesselierung: In dieser Phase werden die neuen Knoten generiert. Je höher der Tesselierungs-Faktor aus der vorangegangenen Phase ist, desto mehr Dreiecke entstehen.
3. Domain Shader: Teilt jedem Knoten die zugehörige Position in Form von parametrischen Koordinaten zu.

Ein großer Vorteil von Tesselierung besteht darin, dass man den Detailreichtum einer Oberfläche nach Bedarf anpassen kann. Es macht selten Sinn, Objekte, welche sich in weiter Ferne befinden oder für das Gesamtbild irrelevant sind in hohem Detail zu rendern. Je mehr Details in einem Modell gerendert werden müssen, desto mehr Rechenleistung nimmt dies in Anspruch. Somit ist Tesselierung eine gute Methode, die Belastung der Hardware zu reduzieren. Manche der Verzerrungsalgorithmen in dieser Arbeit besitzen die Anforderung, eigentlich glatte und Detailarme

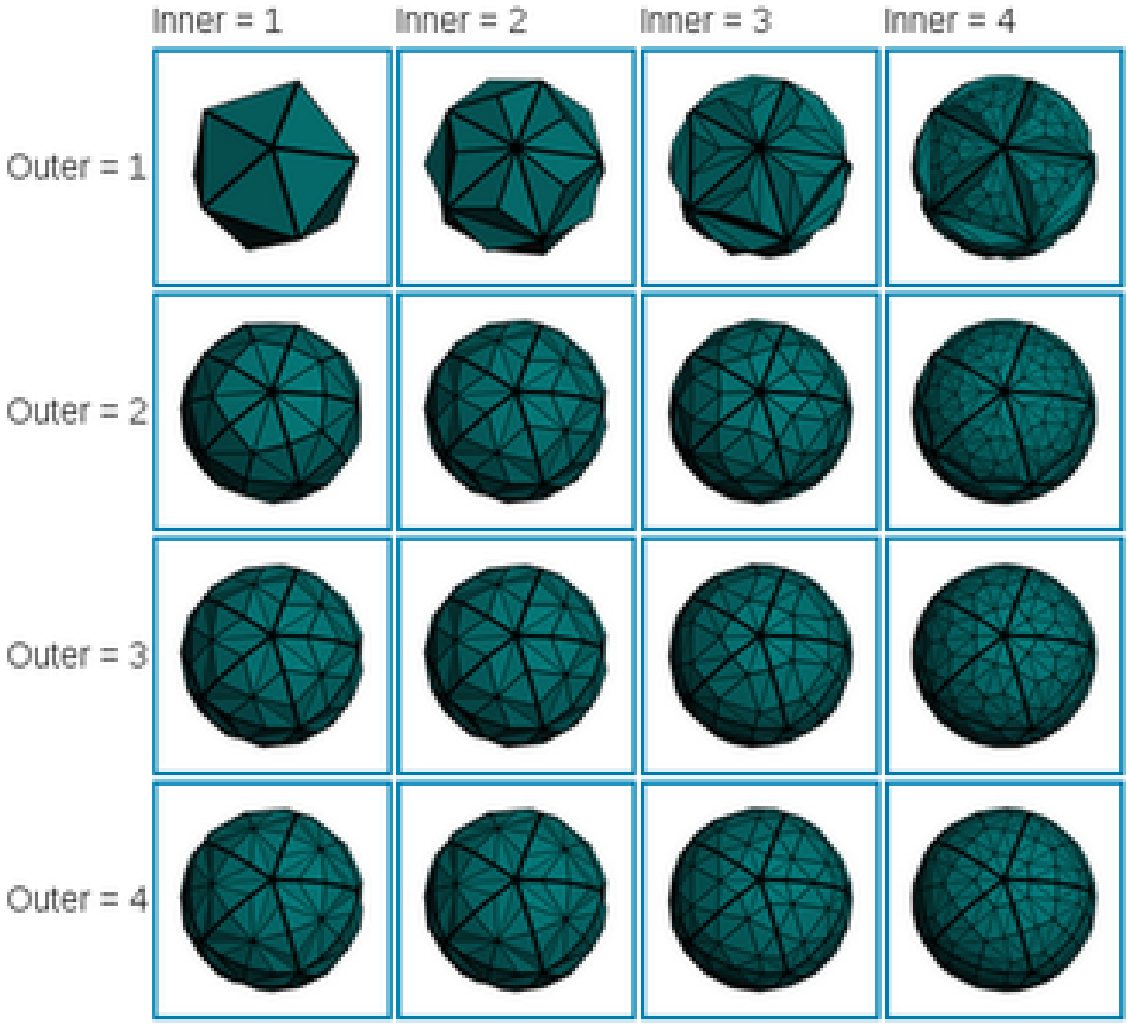


Abbildung 3.4: Eine Illustration unterschiedlicher Tessellierungsfaktoren [Wik].

Oberflächen aufwändig (beispielsweise kreisförmig) zu verformen. Da dies bei Objekten, wie beispielsweise Quadern, welche grundsätzlich nur aus acht Knoten bestehen, nicht sonderlich realitätsnah aussieht, wird hierfür Tesselierung benötigt.

3.3.3 Geometrie-Shader

Im Geometrie-Shader werden, ähnlich wie bei dem Vertex-Shader, Operationen auf den Knoten ausgeführt. Allerdings werden diese Operationen nicht nur auf einem Knoten ausgeführt, wie es im Vertex-Shader der Fall war, sondern auf allen Knoten eines Primitives. Die Operationen können nicht nur auf dem Primitiv ausgeführt werden, welches er als Eingabe erhält, sondern auch, falls vorliegend, auf den Nachbarn. Somit besteht die Ausgabe des Geometrie-Shader aus mehreren Knoten, allerdings stellen diese eine einzelne, festgelegte Topologie dar. Der Geometrie-Shader

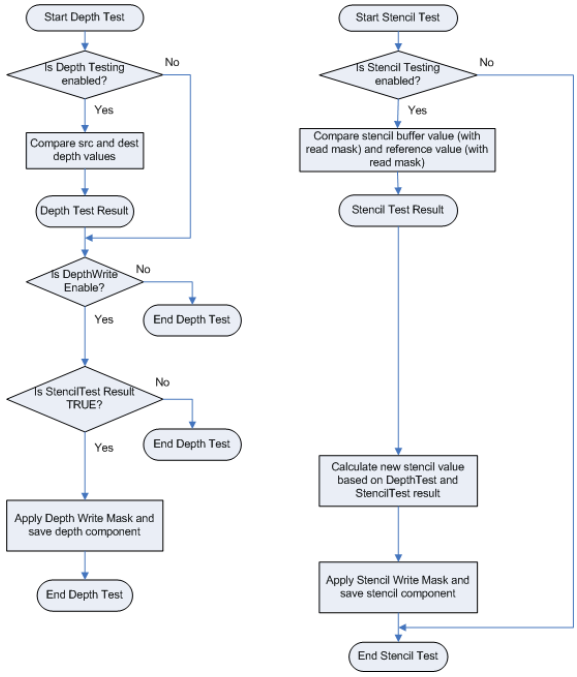


Abbildung 3.5: Ein Überblick über die Funktionsweise des Tiefen-/ Stencilbuffers [Micb].

wird für jedes Primitiv, welches in einer früheren Phase erstellt wurde, einmal aufgerufen. Somit wird er beispielsweise im Falle eines Dreieck Strips einmal für jedes darin befindliche Dreieck aufgerufen.

3.3.4 Fragment-Shader

Im Fragment-Shader findet post-processing und Beleuchtung der einzelnen Pixel statt. Als Eingabe erhält er Textur-Daten, interpolierte Knoten-Werte und andere Variablen, um eine Ausgabe für jeden Pixel zu generieren. Diese spiegelt sich in Form eines Farbwertes wieder, welcher anhand von Interpolation oder genauer Berechnung der für den jeweiligen Pixel relevanten Knoten-Werte ermittelt werden. Die Ausgabe besteht für jede Eingabe aus einem Farb- und einem Tiefen-Wert, welche in der nächsten Phase verarbeitet werden. Es kann aber auch keine Farbe zurück gegeben werden, für den Fall dass der Pixel verworfen wird. Der Fragment-Shader Shader wird in dieser Arbeit verwendet, um mögliche Blending-Effekte zu implementieren.

3.3.5 Ausgabe

Hier wird das Pixel gerendert, basierend auf einer Kombination aus den vorangehenden Informationen aus der Pipeline und dem Tiefen-/ Stencilbuffer. Deren Funktionsweise wird in Abbildung 3.5 erläutert. Mithilfe der Tiefenwerte kann bestimmt werden, welche Pixel am nächsten zu der Kamera liegen und der Stencilwert bestimmt, welche Pixel upgedatet werden müssen.

Nachdem bestimmt wurde, welche Pixel welchen Einfluss auf das angezeigte Bild haben, wird mithilfe von Blending der Farb- und Alphawert der resultierenden Pixel berechnet.

3.3.6 Shader in Unity

Unity stellt dem Entwickler einige Shader-Varianten als Blaupausen zur Verfügung. Eine davon, der sogenannte „Surface-Shader“, hat die Eigenschaft, dass darin, im Gegensatz zu der unbeleuchteten Shader-Variante, bereits die komplexe Berechnung der Beleuchtung vorgegeben ist. Aus diesem Grund, und weil im Surface-Shader Tesselierung möglich ist, wird diese Shader-Variante in dieser Arbeit verwendet. Innerhalb des Surface-Shaders gibt es einzelne Blöcke für die verschiedenen Shader-Stages, welche man in der von Unity benutzten Sprache High-Level Shading Language (HLSL) nach belieben bearbeiten kann. In dieser Arbeit ist in erster Linie der Vertex-Shader von Interesse, da dort die gesamte Verzerrung der Oberflächen unserer Minimap stattfindet, jedoch ist auch für die Implementierung verschiedener Blending-Effekte der Fragment-Shader für diese Arbeit von großem Interesse.

4 Hauptteil

In diesem Kapitel wird die Entwicklung meines Systems behandelt. Der konzeptionelle Part wird in Abschnitt 4.1 vorgestellt. Dabei wird auf die entwickelten Algorithmen eingegangen und auf die Informationen, welche dafür benötigt werden. In Abschnitt 4.2 wird auf die Implementierung in Unity eingegangen. Insbesondere werden hierbei die Umsetzung in den Shadern und verschiedene Unity-Eigenschaften behandelt. In Abschnitt 4.3 wird schließlich eine Übersicht über die Ergebnisse präsentiert. Darunter fallen unter anderem Probleme und Beobachtungen, welche bei den verschiedenen Implementierungen aufgetreten sind.

4.1 Algorithmischer Ansatz

Am Anfang dieser Arbeit steht die Implementierung des "Fisheye-Effekts"(siehe Abschnitt 2.3). Hierbei habe ich mich in dieser Arbeit stark an dem von Sarkar und Brown [SB94] entwickelten Algorithmus orientiert. Da in dieser Arbeit getestet werden soll, wie die Verzerrungseffekte einem Nutzer des Systems bei der Orientierung behilflich sein können, habe ich für die Entwicklung des Systems das 3D-Modell des VISUS Gebäudes benutzt (siehe Abbildung 4.8). Dies war für die Entwicklung der Algorithmen und der Minimaps sehr hilfreich, da ich das System direkt dort implementiert habe und somit jederzeit in der Lage war, mein System direkt vor Ort zu testen. Das Modell des Gebäudes besteht aus sehr simplen Geometrien. Die Räume beispielsweise werden durch einfache Quader dargestellt. Diese Geometrien haben die Eigenschaft, dass sie nicht viele Dreiecke benötigen, um sie realitätsnah zu rendern. Folglich sind affine Transformationen, die man auf die Räume anwendet, einfach zu realisieren. Dazu zählen beispielsweise Translationen, Skalierungen oder Rotationen. Bei der Fisheye-Verzerrung allerdings ist man darauf angewiesen, dass sich die Oberfläche der betroffenen Geometrie konkav verzerren lässt (siehe Abbildung 4.1). Aus diesem Grund ist es nötig, die Oberfläche der Geometrie in kleine Dreiecke zu unterteilen. Dies wird in der sogenannten Tesselierungsphase gemacht, welche in Abschnitt 3.3.1 bereits erklärt wurde. Nach der Tesselierung der Oberfläche können nun alle Knoten der neu entstandenen Dreiecke unabhängig voneinander manipuliert werden, wodurch sich ein konkaver Eindruck entwickeln lässt. Da die Tesselierung jedoch die Geometrie sehr viel komplexer macht, was sich negativ auf die Render-Performance auswirkt, muss hierbei auf unterschiedliche Faktoren geachtet werden. Einerseits muss herausgefunden werden, in wie viele Dreiecke die Oberfläche unterteilt werden muss, um einen realitätsnahen, konkaven Effekt, wenn möglich ohne Kanten, zu erhalten. Zum anderen gilt es zu berücksichtigen, welche Geometrien man überhaupt tesselieren möchte. Da der Fisheye-Effekt nur eine bestimmte Reichweite besitzt, zieht man keinen Vorteil daraus, Räume, welche außerhalb dieses Radius liegen, zu tesselieren.

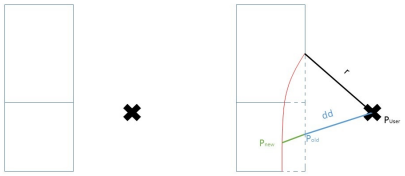


Abbildung 4.1: Konzeptionelle Darstellung der Funktionsweise des Fisheye Algorithmus auf einen Raum. Das X steht für die Position des Anwenders (bzw. der Kamera) und die Rechtecke repräsentieren die Räume. Man sieht dass nach der Anwendung des Fisheye-Algorithmus (rechte Seite) der Raum verzerrt wird.

4.1.1 Verzerrungsalgorithmen

Nachdem die Tesselierung abgeschlossen ist, gilt es den entsprechenden Verzerrungsalgorithmus zu implementieren. Algorithmisch orientiert sich der hier implementierte Fisheye-Effekt an dem von Mike Bostock implementierten Fisheye Algorithmus [Bos]. Dieser nimmt als Eingabe die Position eines Knoten und gibt die transformierte Fisheye-Position zurück. Da der Algorithmus nur für die Verzerrung zweidimensionaler Graphen entwickelt wurde, muss bei der Umsetzung für diese Arbeit eine entsprechende Konvertierung in den dreidimensionalen Raum berücksichtigt werden. Der Fokus der Verzerrung in 3D besteht allerdings in erster Linie auf der xz-Ebene. Man möchte die Höhe der Räume nicht maßgeblich manipulieren. Aus diesem Grund besteht die neue Fisheye-Position eines Knoten nach Anwendung des Algorithmus aus dem transformierten xz-Wert und aus dem unveränderten y-Wert des Knoten. Neben der Position des Knoten wird außerdem über einen Parameter p die Intensität der Verzerrung gesteuert. Durch die Manipulation dieses Parameters kann die Distanz reguliert werden, wie weit ein Knoten vom Anwender weg gedrückt wird. Darüber hinaus gibt es noch einen Radius r welcher beeinflusst, bis zu welcher Distanz der Fisheye-Effekt von der Position des Nutzers aus betrachtet auf die Knoten wirkt. In Gleichung 4.3 wird gezeigt, wie der Faktor k berechnet wird, welcher für diese Regulation der Ausprägung des Effekts zuständig ist. Hierbei steht dd für den Abstand zwischen dem Standort des Anwenders und der betrachteten Knotenposition. Aus Gleichung 4.4 geht schließlich die Berechnung der endgültigen Position hervor. Diese ergibt sich aus der Position des Nutzers, dem Abstand zwischen dem Nutzer und dem betrachteten Knoten d und dem eben berechneten Faktor k. In Abbildung 4.1 ist die konzeptionelle Funktionsweise des Fisheye-Algorithmus auf ein geometrisches Objekt dargestellt.

$$k_0 = \frac{e^p}{(e^p - 1) \cdot r} \tag{4.1}$$

$$k_1 = \frac{p}{r} \tag{4.2}$$

$$k = \frac{k_0 \cdot (1 - e^{-dd \cdot k_1})}{dd} \cdot 0.75 + 0.25 \tag{4.3}$$

$$P_{new} = P_{User} + d \cdot k \tag{4.4}$$

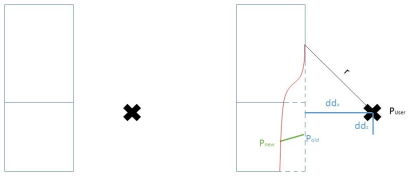


Abbildung 4.2: Darstellung der modifizierten Variante des Fisheye-Algorithmus. Der Unterschied liegt darin, dass der Übergang an den Kanten sehr viel glatter ist als bei der normalen Variante.

Der Fisheye-Algorithmus wirkt sich abhängig von der Distanz zwischen Knoten und Nutzer unterschiedlich aus. Der Effekt ist umso stärker, je geringer der Abstand ist. Dadurch, dass sich der Effekt jedoch nur auf einen vom Radius abhängigen Bereich des Modells auswirkt, ist der Randbereich kritisch. Der Übergang zwischen Fisheye-Effekt und dem normalen, unverzerrten Modell ist bei der normalen Implementierung des Fisheye-Effekts sehr grob, wie man in Abbildung 4.1 auch sehen kann. Um diesen Übergang zu glätten, wurde eine modifizierte Version des Fisheye-Algorithmus entwickelt. Ursprünglich war es so, dass der Abstand zwischen Nutzer und Knoten absolut betrachtet wurde. Das bedeutet, dass die Distanz über deren Ortsvektor als Ganzes berechnet wird. Um die Glättung im Übergang zwischen dem Bereich, in dem der Fisheye-Effekt angewendet wird und dem, bei dem dies nicht der Fall ist, da die Distanz zum Anwender zu groß ist, zu erreichen, müssen die verschiedenen Koordinatenachsen getrennt betrachtet werden. Diese Modifikation wurde ebenfalls von Sarkar und Brown in ihrer Arbeit behandelt [SB94]. Somit werden für die Modifikation der Position eines Knoten dessen x- und z-Koordinate getrennt voneinander manipuliert, genauso wie für die Berechnung der Distanz zwischen Nutzer und Knoten die beiden Koordinatenpaare getrennt voneinander betrachtet werden. Die Anpassungen der Berechnung der neuen Knotenposition ist in den Gleichungen 4.5 - 4.8 zu sehen. Hierbei steht dd_x und dd_z für den Abstand der entsprechenden Koordinate zwischen Nutzer und Knoten, separiert voneinander. Die Auswirkung dieser Modifikation ist in Abbildung 4.2 illustriert.

$$k_x = \frac{k_0 \cdot (1 - e^{-dd_x \cdot k_1})}{dd_x} \cdot 0.75 + 0.25 \tag{4.5}$$

$$P_{newX} = P_{User} + d_x \cdot k_x \tag{4.6}$$

$$k_z = \frac{k_0 \cdot (1 - e^{-dd_z \cdot k_1})}{dd_z} \cdot 0.75 + 0.25 \tag{4.7}$$

$$P_{newZ} = P_{User} + d_z \cdot k_z \tag{4.8}$$

Wie bereits besprochen, wirkt sich der Fisheye-Algorithmus auf die Knoten, welche sich innerhalb des Radius befinden, abhängig von der Distanz unterschiedlich stark aus. Daher besteht der nächste Schritt darin, einen Algorithmus zu implementieren, welcher alle Knoten in einem bestimmten Radius in gleichem Maße verschiebt (siehe Abbildung 4.3). Aus diesem Grund wurde im Anschluss an den modifizierten Fisheye-Algorithmus ein solcher Effekt entwickelt. Für diesen Effekt wird

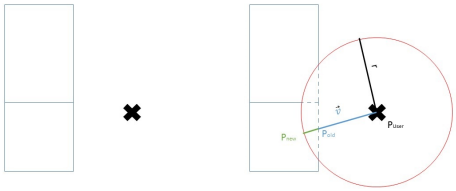


Abbildung 4.3: 2D-Darstellung des implementierten Effekts. Um den Nutzer herum wird eine Kuppel gelegt, welche alle Knoten bis zum Rand des Effekts weg drückt.

ebenfalls die Position des Knoten und die des Anwenders, beziehungsweise der Kamera, benötigt. Des Weiteren wird der Radius r benötigt, mit dem man die Distanz, um die ein Knoten verschoben wird, einstellen kann. Einen Parameter für die Intensität des Effekts benötigt man hier nicht, da alle Knoten innerhalb des Radius auf die gleiche Art manipuliert werden sollen. Die Distanz, um die der Knoten verschoben werden soll, wird, wie in Gleichung 4.9 gezeigt, mithilfe des Faktors b berechnet. Dieser ergibt sich aus dem Radius und dem Abstand zwischen dem Knoten und dem Anwender. Der Faktor b drückt somit den Abstand zwischen dem Knoten und dem Rand der Kuppel dar. Als zusätzlichen Wert benötigt man noch den Richtungsvektor $\vec{v}$ für die Verschiebung der Knoten. Da sich der Raum kreisförmig um den Nutzer herum verzerren soll, muss der Knoten lediglich vom Nutzer weg gedrückt werden. Somit muss man die Differenz aus der Position des Knoten und der Position des Nutzers ziehen, um den Richtungsvektor für die Verschiebung zu erhalten. Mithilfe des normalisierten Richtungsvektors und dem im vorherigen Schritt berechneten Faktor b kann nun die neue Position des Knoten berechnet werden (siehe Gleichung 4.11). Außerdem musste die Verschiebung auf die xz-Ebene eingeschränkt werden, weil eine dreidimensionale Verschiebung nicht sichtbar war (näheres hierzu in Abschnitt 4.3). Somit bleibt der y-Wert eines Knoten unverändert. Da der Effekt eine runde Auswölbung in den Wänden schaffen soll, ist eine hohe Tesselierung, ähnlich der bei dem Fisheye-Effekt, nötig.

$$b = r - d \tag{4.9}$$

$$\vec{v} = P_{Vertex} - P_{User} \tag{4.10}$$

$$P_{new} = P_{old} + b \cdot \frac{\vec{v}}{|\vec{v}|} \tag{4.11}$$

Der Nachteil der implementierten Verzerrungseffekte besteht darin, dass sie eine ungewollte optische Täuschung mit sich brachten. Befindet man sich über einem Raum, so müsste sich dieser durch den Effekt in der Breite ausdehnen. Allerdings war dies für den Anwender nicht erkennbar. Stattdessen hat es den Anschein, als bewege sich der Raum in einer wellenförmigen Bewegung auf den Anwender zu (siehe Abbildung 4.12). Daher bestand der nächste Schritt darin, einen Effekt zu implementieren, welcher die Räume nicht mehr partiell verzerrt, sondern als Ganzes beeinflusst. Aus diesem Grund

werden in einer weiteren Variante, dem sogenannten Control-Effekt [1] die Räume als Ganzes verschoben. Da die Verschiebung im Shader erfolgt und ein Knoten keine Information darüber besitzt, wo er sich in Relation zum gesamten Objekt befindet, ist die Verschiebung komplizierter als eine einfache Translation des kompletten Raumes. Eine Information, über die ein Knoten in Unity jedoch zu jeder Zeit verfügt, ist die Position des Mittelpunktes des Objektes, zu dem er gehört. Um also den kompletten Raum verschieben zu können, wird folgende Vorgehensweise benutzt:

1. Man ermittelt die Ortsvektoren des Knoten (P_{Vertex}) und die des Mittelpunktes des Objekts (P_{Center}) und bildet daraus die Differenz. Mithilfe dieses Vektors o kann nun jeder Knoten auf die Position des Mittelpunktes seines Objekts verschoben werden (siehe Gleichungen 4.12 und 4.13).
2. Wende auf den Knoten, der sich jetzt an seiner neuen Position befindet einen beliebigen Verschiebungsalgorithmus an. Im Fall dieser Arbeit wurde die Fisheye-Verschiebung benutzt, wodurch die Vorgehensweise äquivalent mit den Gleichungen 2.1 - 2.4 funktioniert. Somit befinden sich nun alle Knoten eines Objekts an einem Ort der weiter entfernt vom Nutzer liegt als zuvor.
3. Verschiebe den Knoten mithilfe des aus Punkt 1 ermittelten Vektors wieder zurück auf seine alte Position innerhalb des Objekts.

$$o = P_{Center} - P_{old} \tag{4.12}$$

$$P_{new} = P_{old} + o \tag{4.13}$$

Dieser Effekt hat den Vorteil, dass die Räume nicht aufwendig tesseliert werden müssen. Das liegt daran, dass keine aufwendige Verzerrung der Geometrien stattfindet, sondern die Räume als Ganzes verschoben werden und wie oben bereits besprochen benötigt man für eine so triviale Operation wie Translation keine hohe Anzahl an Knoten, um sie realitätsnah wirken zu lassen. Der Effekt wird in Abbildung 4.4 dargestellt. Neben der Funktionsweise des Effekts kann man hier auch erkennen, dass die Räume auseinander gedrückt werden. Das resultiert aus der Tatsache, dass als Richtungsvektor für die Verschiebung die Differenz aus dem Raummittelpunkt und der Position des Nutzers genommen wird. Da also unterschiedliche Richtungsvektoren auf die Räume wirken, entstehen zwischen ihnen Lücken.

Das Problem an diesem Effekt besteht darin, dass der Anwender aufgrund der Lücken den räumlichen Zusammenhang verliert. Aus diesem Grund wurde eine dynamische Version des Effekts entwickelt, bei der sich die Räume ausdehnen, sobald eine Lücke entsteht, um diese zu schließen. Die Ausdehnung eines Raumes funktioniert so, dass die Position bestimmter Knoten in eine bestimmte Richtung verschoben werden, während andere statisch bleiben. Bewegt man sich also auf zwei Räume zu, welche in Folge dessen auseinander gedrückt werden, wird die Position der Knoten der aneinander anliegenden Wände in Richtung des Nachbars verschoben. Da jedoch Kanten die Knoten verbinden und diese darauf ausgelegt sind, keine Lücken innerhalb eines Objekts entstehen zu lassen, dehnen sich diese aus und vergrößern damit den Raum (siehe Abbildung 4.5). Sobald

[1] Der Name ist angelehnt an das gleichnamige Spiel, in dem die Protagonistin Objekte mithilfe von Telekinese bewegen kann.

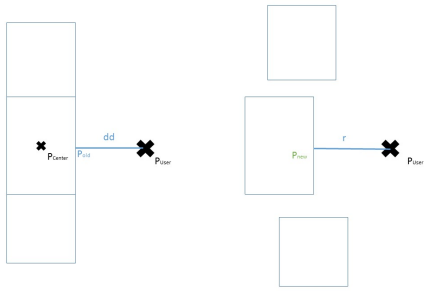


Abbildung 4.4: Darstellung des Control-Effekts. Das x repräsentiert die Nutzerposition und die Rechtecke symbolisieren die Räume. Nach Anwendung des Algorithmus werden die Räume vom Nutzer weg geschoben.

ein Raum in den Radius des Control-Effekts gerät, wird er vom Nutzer weg gedrückt. Da jedoch nur die einzelnen Knoten verschoben werden und nicht das Objekt an sich, funktioniert der Effekt nicht endlos weit, sondern nur bis zu einem gewissen Punkt. Bewegt man sich folglich auf einen Raum zu, der einen Nachbarn hinter sich hat, schiebt man ihn so lange in seinen Nachbarn hinein, bis der Effekt keinen Einfluss mehr auf den betroffenen Raum hat. Ab diesem Punkt werden die Nachbarn dann voneinander weg gedrückt. Wendet man die Ausdehnung des Raumes also an, sobald er in seinen Nachbarn hinein gedrückt wird, vermeidet man die Entstehung der Lücke und verliert den räumlichen Kontext nicht. Für die Menge der Ausdehnung und die Richtung benötigt man allerdings eine Menge Informationen, die teilweise nicht über den Shader abrufbar sind. Die Position des Nutzers und die des Knoten können wie gehabt vom Shader direkt verwendet werden. Die anderen Informationen kann man jedoch nur zur Laufzeit beziehen, da die Knoten für die Anwendung des Effekts, neben Informationen über die eigene Position, auch Informationen über die Position anderer Räume innerhalb des Szene benötigen. Diesen globalen Kontext kann man jedoch nicht im Shader direkt erhalten, sondern benötigt dafür Skripte, welche die Informationen an den Shader weiterleiten. Hierzu gehören die Position aller direkt angrenzenden Nachbarn eines Raumes, sowie die Normale der anliegenden Wand. Die Position des Nachbarraums ist dafür nötig, um die Menge der Ausdehnung des Raumes zu berechnen. Nun kann man für jeden Knoten nicht nur die eigene Verschiebung berechnen, sondern auch die potentielle Verschiebung des Nachbarraumes, sollte dieser in den Radius des Verschiebungseffekts geraten. Somit kennt jeder Knoten zu jeder Zeit die Position des eigenen Mittelpunkts, und die des Nachbarraumes. Daraus lässt sich nun die Größe der Lücke und die Distanz berechnen, um die ein Raum vergrößert werden muss. Außerdem wird dadurch eine kontinuierliche Vergrößerung des Raumes gewährleistet und keine schlagartige. Diese fließende Ausdehnung wird durch den in Gleichung 4.14 gezeigten Faktor *m* hervorgerufen. Dadurch dehnt sich der Raum stetig aus, wenn der Anwender sich auf ihn zubewegt und schrumpft wieder zusammen, wenn man sich von ihm distanziert. Hierbei bezeichnet *dd* den Abstand zwischen dem Anwender und dem Knoten und der Radius *r* ist der gleiche wie auch schon bei dem Fisheye-Algorithmus. Der Faktor m wird auf die Berechnung der neuen Knotenposition hinzu multipliziert, wodurch die fließende Ausdehnung gewährleistet wird. Mithilfe der Normalen $\vec{n}$ in Richtung des Nachbarn und dem Abstand $d_{Nachbar}$ zu ihm lässt sich anschließend die neue Knotenposition berechnen (siehe Gleichung 4.15). Der Faktor *s* beschreibt die Skalierung des gesamten Modells. Durch dessen Berücksichtigung dehnt sich der Raum weniger aus, je kleiner das Modell skaliert wird.

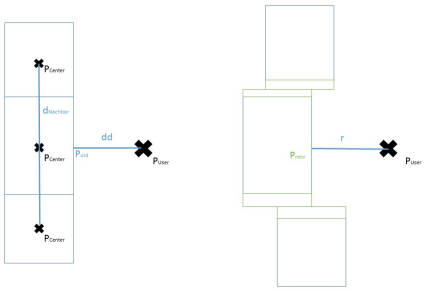


Abbildung 4.5: Hier wird der dynamische Control-Effekt illustriert. Er funktioniert so, dass Räume sich ausdehnen, sobald sich eine Lücke zwischen zwei aneinander anschließenden Räume bildet.

$$m = 1 - \frac{dd}{r} \tag{4.14}$$

$$P_{new} = m \cdot \vec{n} \cdot \frac{d_{Nachbar}}{2 \cdot s} \tag{4.15}$$

An der Vergrößerung des Raumes sind nicht alle Knoten beteiligt. Befindet sich ein Nachbar zum Beispiel links von einem Raum, reicht es vollkommen aus, nur die Position der Knoten zu verändern, welche sich auf der linken Seite des Mittelpunkts befinden. Aus diesem Grund benötigt man den Normalenvektor, der angibt, in welcher Richtung der Nachbarraum liegt und mit dessen Hilfe die richtigen Knoten für die Verschiebung angesprochen werden können. Durch die Nutzung des Normalenvektors ist der Effekt außerdem unabhängig gegen jede Form der Rotation um die y-Achse.

4.1.2 Minimaps

In Verbindung mit diesen Verzerrungsmethoden wurden in dieser Arbeit des Weiteren verschiedene Minimap-Varianten entwickelt. Hier wird zwischen drei unterschiedlichen Modi unterschieden:

1. Eine Minimap, welche statisch in den Raum gelegt wird. Hier kann der Nutzer durch die 3D-Karte laufen und mithilfe der Verzerrungsmethoden bestimmte Einzelheiten im Detail anschauen. Da bei dieser Variante keine Korrelation zwischen der Position des Nutzers im virtuellen und dem realen Gebäude existiert, gibt es hier keinen Positionsindikator.
2. Eine Karte, welche dynamisch auf die Veränderung der Nutzerposition reagiert. Hierbei wird der Nutzer selbst als Positionsindikator genutzt und die Position, in der er sich innerhalb der 3D-Karte befindet soll identisch zu der Position in der realen Welt sein (siehe Abbildung 4.6).
3. Eine Minimap, welche zu jeder Zeit im Sichtfeld des Anwenders ist. Hier ist nicht mehr er selbst der Positionsindikator, sondern eine kleine Figur, welche sich mit dem Anwender mit bewegt. Somit ist die Position des Anwenders in der realen Welt äquivalent zu der Position der Figur in der Karte (siehe Abbildung 4.18).

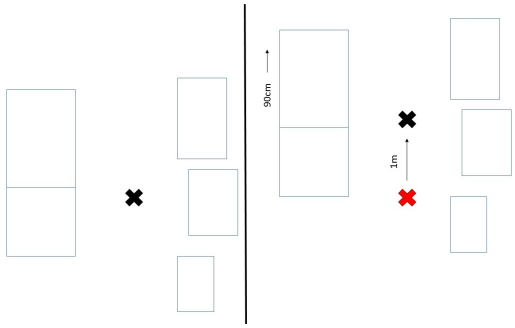


Abbildung 4.6: Hier sieht man die Bewegung eines Nutzers in der Minimap, in der er selbst als Positionsindikator fungiert. In diesem Beispiel wird von einer Skalierung der virtuellen Karten von 10% der Originalgröße ausgegangen. Dadurch muss sich die Karte, wenn sich der Nutzer einen Meter in der realen Welt bewegt, 90cm weit mit bewegen.

Für die erste Kartenvariante ist kein zusätzlicher rechnerischer Aufwand nötig. Die Karte wird lediglich von der Nutzerposition aus gesehen in den Raum platziert, wobei die prinzipielle Positionierung und Orientierung keine Rolle spielen.

Bei Variante zwei wiederum sieht das anders aus, da hier sowohl die Positionierung als auch die Orientierung der Karte von essentieller Bedeutung sind, um einen funktionierenden Effekt zu erzielen. Die initiale Position der Karte muss so gewählt werden, dass die Position des Anwenders in der realen und der virtuellen Welt exakt übereinstimmen. Damit die Bewegung des Anwenders nach der Initialisierung anschließend in beiden Welten identisch ist, muss darüber hinaus auch die Rotation der virtuellen Karte an die reale Welt angepasst werden. Dies ist mithilfe von Markern möglich, welche man an eine beliebige Stelle platzieren, und von deren Position aus die Karte generiert werden kann. Wie genau diese Marker funktionieren, wird in Abschnitt 4.2 erläutert. Für erhöhte Präzision und für die Berechnung der Orientierung werden hier vier, in einem Viereck platzierte, Marker verwendet. Die Position lässt sich folglich sehr einfach als der Mittelpunkt dieser vier Marker wählen. Für die Rotation der Minimap werden die beiden linken Marker benutzt. Indem man die Differenz der beiden Positionen bildet erhält man einen Richtungsvektor. Als nächstes macht man nun das gleiche mit den Positionen zweier benachbarter Räume, und rotiert die Minimap so, dass die beiden Richtungsvektoren parallel zueinander stehen. Nun muss noch sicher gestellt werden, dass die beiden Richtungsvektoren in die gleiche Richtung zeigen, ansonsten könnten Minimap und reale Welt genau spiegelverkehrt zueinander stehen. Ist dies gewährleistet, stimmt die Orientierung der Minimap mit der realen Welt überein.

Neben der Position und der Orientierung muss allerdings auch noch die Geschwindigkeit, mit der man sich in der Welt bewegt angepasst werden. Da man möchte, dass die Position innerhalb der Karte der Position in der realen Welt entspricht, die Minimap jedoch um ein Vielfaches kleiner ist, muss man die Karte zu einem gewissen Grad hin mitbewegen lassen, damit man sich immer an der richtigen relativen Position befindet. Die Position der Minimap lässt sich nach folgender Formel berechnen:

$$P_{new} = P_{old} + f \cdot P_{User}, mit \tag{4.16}$$

$$f = 1 - s/100 \tag{4.17}$$

Bleibt das Gebäude unskaliert, so entspricht das virtuelle Gebäude zu 100% dem Maßstab des realen Gebäudes. Es ist jedoch notwendig, das Modell kleiner zu skalieren, um einen Überblick über das komplette Gebäude zu erhalten. Diese Skalierung wird mit dem Faktor s beschrieben. Ist s beispielsweise 10, so entspricht das Modell 10% der Originalgröße. Folglich bewegt sich in diesem Beispiel die Karte 90% des Weges mit dem Benutzer mit (bewegt er sich also zum Beispiel einen Meter in der realen Welt, so bewegt er sich nur 10cm in der virtuellen Umgebung, siehe Abbildung 4.6). Auf der Gegenseite sieht man aber auch, dass bei einer Skalierung von 100% (das virtuelle Modell und das reale Gebäude stimmen komplett überein) die Position des Gebäudes überhaupt nicht verändert wird. Hier entspricht ein Meter in der realen Welt genau einem Meter in der virtuellen Welt.

Die dritte Kartenvariante ist, zumindest was die Rechnungen angeht, der zweiten sehr ähnlich. Der Unterschied liegt darin, dass die Minimap immer im Sichtbereich des Nutzers sein sollte, um ihn bei der Orientierung zu unterstützen. Somit wird nicht die Kartenpositionierung mithilfe der Marker initialisiert, sondern die Platzierung der Figur, welche die Position des Nutzers repräsentiert. Da die Bewegung ebenso wie bei der zweiten Kartenvariante nur von der Position des Anwenders und der Skalierung der Minimap abhängig ist, kann hierfür ebenso die Gleichung aus 4.16 genutzt werden.

4.2 Implementierung

Wie in Abschnitt 3.2 bereits erwähnt, wurde für die Implementierung des Systems Unity benutzt. Das liegt daran, dass dafür bereits einige SDK entwickelt wurden, welche die Entwicklung mit der Microsoft Hololens ermöglichen. In Abbildung 4.7 wird die in dieser Masterarbeit verwendete Szene gezeigt. Auf der linken Seite eingerahmt sieht man die Bestandteile des MRTK. Darunter gehört vor allem die Main Camera, welche in der Szene für die Hololens steht. Im unteren Rahmen sind verschiedene Marker und ein Controller zu sehen. Diese sind essentiell für die Ortung der Marker, auf deren Position das Gebäude gemappt wird.

Auf der rechten Seite in dem blauen Rahmen ist das Kernstück der Masterarbeit zu sehen. Je nachdem welchen Verzerrungseffekt man benutzen möchte wird das Material mit einem anderen Shader versehen. Um also einen Effekt gleichermaßen auf alle Räume des Gebäudes wirken zu lassen, muss jedem Raum das gleiche Material zugewiesen werden. Diesem Material wird dann der entsprechende Shader angeheftet, und dieser bewirkt dann die Verzerrung der Räume.

Unity besitzt eine Vielzahl verschiedener Shader-Varianten. Für diese Arbeit war es jedoch essentiell notwendig, einen Surface Shader zu benutzen. Mit dessen Hilfe besitzt man nämlich die Möglichkeit, Tesselierung zu implementieren. Das VISUS-Gebäude wurde in Blender gebaut und als Asset in Unity importiert. Die Schnittstelle zwischen den beiden Entwicklungsumgebungen ist praktischerweise direkt gegeben, jedoch benutzen die beiden unterschiedliche Koordinatensysteme, sodass die y- und z-Achse getauscht sind. Darauf musste bei der Manipulation der Vertex-Position geachtet werden. In Abbildung 4.8 kann man das Modell des Gebäudes einmal ohne Tesselierung und einmal mit Tesselierung sehen.

Die meisten Verzerrungseffekte konnten mit ausschließlich den Informationen implementiert werden, welche einem Shader zur Verfügung stehen. Dazu gehören die Position des Knoten, die Position des Gameobjekt, zu dem der Knoten gehört, und die Position der Kamera, welche den Anwender

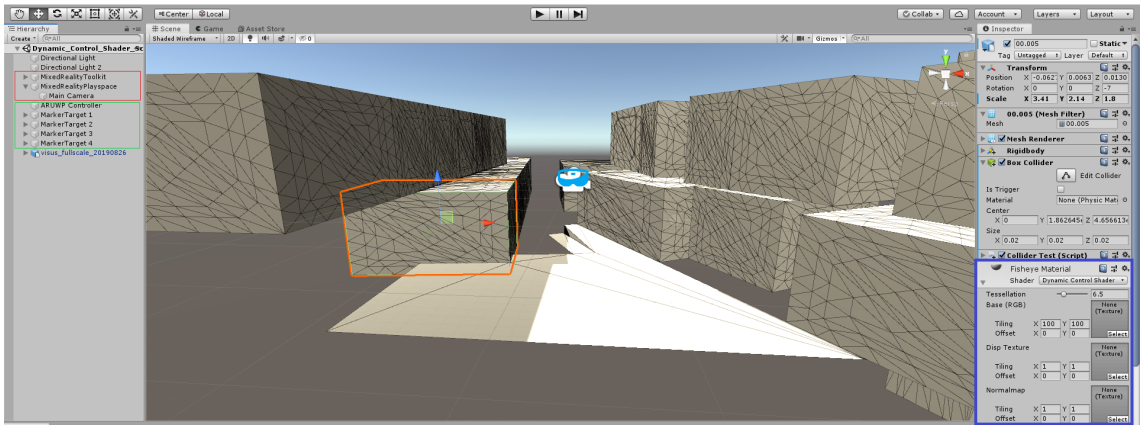


Abbildung 4.7: User Interface von Unity. Zentral ist die Szene zu sehen, in der sich das abspielt, was nachher in der Brille gezeigt werden soll. Auf der rechten Seite werden exemplarisch einige Eigenschaften gezeigt, welche ein Unity Gameobject besitzt. Auf der linken Seite sieht man die verschiedenen Elemente innerhalb der Szene. Die Inhalte des roten Kasten sind Teile des MRTK und unverzichtbar, wenn man für die Hololens entwickeln möchte. Der Inhalt des grünen Kasten sind für die Ortung der Marker (siehe Abbildung 4.9).

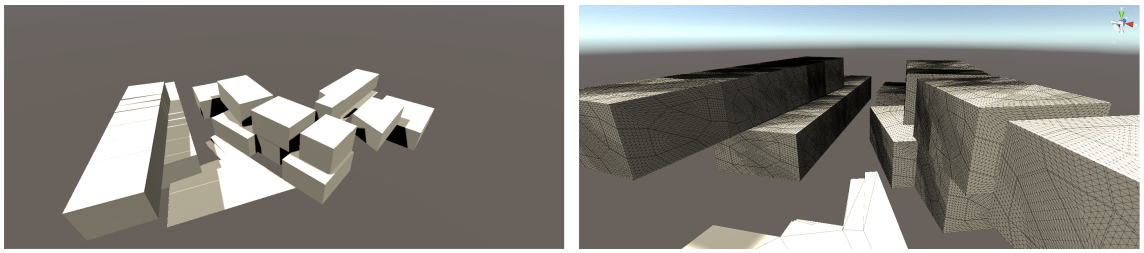


Abbildung 4.8: Modell des VISUS-Gebäudes. Links sieht man das Gebäude während des Play-Mode, wodurch die Tesselierung nicht zu sehen ist. Auf der rechten Seite wird das Gebäude im Editor-Modus gezeigt, wodurch man die Tesselierung sehen kann.

repräsentiert. Kein Verzerrungsalgorihtmus, außer dem modifizierten Control-Effekt benötigte zusätzliche Informationen. Allerdings verfügt ein Shader über keine Informationen über benachbarte Gameobjekte. Da für die Anpassung der Größe eines Raumes bei dem dynamischen, Lücken füllenden Control-Effekt die Position und Richtung des benachbarten Gameobjekts notwendig ist, musste man dem Shader diese Informationen zukommen lassen. Dies war jedoch nur zur Laufzeit mithilfe eines Skripts möglich. Dieses Skript muss jedem Raum individuell angehängt werden (siehe Abbildung 4.7). Darüber hinaus muss jedem Raum auch noch ein Rigid Body und ein Box Collider angefügt werden. Der Rigid Body ist dafür da, um physikalische Effekte wie beispielsweise Gravitation oder Kollisionen zu erkennen und auf das Objekt wirken zu lassen. Der Box Collider setzt die Grenzen eines Gamobjekts. Diese Grenzen sind in Abbildung 4.10 zu sehen. Mithilfe dieser beiden von Unity angebotenen Hilfsmittel ist man in der Lage, Schnittstellen zweier Objekte zu ermitteln. Damit erhält man nicht nur die ID der betroffenen Räume, und somit auch deren Position, sondern auch die Richtung, in der sich der Nachbar befindet. Für die Position und die Normale in Richtung des Nachbarn muss man im Shader dann separate Variablen erstellen, welche

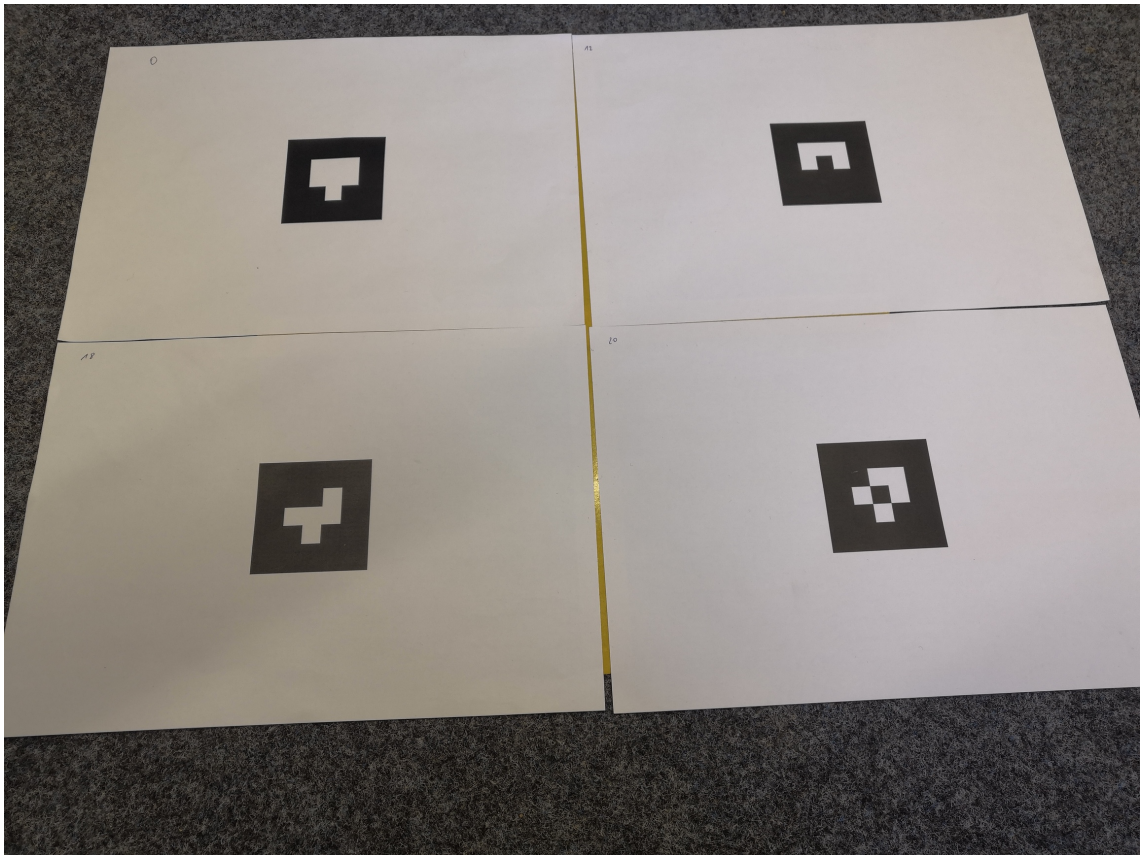


Abbildung 4.9: Die Marker, welche für die exakte Positionierung und Orientierung des Gebäudes in der realen Welt genutzt werden.

man zur Laufzeit dann von Seiten des Skripts aus füttern kann. Wichtig in Verbindung mit den Box Collidern ist, dass sie, wenn man die Größe des Modells ändert, mit skalieren. Verkleinert man also das komplette Modell, schrumpfen auch die Box Collider entsprechend mit. Sie bleiben jedoch komplett unabhängig von allen Manipulationen, welche im Shader stattfinden. Egal wie man die einzelnen Knoten manipuliert, die Größe der Box Collider verändert sich nicht. Da die Box Collider allerdings größer sind als deren Objekte, kann es jedoch passieren, dass bei genügend kleiner Skalierung Räume als direkt angrenzende Nachbarn erkannt werden, obwohl zwischen ihnen im Modell eigentlich eine Lücke sein sollte. Das kommt dann von einer Überlappung der Box Collider. In diesem Fall muss man im Skript dann explizit die Kollisionserkennung dieser Räume ignorieren, beispielsweise über einen Ausschluss zweier IDs.

Wie bereits erwähnt bleibt die Bounding Box eines Objekts unverändert, egal was mit den Knoten im Shader gemacht wird. Dies führt jedoch zu einem unerwünschten Nebeneffekt, nämlich dem Verschwinden von Räumen, obwohl sie eigentlich noch sichtbar sein sollten. Startet man die Szene auf der Hololens und bewegt sich auf einen Raum zu, so löst das die Ausdehnung des Raumes aus, wenn man den dynamischen Control-Effekt verwendet. Unity nutzt jedoch standardmäßig ein Feature, welches Frustum Culling genannt wird. Dabei wird zur Laufzeit durchgehend geprüft, ob sich ein Objekt im Sichtbereich des Anwenders befindet. Ist dies nicht der Fall, wird das Objekt nicht gerendert. Dies ist prinzipiell wünschenswert, da Objekte, die außerhalb des Sichtbereichs

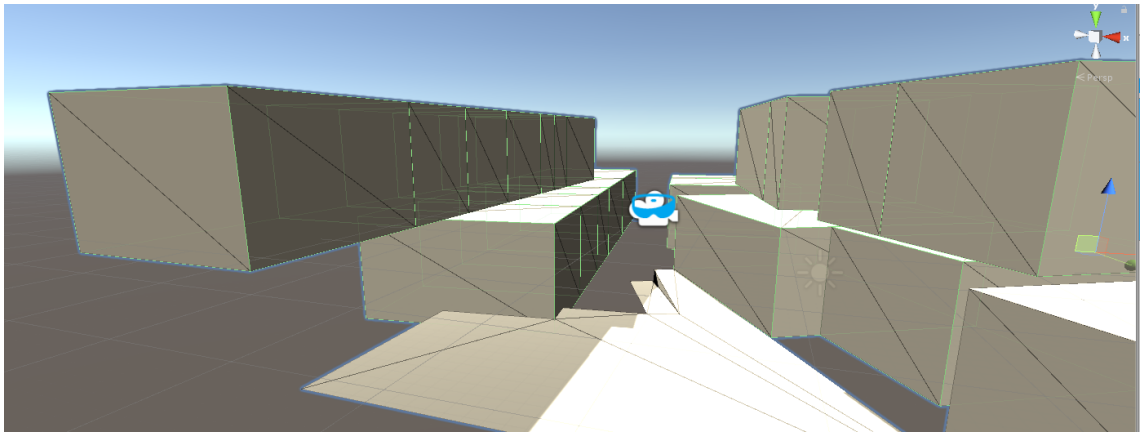


Abbildung 4.10: Eine Darstellung der Box Collider, welche für die Kollisionserkennung benötigt werden. Diese stellen die Grenzen eines Objekts dar und verändern sich nicht, egal was im Shader mit den Knoten gemacht wird.

gerendert werden unnötig Rechenleistung verbrauchen, in unserem Fall jedoch hinderlich. Da das Objekt durch die Ausdehnung der Kanten größer als dessen Bounding Box wird, kann es passieren dass man Knoten und Kanten betrachten möchte, welche sich außerhalb der Box befinden. Gerät der äußere Rand der Box jedoch außerhalb des Sichtbereichs der Hololens, wird das komplette Objekt verworfen, obwohl eigentlich ein paar Knoten und Kanten noch sichtbar sein müssten. Da das Modell des VISUSGebäudes nicht allzu groß ist, profitieren wir überhaupt nicht so deutlich von diesem eingebauten Culling Feature. Allerdings besitzt Unity keine Möglichkeit, Culling zu deaktivieren. Da jedoch damit oft schlagartig Räume verschwinden und dies sich auf die Anwenderfreundlichkeit des Systems sehr negativ auswirkt, musste trotzdem eine Möglichkeit gefunden werden, dies zu verhindern. Dies war mithilfe eines Workaround möglich, indem die Bounding Box der einzelnen Räume extrem hoch gesetzt wurde. Dadurch wurden mithilfe eines Skripts die Grenzen jedes Raumes auf eine Größe gesetzt, welche die Größe des gesamten Modells um ein Vielfaches übersteigt. Somit befindet sich die Bounding Box jedes Raumes durchweg im Blickfeld und wird nicht verworfen. Praktischerweise wird bei der Vergrößerung dieser Bounding Box nicht die Größe des Box Collider, welcher für die Kollision genutzt wird, manipuliert. Dieser kann unverändert bleiben, wodurch die Nachbarschaftserkennung weiterhin fehlerfrei funktioniert. Wenn man es jedoch zukünftig mit größeren Modellen zu tun hat, muss für die Größe der Bounding Boxen vermutlich ein sinnvoller Wert gefunden werden, damit zumindest für Objekte, die sich deutlich außerhalb des Sichtfeldes befinden, Culling angewendet wird.

Ein weiterer Faktor der sich kritisch auf die Performance auswirkt ist das ARTK. Dieses integriert drei essentielle Scripte in die Unity-Szene:

1. Controller: Dieser legt fest, welche Kamera für das Tracking der Marker benutzt wird und definiert die Art der Marker, die getrackt werden. Unterschieden wird dabei zwischen einfachen Matrix-Markern, wie sie beispielsweise in Abbildung 4.9 zu sehen sind, oder Abstraktionen, wie zum Beispiel Schriftzeichen.

2. Marker: Hier werden die verschiedenen Marker initialisiert. Für jeden Marker, den man in der Szene benutzen möchte, benötigt man jeweils dieses Skript. Angegeben wird hier die ID des Matrix-Markers, beziehungsweise der Name des Schriftzeichen-Markers, und dessen Größe, wenn er dann getrackt werden soll.
3. Video: Erstellt eine separate Sichtebene, auf welcher eine Vorschau der generierten Szene gezeigt wird.

Kritisch für die Framerate ist vor allem das Video-Skript, da hier die komplette Szene erneut gerendert werden muss. Allerdings läuft das ARTK nicht ohne dieses Skript, auch wenn man diese zusätzliche Ebene überhaupt nicht möchte. Da alle Features des ARTK nach dem erfolgreichen Tracking der Marker nicht mehr von belang für das System sind, habe ich mich für dies Verbesserung der Framerate zur Laufzeit dazu entschlossen, das komplette ARTK zu deaktivieren, sobald der Tracking Prozess abgeschlossen wurde. Dies führte teilweise zu einer Verbesserung der Framerate um bis zu 10 FPS.

Der letzte wichtige Faktor für eine gute Performance, auf den ich bei meinem System eingehen möchte, ist die Tesselierung. Es ist extrem wichtig, diese für jede Form der Verzerrung sinnvoll zu wählen. Je mehr die Oberfläche eines Objekts tesseliert wird, desto mehr mehr Knoten gibt es, auf denen Operationen ausgeführt werden. Dies zieht logischerweise Einbußen in der Performance nach sich. Dementsprechend sollte man sich genau überlegen, bei welchem der Verzerrungsalgorithmen ein hohes Level an Tesselierung notwendig ist. Prinzipiell gilt, je mehr die Oberfläche eines Objekts verformt werden soll, desto mehr möchte ich diese in kleinere Dreiecke unterteilen. Folglich muss beispielsweise bei dem Fisheye-Effekt, welcher die komplette Oberfläche aus kerbt, ein recht hohes Level an Tesselierung gewählt werden. Bei dem Control-Effekt wiederum, bei dem nur die Räume als Ganzes verschoben werden, benötigt man überhaupt keine Tesselierung.

4.3 Diskussion

4.3.1 Effekte und Problemstellungen

Bei der Implementierung der verschiedenen Verzerrungsmethoden wurden einige interessante Beobachtungen gemacht. Begonnen wurde mit der Umsetzung des Fisheye-Effekts, welcher als Basis der Verzerrungsmethoden genutzt werden sollte. In Abbildung 4.11 ist der Effekt, angewandt auf das Modell des VISUS Gebäudes dargestellt. Man kann hier sehr gut die Kurve erkennen, welche die Fläche der Räume aus kerbt. Befindet man sich nun vor einem Raum, welchen man genauer betrachten möchte, so wird dieser relevante Bereich für den Anwender vergrößert und wölbt sich um ihn herum. So gesehen hatte der Fisheye Algorithmus den gewünschten Effekt.

Allerdings wurde hier schnell ein Problem bemerkbar, welches bei den in dieser Arbeit implementierten visuellen Effekten noch öfters auftreten sollte, und zwar die visuelle Wahrnehmung des Anwenders. Da der Algorithmus für die 2D Ebene konfiguriert wurde, dürfte er keinen Einfluss auf die y-Koordinaten der Knoten des Raumes haben. Der Sinn besteht lediglich darin, den Raum in die Tiefe beziehungsweise Breite (also die xz-Ebene) zu verkrümmen. Sieht man sich jedoch Abbildung 4.12 an, so bemerkt man dass sich die Räume, sobald man sich über ihnen befindet, wie

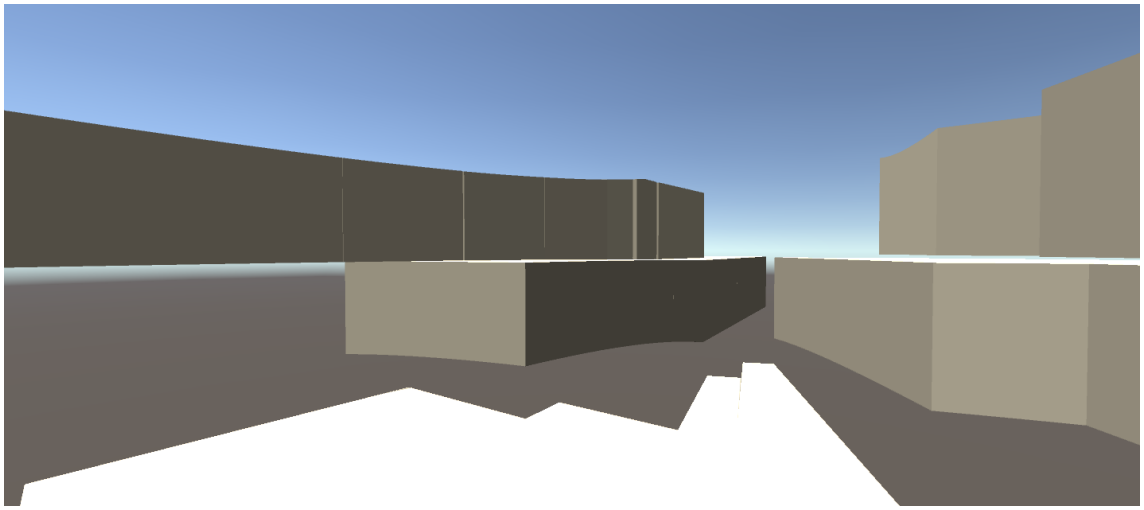


Abbildung 4.11: Eine Illustration des Fisheye-Effekts, welcher die Wände in einem gewissen Radius um den Anwender herum verzerrt.

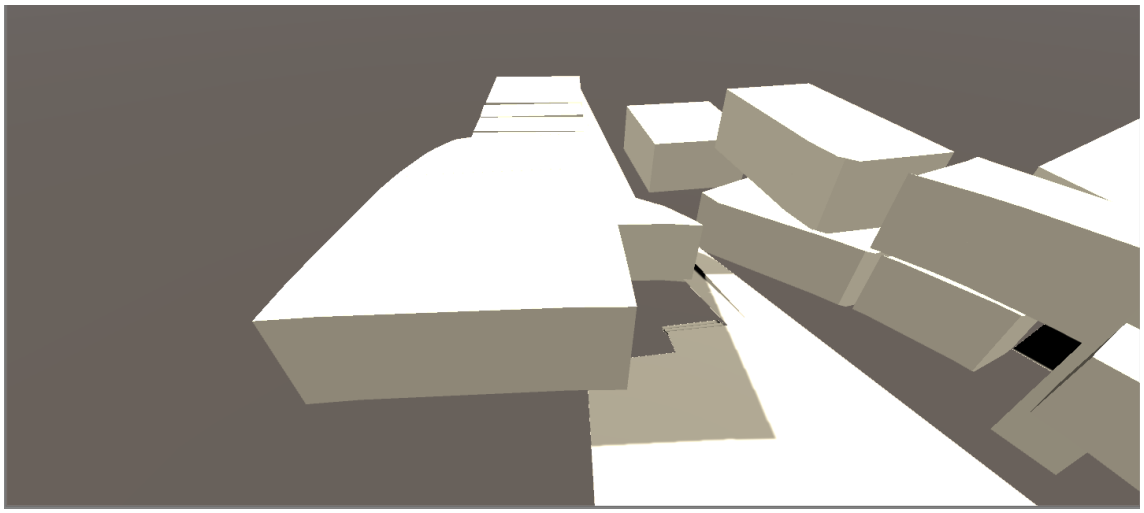


Abbildung 4.12: Eine Darstellung des Welleneffekts, welcher bei der Betrachtung des Modells nach Anwendung des Fisheye-Effekts auftritt, sobald man das Modell von oben herab betrachtet.

eine Welle auf sich zu bewegen. Der gewünschte wahrgenommene Effekt besteht darin, dass sich der Raum den Anwender herum wölbt, sobald man über oder in ihm steht. Dies hätte zur Folge, dass der Raum breiter wird. Die Vermutung die hier nahe liegt is die, dass sich der Raum tatsächlich in die Breit ausdehnt, das jedoch aus der Perspektive des Anwenders nicht sichtbar ist. Für ihn sieht es so aus, als würde sich der Raum aufbäumen, anstatt breiter zu werden. Es handelt sich daher um eine optische Täuschung, da der menschliche Verstand nicht in der Lage ist, diesen Effekt in der virtuellen Umgebung richtig zu realisieren.

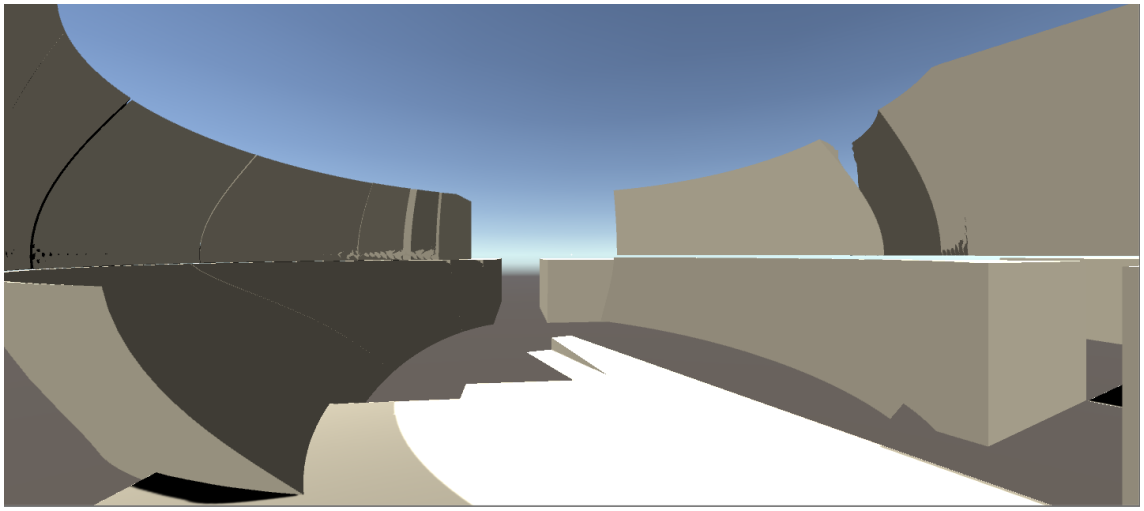


Abbildung 4.13: Eine Darstellung des Dome Effekts, allerdings mit einer fixen Position in der Mitte des Raumes. Um diesen Punkt wird eine Kuppel gelegt, welche alle Knoten der Räume von diesem Punkt gleichmäßig weg drückt.

Um diese Vermutung zu bestärken und sicher zu gehen, dass dies nicht nur eine Begleiterscheinung des Fisheye-Effekts ist, sollte als nächstes ein anderer Effekt implementiert werden, welcher die Räume zumindest in ähnlicher Weise verzerrt. Bei dem Fisheye Algorithmus werden die Merkmale in der Nähe des Betrachters ähnlich wie bei einer Lupe vergrößert, während er die Räume nach außen hin zunehmend verzerrt, da sich dort weniger relevante Informationen befinden. Die Idee des nächsten Effekts bestand darin, alle Knoten in einem bestimmten Radius um den Anwender herum gleichmäßig weg zu drücken. Da dies den Anschein erweckt, dass man eine Kuppel um den Anwender herum legt, wird dieser Effekt im Folgenden als "DomeEffekt bezeichnet. Der algorithmische Teile dieses Effekts wurde bereits in Abschnitt 4.1 betrachtet. Das Interessante bei der Implementierung in Unity war hier jedoch, dass der Effekt nicht sichtbar war. Egal wie groß man den Radius oder die stärke der Verzerrung einstellte, es war kein sichtbarer Effekt zu sehen, solange man den Benutzer als Ursprung des Effekts benutzte. Um sicher zu gehen dass der Effekt auch wirklich so funktioniert wie er sollte, wurde testweise nicht die Position des Nutzers (welche durch die Kamera in der Unity Szene dargestellt wird) genutzt, sondern ein beliebiger Punkt im Raum. Wie man in Abbildung 4.13 sehen kann, funktioniert der Effekt in dieser Konstellation problemlos.

Somit stellt sich die Frage, woran es liegt dass der Effekt von der Position des Nutzers aus nicht sichtbar ist. Die Vermutung welche hier Nahe liegt ist, dass es an der Perspektive liegt. Wenn man alle Punkte vom Anwender komplett gleichmäßig weg schiebt und das auch immer in Richtung des Blickvektors, so ändert sich aus Sicht des Anwenders lediglich die Tiefe der Knoten relativ zu der eigenen Position. Da die Tiefenwahrnehmung im virtuellen Raum jedoch möglicherweise nicht richtig funktioniert kann man von der eigenen Position aus keine Veränderung in der Szene wahrnehmen. Um den Effekt aus Sicht des Anwenders sichtbar zu machen, darf man die Knoten also nicht gleichmäßig entlang des Blickstrahls verschieben. Da für den von uns gewünschten Effekt in erster Linie die Verzerrung innerhalb der xz-Ebene relevant ist, wurde der Algorithmus folglich

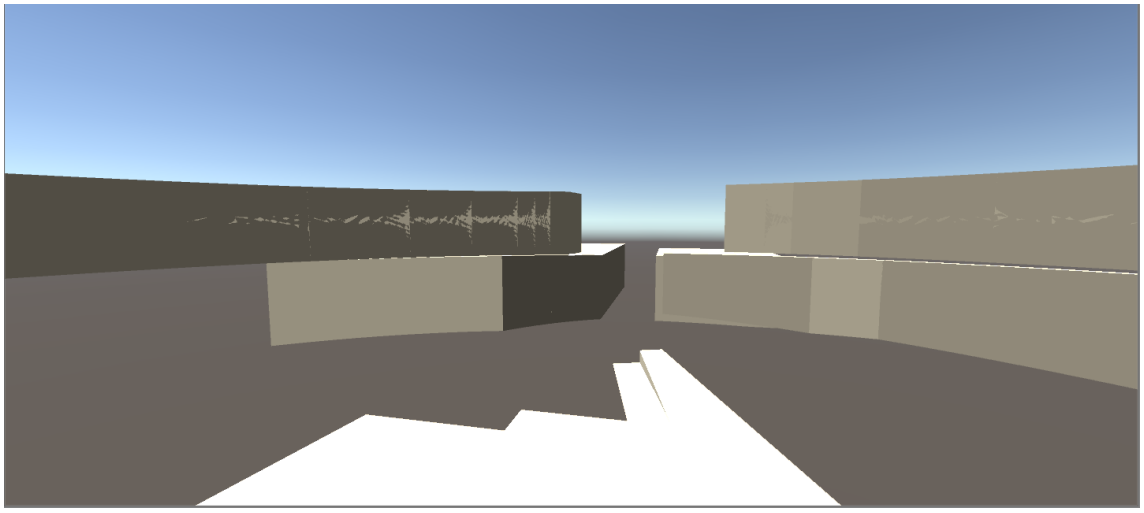


Abbildung 4.14: Eine Darstellung des Dome Effekts, dieses Mal aus Sicht des Anwenders.

dahingehend verändert, dass die y-Koordinate unverändert blieb. Dies resultiert dann in einer Art SStauchung"der Räume, wenn sie vom Nutzer weg geschoben werden, allerdings ist der Effekt dadurch sichtbar (siehe Abbildung 4.14)

Allerdings resultiert auch dieser Verzerrungseffekt in der wellenförmigen Ausprägung, sobald man sich über den Räumen befindet, wie es auch schon beim Fisheye-Effekt der Fall war. Die Vermutung liegt also nahe, dass dieser Effekt von der Konversion eines eigentlich dreidimensionalen Effekts in den zweidimensionalen Raum herbeigeführt wird.

Aus diesem Grund will ich mich bei dem nächsten Effekt von den kurvenförmigen Effekten distanzieren und eine für die menschliche Wahrnehmung einfacheren Effekt, nämlich eine einfache Translation, implementieren. Ziel dieses Effekts besteht darin, den für den Anwender relevanten Raum vor sich im Zentrum zu haben und alle anderen Räume als ganzes von sich weg zu drücken. Somit liegt der Fokus auch wirklich nur auf dem vom Benutzer gewünschten Bereich. Die erste Version des implementierten Effekts ist in Abbildung 4.15 zu sehen. Das Problem an dieser Umsetzung bestand darin, dass Lücken zwischen den Räumen aufgerissen wurden, obwohl dort im eigentlichen Modell überhaupt keine sind.

Da diese Lücken allerdings das räumliche Verständnis des Anwenders über das Modell negativ beeinflussen (schließlich kann hier nicht unterschieden werden, ob eine Lücke im tatsächlichen Modell vorhanden ist oder nicht), wird eine dynamische Version dieses Effekts entwickelt, welche die Lücken zur Laufzeit schließt (siehe Abbildung 4.16). Hier werden die Lücken geschlossen, indem die Räume, sobald sie in den Bereich des Translationseffekts geraten, ausgedehnt werden. Dieser Effekt erfüllt prinzipiell die Bedingungen, welche er einhalten musste. Zum einen stellt er den Raum, welchen der Anwender Betrachten möchte in den Vordergrund, indem er den gewünschten Raum vor sich, und alle anderen Räume von sich weg schiebt. Zum anderen bleibt das räumliche Verständnis innerhalb des Modells erhalten. Doch auch hier gab es für den Anwender Probleme in der Tiefenwahrnehmung. Bei dem Effekt werden alle Räume des Modells verschoben, der Boden bewegt sich jedoch nicht. Sobald die Szene geladen ist und der Anwender inmitten des Modells steht, stellt er eine Relation aller innerhalb dieses Modells befindlichen Objekte her. Da sich diese Relationen im Verlauf der Nutzung jedoch ungleichmäßig ändern (Räume werden verschoben, der

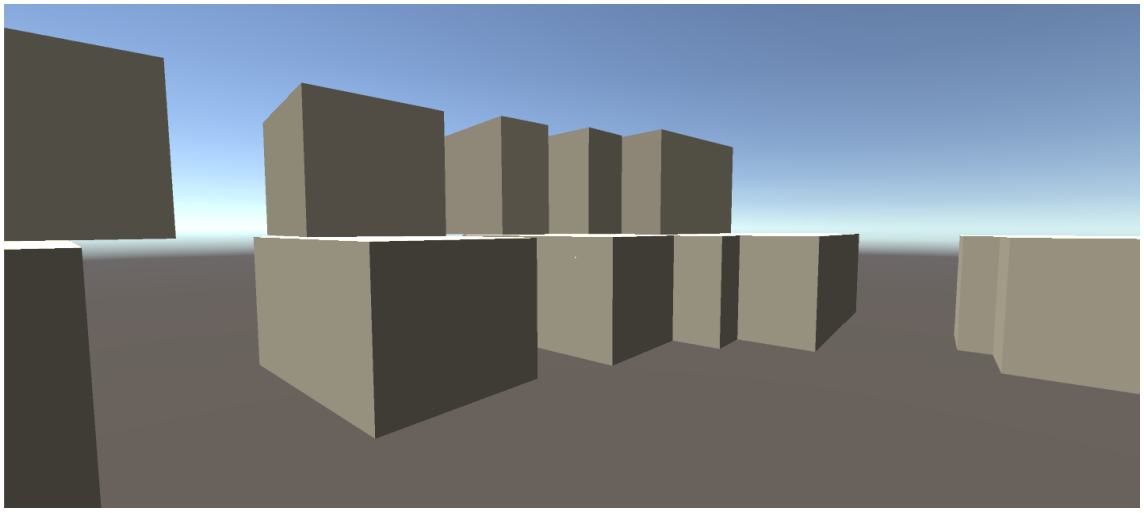


Abbildung 4.15: Hier eine beispielhafte Illustration des Effekt, bei dem die Räume als ganzes verschoben werden. Ziel hierbei ist es, die Räume, welche sich unmittelbar bei der Nutzerposition befinden so zu verschieben, dass er sie in Gänze betrachten kann. Bei dieser Version des Effekts entstehen noch Lücken zwischen den Räumen, obwohl in dem realen Modell diese eigentlich aneinander anschließen.

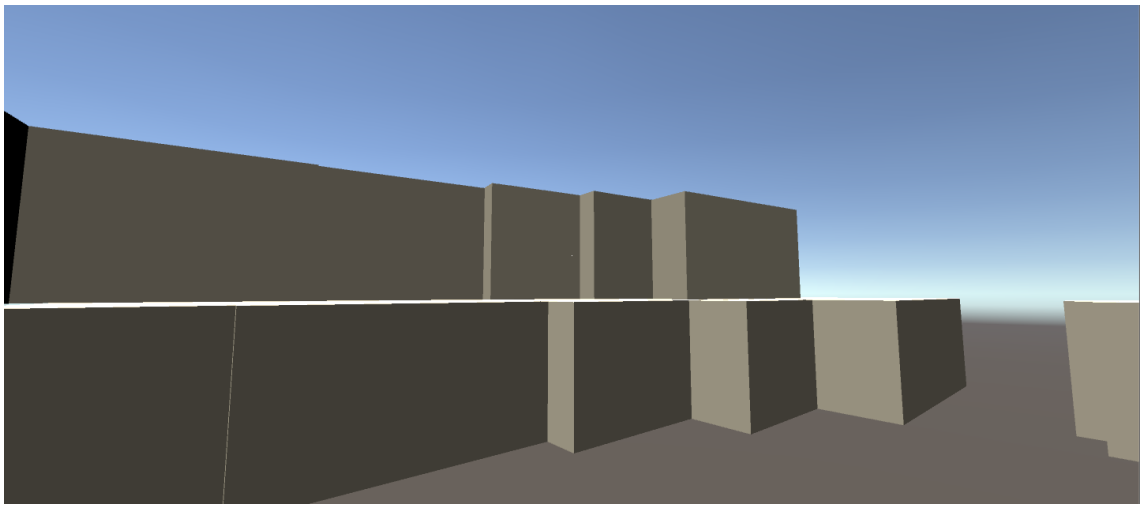


Abbildung 4.16: Hier sieht man den Lücken schließenden Translationseffekt.

Boden bleibt gleich), wirkt sich der Effekt anders aus, als es der Verstand eigentlich erwarten würde. Aus diesem Grund fühlt sich der Effekt, auch wenn er seinen Zweck prinzipiell erfüllt, unnatürlich an.

In Abbildung 4.17 ist der Effekt durch die Hololens zu sehen, also genau so, wie der Nutzer ihn während der Anwendung zu sehen bekommt.

Abbildung 4.17: Hier sieht man den Lücken schließenden Translationseffekt aus der Sicht des Anwenders durch die Hololens. Auf der linken Seite wird lediglich das Gebäude gezeigt, ohne Einwirkung des Effekts, während man auf der rechten Seite den Effekt in Aktion sieht.

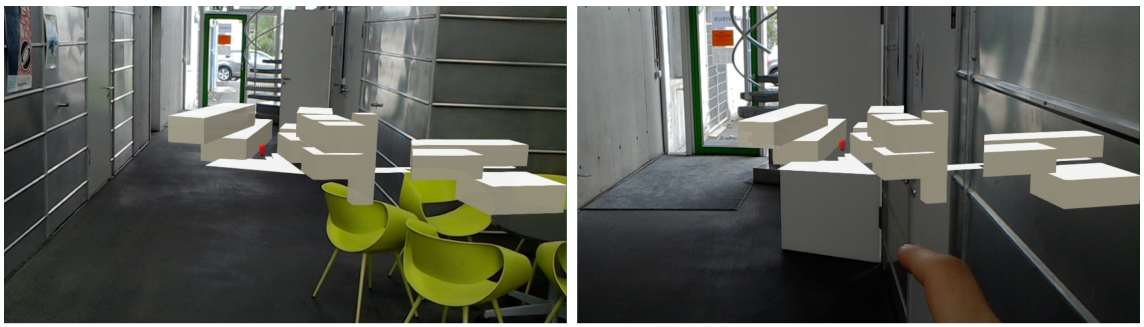


Abbildung 4.18: Hier wird die Minimap gezeigt, in der ein roter Zylinder die Nutzerposition in der realen Welt repräsentiert. Im Übergang von dem linken zu dem rechten Bild kann man sehen, wie sich der Nutzer weiter in das Gebäude hinein bewegt, was auch in einer Verschiebung der Position des Indikators führt.

4.3.2 Entwurf einer Studie

Zum Abschluss der Entwicklung eines solchen Systems ist es normalerweise sinnvoll, eine Studie durchzuführen, um zu testen, ob die gegebenen Anforderungen von dem System erfüllt werden, beziehungsweise ob der gedachte Ansatz überhaupt der richtige war. Gerade ein System, dessen Nutzen darin besteht, Menschen in einer bestimmten Aufgabe zu unterstützen, so wie es hier der Fall ist, kann unmöglich vollumfänglich von dem Entwickler des Systems alleine getestet werden. Es benötigt eine Menge unbeteiligter Personen, welche das System testen, dieses neutral bewerten und dadurch objektives Feedback geben können. Leider war eine solche Studie während des Bearbeitungszeitraumes nicht möglich, da es mir durch die COVID-19 Pandemie [RKI] und deren gesetzlicher Auflagen nicht gestattet war, die für die Ausführung einer Studie notwendigen Grundvoraussetzungen zu erfüllen. Dadurch zählt beispielsweise schon allein die Benutzung einer AR-Brille von mehr als einer Person.

Für das Testen des implementierten Systems bietet sich vor allem eine Suchaufgabe an, da dies die primäre Funktionalität einer Karte ist. Folglich sollte die Aufgabe des Probanden darin bestehen, mehrere Räume innerhalb des Gebäudes zu finden. Diese Räume werden ihm abhängig von dem

jeweiligen Modus auf unterschiedliche Art und Weise visualisiert. Gestartet werden sollte die Suche bei jedem Anlauf immer an einem fixen Punkt. Die dem Anwender zur Verfügung stehenden Varianten, die ihn bei der Aufgabe unterstützen sollen, sehen wie folgt aus:

1. Variante 1: Hier steht dem Anwender lediglich ein auf Papier gemalter Grundriss des Gebäudes zur Verfügung. Dieser Plan darf nicht transportiert werden, er bleibt fest an einer Stelle und der Proband muss anhand dieses Plans so schnell wie möglich eine festgelegte Anzahl Räume finden.
2. Variante 2: Der Proband bekommt mithilfe der Hololens eine 3D Ansicht des Gebäudes visualisiert. Dieses Modell bleibt fix am Startpunkt. Der zu suchende Raum wird auf bestimmte Weise innerhalb dieses Modells markiert. Der Proband zieht die Brille aus, sobald er sich in der Lage fühlt den Raum zu finden, und macht sich auf den Weg.
3. Variante 3: Der Proband arbeitet durchgehend mit der Hololens, da dieses Mal das Modell um den Anwender herum aufgebaut wird. Er selbst bildet den Positionsindikator und bewegt sich durch das virtuelle Modell. Das Ziel wird ihm wie in Variante 2 markiert. Bei dieser Variante ist zu überlegen, ob man mit einem oder mehreren der Verzerrungseffekte arbeitet. Da es unter Umständen interessant sein kann, ob diese Effekte sich positiv oder negativ auf das Erreichen der Ziele auswirkt, kann man Variante 3 beliebig aufsplitten und Verzerrungsmethoden benutzen.
4. Variante 4: Auch hier trägt der Proband die Brille durchgehend, allerdings befindet sich das 3D Modell des Gebäudes nun jederzeit im Blickfeld des Probanden. Nun ist auch nicht mehr er der Positionsindikator innerhalb dieses Modells, sondern eine kleine Figur, die sich relativ zu seiner Position in der realen Welt mit bewegt. Das Ziel ist wie in Variante 2 markiert.

Als Maßstab, wie erfolgreich ein Proband die Aufgabe meistert, werden Zeit und Fehlerrate gemessen. Fehler können dann entstehen, wenn der Anwender angibt, dass er sich nach seiner eigenen Einschätzung eigentlich am Ziel befindet, obwohl dies nicht der Fall ist. Damit die Zeit, die der Proband für das Finden eines Raumes benötigt richtig verwertet werden kann, muss darauf geachtet werden, dass die Distanz zu den Räumen berücksichtigt wird. Somit sollten die gesuchten Räume immer von einem Modus zum nächsten rotieren, wenn der nächste Proband an der Reihe ist, und die durchschnittliche Distanz zu den Räumen innerhalb der verschiedenen Modi sollte in etwa die gleiche sein, oder zumindest anhand eines Multiplikators mit der Zeit relativiert werden. Außerdem ist auch das persönliche Befinden der Probanden von primären Interesse. Ist die dreidimensionale Ansicht des Modells hilfreich bei dem Erreichen der Ziele, oder stört sie eher? Fühlt sich die Nutzung von Augmented Reality natürlich an oder ist es eher auf Dauer belastend? Dies alles sind Einschätzungen, die sowohl für die Nutzung des Systems an sich, aber auch für den Einsatz von Augmented Reality Geräten von Bedeutung sind.

Der Grund, aus dem eine allumfassende Studie bei dem momentanen Stand des Systems noch nicht durchgeführt werden sollte, ist, dass das implementierte Gebäude um ein vielfaches zu klein ist, um eine sinnvolle Studie durchzuführen. Die Räume sind sinnvoll und intuitiv durchnummeriert und es gibt auch nur zwei Stockwerke. Somit wird es einem Probanden sehr leicht fallen, sich auch ohne Karte in dem Gebäude zurecht zu finden, selbst wenn er dort noch nie war. Aus diesem Grund sollte zumindest das 38er Gebäude der Universität Stuttgart nachgebaut und die Studie darin vollzogen werden. Neben den 3 Stockwerken die das Gebäude hat verfügt es auch über eine nicht sonderlich intuitive Nummerierung der Räume, und das Layout der oberen beiden Stockwerke ist

sehr komplex und verschachtelt. In solchen Gebäuden ist es anzunehmen dass eine dreidimensionale Kartenvariante den besten Vorzug mit sich bringt. Außerdem sollte der Proband natürlich auch nicht mit dem Gebäude vertraut sein, da dies die Ergebnisse verfälschen würde.

5 Zusammenfassung und Ausblick

Im Verlauf dieser Arbeit wurde ein dreidimensionales Modell des VISUS Gebäudes nachgebaut. Mithilfe des hier entwickelten Systems ist der Anwender in der Lage, das VISUS Gebäude zu durchschreiten, während ihm mithilfe der Microsoft Hololens das virtuelle 3D Modell angezeigt wird und ihn bei der Orientierung in dem Gebäude unterstützt. Basierend auf diesem Modell wurden mehrere Visualisierungseffekte entwickelt, welche den Betrachter dabei unterstützen sollen, für ihn relevanten Informationen zu erhalten. Darüber hinaus wurden die verschiedenen Effekte auf ihre Vor- und Nachteile untersucht. Neben den unterschiedlichen Visualisierungseffekte wurden außerdem drei verschiedene Minimaps entwickelt. In einer Variante stellt der Benutzer selbst den Zentrum der Map dar, während sich diese dynamisch mit bewegt, abhängig davon wo er sich innerhalb der realen Welt befindet. In einer weiteren Variante wird dem Anwender eine statische Minimap zur Verfügung gestellt, in der er sich frei bewegen und die er nach belieben untersuchen kann. In der letzten Variante wird dem Benutzer die Minimap im Sichtfeld angezeigt, während ein kleiner Positionsindikator vorgibt, wo er sich gerade in der realen Umgebung befindet.

Ausblick

Es bleibt herauszufinden, welcher der Effekte in Kombination mit den verschiedenen Minimaps am besten für die Unterstützung der Navigation verwendet werden kann. Hierfür wurde die Basis einer Studie entworfen, in der das anhand unterschiedlicher Testfälle herausgefunden werden soll. Um diese Tests jedoch so aussagekräftig wie möglich zu machen, wäre es zunächst von Vorteil, ein größeres Modell zu bauen. Dieses soll den Anwender vor größere Herausforderungen in Sachen Orientierung stellen, damit sich der Gebrauch einer Minimap überhaupt erst lohnt. Darüber hinaus stellt die Implementierung verschiedener Shading-Effekte eine sinnvolle Erweiterung des hier entwickelten Systems dar. Dabei gilt es beispielsweise herauszufinden, wie solche Effekte in Kombination mit den verschiedenen Verzerrungseffekten dafür genutzten werden können, den Nutzer bei der Orientierung zu unterstützen.

Literaturverzeichnis

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, B. MacIntyre. “Recent advances in augmented reality”. In: *IEEE computer graphics and applications* 21.6 (2001), S. 34–47 (zitiert auf S. 17, 18).

[AQNK18] E. Azimi, L. Qian, N. Navab, P. Kazanzides. “Alignment of the Virtual Scene to the 3D Display Space of a Mixed Reality Head-Mounted Display”. In: *arXiv preprint arXiv:1703.05834* (2018) (zitiert auf S. 28).

[BD06] T. Bailey, H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): Part II”. In: *IEEE robotics & automation magazine* 13.3 (2006), S. 108–117 (zitiert auf S. 20).

[Bos] M. Bostock. *Git-Hub des 2D Fisheye Algorithmus*. https://github.com/d3/d3-plugins/tree/master/fisheye. Accessed: 2020-04-21 (zitiert auf S. 34).

[CFA+11] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, M. Ivkovic. “Augmented reality technologies, systems and applications”. In: *Multimedia tools and applications* 51.1 (2011), S. 341–377 (zitiert auf S. 17).

[GVK16] G. Gerstweiler, E. Vonach, H. Kaufmann. “HyMoTrack: A mobile AR navigation system for complex indoor environments”. In: *Sensors* 16.1 (2016), S. 17 (zitiert auf S. 20, 21).

[HFT+99] T. Höllerer, S. Feiner, T. Terauchi, G. Rashid, D. Hallaway. “Exploring MARS: developing indoor and outdoor user interfaces to a mobile augmented reality system”. In: *Computers & Graphics* 23.6 (1999), S. 779–785 (zitiert auf S. 18, 19).

[Mica] Microsoft. *Bild der Microsoft Hololens 1 mit Beschriftung der relevanten Hardware*. https://www.microsoft.com/en-us/research/blog/microsoft-hololens-facilitates-computer-vision-research-by-providing-access-to-raw-image-sensor-streams-with-research-mode/. Accessed: 2020-06-03 (zitiert auf S. 26).

[Micb] Microsoft. *Depth-Stencil Testing Overview*. https://docs.microsoft.com/de-de/windows/win32/direct3d11/d3d10-graphics-programming-guide-output-merger-stage. Accessed: 2020-04-21 (zitiert auf S. 31).

[Micc] Microsoft. *Direct3D 11 Pipeline*. https://docs.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-graphics-pipeline. Accessed: 2020-04-21 (zitiert auf S. 29).

[Micd] Microsoft. *Hardware Informationen zur Microsoft Hololens 1*. https://docs.microsoft.com/de-de/hololens/hololens1-hardware. Accessed: 2020-04-03 (zitiert auf S. 25).

[Mice] Microsoft. *Mixed Reality Toolkit*. https://github.com/microsoft/MixedRealityToolkit-Unity. Accessed: 2020-03-30 (zitiert auf S. 27).

[Micf] Microsoft. *Technische Details der Microsoft Hololens 1*. https://docs.microsoft.com/de-de/hololens/hololens1-hardware. Accessed: 2020-06-03 (zitiert auf S. 26).

[NPF+03] W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, M. Reiner, J. Wieghardt, H. Horst, C. Lindinger et al. "Pervasive information acquisition for mobile AR-navigation systems". In: *null*. IEEE. 2003, S. 13 (zitiert auf S. 19, 20).

[RKI] RKI. *Steckbrief zur COVID19-Krankheit des RKI*. https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Steckbrief.html. Accessed: 2020-06-08 (zitiert auf S. 50).

[RMS+08] T. Reinhard, S. Meier, R. Stoiber, C. Cramer, M. Glinz. "Tool support for the navigation in graphical models". In: *Proceedings of the 30th international conference on Software engineering*. 2008, S. 823–826 (zitiert auf S. 22).

[SB94] M. Sarkar, M. H. Brown. "Graphical fisheye views". In: *Communications of the ACM* 37.12 (1994), S. 73–83 (zitiert auf S. 22, 23, 33, 35).

[Uni] Unity. *Homepage of Unity*. https://unity.com/. Accessed: 2020-03-30 (zitiert auf S. 26).

[Wik] Wikipedia. *Artikel über Tessellierung-Shader*. https://de.wikipedia.org/wiki/Tessellation-Shader. Accessed: 2020-03-30 (zitiert auf S. 30).

Alle URLs wurden zuletzt am 22. 06. 2020 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift