Fachhochschule Kiel / University of Applied Sciences

Interaktive Hologramme

Realisierbarkeit von interaktiven Echtzeitanwendungen
auf einer holographischen Pyramide

Bachelorthesis
im Rahmen des Studiengangs Multimedia Production,
Sommersemester 2016

Erstprüfer: Thomas Heuer M. A.
Zweitprüferin: Prof. Dr. Franziska Uhing

Abgabetermin: 17.06.2016

vorgelegt von:

Student:	Nils Gudat
Fachbereich:	Medien
Studiengang:	Multimedia Production
Semester:	8
Matrikelnummer:	921247
Geburtsdatum:	14.05.1986
Adresse:	Ernestinenstraße 9, 24143 Kiel
Email:	nils.gudat@student.fh-kiel.de

Kiel, den 14.06.2016

Gender-Erklärung

Zur besseren Lesbarkeit werden in dieser Bachelorthesis personenbezogene Bezeichnungen, die sich zugleich auf Frauen und Männer beziehen, generell nur in der im Deutschen üblichen männlichen Form angeführt, also z.B. „Betrachter“ statt „Betrachterin“.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AR	Augmented Reality
CPU	Central Processing Unit
CSS	Cascading Style Sheet
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
js	JavaScript
LED	Light-Emitting Diode
MR	Mixed Reality
PHP	Hypertext Preprocessor
UE4	Unreal Engine 4
Voxel	Volumetrischer Pixel
VR	Virtual Reality
WebGL	Web Graphics Library

1 Einleitung

In diesem Kapitel erfolgt zunächst die Darlegung von Motiven, welche die Relevanz der vorliegenden Ausarbeitung begründen. Anschließend wird eine nähere Erläuterung der Aufgabenstellung dieser Thesis sowie die damit einhergehende Zielsetzung angeführt. In dem letzten Abschnitt der Einleitung wird der Aufbau der Arbeit dargestellt.

1.1 Motivation

Seit der Wiederbelebung des 3D-Kinofilms in 2009 durch den Film AVATAR gewinnen dreidimensionale Anzeigetechniken zunehmend an Einfluss auf die Medienwelt und den Konsumenten (Wegener et al. 2012, S. 13). Das zeigen auch die derzeitigen Entwicklungen im Bereich Virtual Reality (VR). Hierbei konkurrieren mehrere Hersteller von VR-Brillen für verschiedene Plattformen um die Platzierung am Markt (Austinat et al. 2015, S. 42 - 48). Ebenso wird an der Entwicklung von Augmented Reality Lösungen und volumetrischen Anzeigetechniken gearbeitet (Microsoft 2016a, Voxon 2016). Es scheint also ein gewisser Bedarf an neuartigen Anzeigetechniken zu bestehen, welche den Konsumenten noch tiefer in das Geschehen, sei es ein Videospiel, eine Visualisierung oder ein Film, hineinziehen können. Bei all den Neuentwicklungen stellt sich die Frage, welche bereits bekannten Techniken zur Verfügung stehen, mit denen es möglich sein kann, einen gewissen Grad an Immersion auf Seiten des Rezipienten zu erzeugen. Hierbei ist es ebenso notwendig in Erfahrung zu bringen, wie ein Eintauchen in gezeigte Darstellungen umzusetzen ist und welchem Zweck eine solche Anzeigetechnik dienen kann.

1.2 Aufgabenstellung und Zielsetzung

Das in dieser Ausarbeitung näher untersuchte Anzeigemedium holographische Pyramide sowie der dem Medium zugrunde liegende optische Effekt wird bereits seit längerer Zeit dazu eingesetzt, um immersive Illusionen zu erzeugen. Ziel dieser Abhandlung ist es zu klären, in wie weit es möglich ist, ein solches Anzeigemedium mit interaktiven Echtzeitanwendungen zu bespielen. Hierzu zählt ebenso die Suche nach geeigneten Entwicklungsumgebungen sowie die testweise Realisierung von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide. Darüber hinaus sollen die Testergebnisse Aufschluss über mögliche Einsatzbereiche des Anzeigemediums, im Zusammenhang mit interaktiven Echtzeitanwendungen, geben. Um die erstellten

Testanwendungen auf ihre Funktionalität sowie Effektivität überprüfen zu können, ist der praktische Aufbau eines holographischen Pyramiden-Systems ebenso Teil der vorliegenden Abhandlung. Da es sich bei dem Anzeigemedium um ein recht komplexes System in Bezug auf das Verständnis der optischen Illusionen im dreidimensionalen Raum sowie dessen Funktionsweise handelt, sind neben den genannten Punkten ebenfalls Ausführungen außerhalb der reinen Anwendungsumsetzung Bestandteil der vorliegenden Ausarbeitungen.

1.3 Aufbau der Arbeit

Der Aufbau der dargelegten Arbeit mit dem Titel „Interaktive Hologramme: Realisierbarkeit von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide“ beginnt mit der Vermittlung verschiedener theoretischer Grundlagen in Kapitel 2. In diesem Abschnitt sind zunächst eine knappe Erläuterung der Bezeichnung „holographische Pyramide“ sowie verschiedene Begrifflichkeiten enthalten, welche für die vorliegende Abhandlung von Bedeutung sind. Da sich die Arbeit vermehrt mit der Wahrnehmung von Dreidimensionalität befasst, findet eine Erläuterung darüber, wie diese zustande kommt, ebenfalls innerhalb des zweiten Kapitels statt. Nachdem die verschiedenen Begriffe definiert wurden, erfolgt eine Beschreibung des Prinzips der Holographie sowie eine Darlegung, in welcher Form der Begriff „Holographie“ verwendet wird. Da sich der Versuchsaufbau mit einem bereits seit längerer Zeit bekannten optischen Prinzip beschäftigt, wird in Unterpunkt 2.4 dieser Arbeit näher auf den Ursprung des sogenannten „Pepper’s Ghost Effekts“ eingegangen. Kapitel 3 stellt den Hauptteil dieser Abhandlung dar. Hierbei wird die Realisierbarkeit von interaktiven Echtzeitanwendungen auf einer holographische Pyramide überprüft. Ebenfalls enthalten ist in diesem Kapitel die Beschreibung des Versuchsaufbaus sowie die programmiertechnische Durchführung mittels zweier Lösungswege und deren Gegenüberstellung. Darüber hinaus werden in diesem Abschnitt die Umsetzung von Spielen für das Pyramiden-System, die Implementierung einer Bewegungssteuerung sowie eines Head-Tracking-Systems beschrieben. Am Ende des dritten Kapitels findet die Darlegung von theoretischen Einsatzbereichen für das vorgestellte Pyramiden-System statt. An den Versuchsaufbau schließt sich im vierten Kapitel eine Beschreibung sowie die testweise Durchführung von Verbesserungen der Bilddarstellung auf der holographischen Pyramide an. In Kapitel 5 wird die theoretische Einordnung des Versuchsaufbaus behandelt. Hierin enthalten ist zunächst die Präsentation von verschiedenen Anzeigetechniken, mit welchen sich dreidimensionale Objekte räumlich darstellen lassen. Ebenso findet an dieser Stelle ein Versuch der Einordnung des Pyramiden-Systems zu

den beschriebenen Techniken statt. In diesem Abschnitt ist ebenfalls eine Darstellung des Realitäts-Virtualitäts-Kontinuums sowie die Zuordnung des Versuchsaufbaus innerhalb des Kontinuums enthalten. Das letzte Kapitel fasst die innerhalb der Ausarbeitung entstandenen Ergebnisse zusammen. Darüber hinaus gibt dieser Abschnitt einen Ausblick auf weiterführende, mögliche Untersuchungsfelder sowie eine Einschätzung bezüglich der Nutzbarkeit von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide.

2 Theoretische Grundlagen

Im folgenden Kapitel werden verschiedene Themen behandelt, welche für das bessere Verständnis der weiteren Ausführungen notwendig sind. Diese bilden die Grundlage für den in Kapitel 3 beschriebenen Versuchsaufbau.

2.1 Notwendige Begrifflichkeiten

Innerhalb dieses Unterpunkts werden unterschiedliche Begrifflichkeiten näher erläutert, welche sich in verschiedenen Teilen der Ausarbeitung wiederfinden lassen. Hierzu zählt unter anderem eine erste Beschreibung der im Versuchsaufbau verwendeten Anzeigetechnik. Ebenso findet eine Erläuterung der Bezeichnungen „Immersion“ und „Interaktion“ sowie die Beschreibung deren Zusammenwirkens, innerhalb dieses Abschnitts, statt. Darüber hinaus werden die Begriffe „Echtzeitanwendungen“, „Rendering“ und „Game-Engine“ definiert.

2.1.1 Bezeichnung „holographische Pyramide“

Die Bezeichnung „holographische Pyramide“ kann für das in dieser Ausarbeitung beschriebene Anzeigesystem angeführt werden. Dieser Ausdruck ist jedoch nicht als eine festgeschriebene Definition für eine solche Darstellungsform zu verstehen. Warum gerade dieser Begriff verwendet wurde, wird in Unterpunkt 2.3.3 der vorliegenden Abhandlung näher beschrieben. Da die folgenden Ausführungen jedoch ohne eine Kenntniss über die verwendete Anzeigetechnik nur erschwert nachvollzogen werden können, erfolgt an dieser Stelle eine grobe Beschreibung der Bezeichnung „holographische Pyramide“. Bei einer holographischen Pyramide handelt es sich um eine pyramidenförmige Konstruktion, welche aus einem reflektierenden Material besteht.

Abb. 1: *Technischer Grundaufbau einer holographischen Pyramide*

Innerhalb des Versuchsaufbaus kommt an dieser Stelle Plexiglas zum Einsatz. Neben einer spiegelnden Oberfläche ist gleichzeitig die Durchlässigkeit von Licht notwendig, um einen holographischen Effekt zu erzeugen. Oberhalb der Pyramide ist ein Display angebracht, welches mit Hilfe einer Stützkonstruktion mittig auf der Pyramide aufliegt. Auf dem Display können Darstellungen angezeigt werden, welche sich auf der darunter befindlichen Pyramide spiegeln. Durch eine korrekte Ausrichtung des auf dem Display angezeigten Inhalts können alle vier Seiten der Pyramide bespielt werden. Erfolgt die Ausgabe von Darstellungen in einer bestimmten Weise, ist es möglich, die Illusion von frei schwebenden Objekten innerhalb der Pyramide zu erzeugen (Abb. 1). An dem aufliegenden Display kann theoretisch jede Art von Abspielmedium angeschlossen werden. Dieses kann bspw. aus einem einfachen DVD-Player oder aber, wie im Fall des Versuchsaufbaus in dieser Abhandlung, aus einem Computer bestehen. Hat der Betrachter die Möglichkeit mit den Darstellungen auf der Pyramide zu interagieren, kann hierbei von interaktiven Hologrammen gesprochen werden.

2.1.2 Immersion und Interaktion

> *„Immersion is produced when works of art and image apparatus converge, or when the message and the medium form an almost inseparable unit, so that the medium becomes invisible" (Grau 2007, S. 148, zitiert nach Jockenhövel 2014, S. 167).*

Der Begriff „Immersion" wird in verschiedener Fachliteratur sehr unterschiedlich definiert. Unter dem Aspekt, dass es sich bei diesen Ausführungen um die Konstruktion von virtuellen Inhalten handelt, kann unter Immersion das Eintauchen in eine virtuelle Welt verstanden werden (Schröder 2013, S. 86). Je intensiver sich ein Betrachter in die Darstellung innerhalb eines Mediums involviert führt, desto höher der Grad an Immersion. Der Immersionsgrad hängt von vielen verschiedenen Faktoren ab. Neben der Fähigkeit des Betrachters, sich auf ein Medium oder die Darstellung einzulassen,

führen bspw. die Verwendung von Dreidimensionalität innerhalb der Darstellungen (Wegener et al. 2012, S. 154) oder die Möglichkeit der Interaktion mit einer gezeigten Szene zu einer erhöhten Immersion. Dies ist besonders unter dem Gesichtspunkt des Experience Marketing von großer Bedeutung. Bei dem Begriff „Experience Marketing“ wird davon ausgegangen, dass sich Marken und Produkte besser durchsetzen können, wenn diese in Verbindung mit einzigartigen Erfahrungen stehen (Overschmidt / Schröder 2013, S. 6). Diese einzigartige Erfahrung kann aus einem möglichst hohen Grad an Immersion bestehen. Im Fall des Versuchsaufbaus stehen die genannten Punkte Dreidimensionalität und Interaktion zur Erreichung einer erhöhten Immersion im Vordergrund.

> *„Als interaktiv wird ein System bezeichnet, bei dem der Benutzer durch Bedienhandlungen den Arbeitsablauf des Systems beeinflussen kann“ (Heinecke 2012, S. 3).*

Die Möglichkeit mit einem System oder einer Anwendung interagieren zu können, führt gleichzeitig zu einer erhöhten Konzentration auf die dargestellten Inhalte. Daraus folgt, dass eine Interaktion zu einer Verbesserung der Wahrnehmung des dargestellten Geschehens führt. Die Wirkungsweise bzw. das Behalten von Informationen oder Marken ist umso nachhaltiger, je mehr Sinne beansprucht werden. Die stärkste Form der Wahrnehmung ist der praktische Umgang mit einer dargestellten Information. Der Grad der Involvierung des Benutzers trägt maßgeblich zum Grad der Immersion bei (Ritter 2013, S. 102). Dementsprechend kann der interaktive Umgang mit Visualisierungen innerhalb des Pyramiden-Systems bspw. bei Produktvisualisierungen die Wahrnehmung einer präsentierten Marke oder eines Produkts steigern.

2.1.3 Echtzeitanwendungen

Unter dem Begriff „Echtzeit“ wird die Reaktion eines Systems auf anfallende Daten innerhalb einer vorgegebenen Zeit verstanden. Die Korrektheit der Verarbeitungsergebnisse ist bei Echtzeitsystemen dementsprechend nicht nur von der Funktionsweise des Programmablaufs sondern ebenfalls vom Zeitpunkt der Ausgabe abhängig. Echtzeitanwendungen können in harte und weiche Echtzeit unterteilt werden. Im Fall der durchzuführenden Versuche handelt es sich bei dem Begriff „Echtzeitanwendungen“ um Programme, welche unter der Definition von weicher Echtzeit erstellt werden. Das bedeutet, dass Abweichungen von vordefinierten Verarbeitungsintervallen tolerierbar sind und nicht zu einem Totalausfall der Programme führen (Scholz 2005, S. 40). Die

vorgegebene Reaktionszeit der Anwendungen richtet sich nach der Anzahl an Bilddarstellungen pro Sekunde, die erfolgen müssen, um dem Betrachter den Eindruck von flüssigen Bewegungen oder Animationen zu vermitteln. Der Begriff „Anwendungen“ ist bewusst sehr weiträumig gewählt, da innerhalb des Versuchsaufbaus keine eindeutige Zuordnung der zu erstellenden Programme getroffen werden kann. In erster Linie ist hierbei unter Echtzeitanwendung die Manipulation von dreidimensionalen Inhalten mittels verschiedener Eingabegeräte in Echtzeit zu verstehen.

2.1.4 Rendering

Der Begriff „Rendering“ spielt im Zusammenhang von Echtzeitanwendungen und insbesondere innerhalb des Versuchsaufbaus eine entscheidende Rolle, so dass hier zunächst eine knappe Begriffsdefinition erfolgt. Rendern oder auch Rendering ist in erster Linie die Umwandlung von mathematischen Formeln oder Berechnungen in für den Betrachter interpretierbare Bildinformationen (Zeman 2015, S. 111 - 114). Der Term „Rendern“ findet in vielen verschiedenen Medien Anwendung, bspw. bei der Berechnung von visuellen Effekten im Film aber auch bei der Ausgabe von 3D-Objekten. Hierbei muss in zwei Kategorien unterschieden werden. Die erste Kategorie ist das sogenannte „Software Rendering“. Hierbei wandelt die CPU des Computers Daten der 3D-Szene wie Objekte, Licht, Schatten und Materialien in als Datei speicherbare Bilder um. Da hierbei die Qualität und Präzision des Ergebnisses entscheidend ist, benötigt das Software Rendering sehr viel Zeit, wodurch keine Interaktionen während der Berechnung mit den Objekten der Szene möglich sind.
Im Gegensatz zum Software Rendering erlaubt das Hardware Rendering oder auch GPU Rendering die Ausgabe von Pixelbildern in Echtzeit unter gewissen Einschränkungen der Bildqualität. Hierbei erfolgen die Berechnung der Bilder durch die Grafikkarte des Computers anstatt des Prozessors. Da diese auf eine schnelle aber weniger exakte Bildberechnung ausgelegt ist, kann eine hohe Bildrate von im besten Fall 60 oder mehr Bildern pro Sekunde erreicht werden. Eine möglichst hohe Bildfrequenz ist notwendig, um Animationen oder Interaktionen mit den Objekten der Szene für das Auge flüssig darzustellen (Zeman 2015, S. 111 - 114). Genau wie beim Software Rendering ist die Geschwindigkeit der Ausgabe von verschiedenen Faktoren abhängig. Die Bildwiederholfrequenz kann bspw. durch das Weglassen von Schatten oder durch eine Reduzierung der Pixelauflösung gesteigert werden.

2.1.5 Game-Engine

In der Fachliteratur lassen sich viele verschiedene Erläuterungen von Game-Engines finden. Eine allgemein gültige Beschreibung für Game-Engines gibt es nicht, da jede Game-Engine über verschiedene Eigenschaften verfügt und für unterschiedliche Zwecke konzipiert ist (Zerbst / Düvel 2004, S. 5). Generell ist eine Game-Engine eine Art Baukasten aus verschiedenen Modulen zur Entwicklung von Spielen für verschiedene Plattformen, die das Zusammenspiel aus Programmierung, Design und Ausgabe von Spielen vereinfachen. Die Kernelemente der Engine hängen von dem Zweck oder der Art der Spiele ab, welche mit der jeweiligen Engine erstellt werden können. Die Realisierbarkeit von interaktiven Echtzeitanwendungen innerhalb des Versuchsaufbaus wurde unter anderem mit der Unreal Engine 4 (UE4) von Epic Games getestet. Da bei der Suche nach geeigneter Literatur keine Beschreibung des Engine-Aufbaus der aktuellen Version gefunden werden konnte, erfolgt an dieser Stelle eine knappe Erläuterung einiger Bestandteile (Module) des Engine-Vorgängers, welcher in seinen Hauptelementen identisch zur aktuellen Version ist. Die Unreal Engine 3 besteht im Kern aus einem Grafikmodul, einer Sound Engine, einem Modul für physikalische Berechnungen, einem Script-Interpretierer und einem Input Manager. Diese Elemente werden von der sogenannten „Core-Engine“ zusammengeführt. Durch den modularen Aufbau ist es möglich, einzelne Komponenten zu verändern und so die Game-Engine an die eigenen Bedürfnisse anzupassen. Die Grafik-Engine sorgt für die visuelle Darstellung bzw. das Rendern von Spielinhalten wie bspw. der Spielumgebung, Figuren, Animationen oder Materialien. Darüber hinaus bewirkt die Grafikeinheit eine Optimierung der dargestellten Inhalte, um die vom System zu leistenden Berechnungen zu minimieren. Die Sound Engine sorgt für die Bereitstellung der importierten Sounddateien zu vorbestimmten Ereignissen im Spielverlauf. Hierbei ist es möglich, verschiedene Sounds innerhalb eines sogenannten „SoundCues“ zu vermischen. Die im Kern integrierte Physik-Engine hat als Aufgabe, die physikalischen Eigenschaften der verwendeten 3D-Objekte zu berechnen. Hierzu zählen neben der Simulation von Schwerkraft sowie dem Gewicht von virtuellen Elementen auch Simulationen von bspw. sich korrekt bewegender Kleidung. Der Input Manager steuert die Eingaben des Spielers. Diese können neben Tastatur, Maus oder Gamepad ebenso durch neuere Formen der Eingabe wie bspw. die Bewegungserkennung des Spielers erfolgen. Der Unreal Script-Interpreter transformiert die einprogrammierten Spielabläufe in für die Engine lesbare Codezeilen. Im Fall der Unreal Engine 3 wird an dieser Stelle die von Epic Games eigens entwickelte Codesprache Unreal Script verwendet (Busby et al. 2009, S. 14 - 21). In der aktuellen Version der Engine kommt allerdings die Programmiersprache C++ zum Einsatz (Epic Games, Inc. 2016a). Wird ein Spiel gestartet, synchronisiert die Core-Engine die einzelnen Bauele-

mente miteinander. Erfolgt bspw. eine Eingabe des Spielers in Form eines Knopfdrucks, wird zunächst ein Signal an die Core-Engine gesendet. Diese überprüft anhand des programmierten Ablaufes, welche Handlungen oder Aktionen im Spiel bei Knopfdruck ausgeführt werden sollen. Anschließend werden die Veränderungen im Spiel an die Grafik-Engine weiter geleitet, welche die entsprechenden Aktionen, Animationen und physikalischen Ereignisse anzeigt (Busby et al. 2009, S. 14 - 21).

2.2 Dreidimensionale Wahrnehmung

Für das Verständnis der weiteren Ausführungen ist es notwendig, das menschliche Sehen und die darin enthaltene dreidimensionale Wahrnehmung, rudimentär zu erläutern. Diese Erörterung ist insbesondere für ein Grundverständnis der Holographie sowie der Wahrnehmung von Objekten innerhalb des Pyramiden-Aufbaus und vorgestellten Anzeigetechniken von Bedeutung.
Entscheidend für die visuelle Wahrnehmung ist das Vorhandensein einer Lichtquelle. Eine Lichtquelle ist ein Erzeuger von Energie, welcher elektromagnetische Wellen an die Umgebung abgibt (Ditzinger 2013, S. 7). Bei Lichtquellen wie der Sonne oder dem Licht von Glühlampen spricht man hierbei von spontaner Emission, da die Lichtwellen in verschiedene Richtungen emittiert werden (Eichler / Eichler 1995, S. 19). Je höher die von der Quelle ausgesandte Energie ist, desto schneller sind ebenfalls die Schwingungen der emittierten Welle, woraus eine verkürzte Wellenlänge (Abstand zweier Wellenberge) resultiert (Ditzinger 2013, S. 7). Diese Wellen mit unterschiedlicher Frequenz (Anzahl der Schwingungen pro Sekunde) werden allgemein als Lichtwellen oder in Bezug auf menschliche Empfindungen vereinfacht als Licht bezeichnet (Ostrowski 1987, S. 9). Licht besitzt im Normalfall eine hohe Bandbreite an verschiedenen Wellenlängen mit unterschiedlicher Streuung. Diese bestimmt in welcher Form das Licht vom menschlichen Auge erfasst wird. Kurze Wellenlängen werden bspw. als blaue Farbtöne erfasst, wohingegen lange Wellen eher im Bereich einer roten Farbwiedergabe liegen (Ditzinger 2013, S. 7). Das Sehen von Gegenständen resultiert aus verschiedenen Eigenschaften der Gegenstandoberflächen. Unterschiedliche Materialien reflektieren, transmittieren oder absorbieren das Licht auf unterschiedliche Weise. Ein rotes Tuch absorbiert bspw. grüne sowie blaue Lichtwellen und leitet lediglich Licht im langwelligen Bereich an das Auge weiter (Abb. 2) (Böhringer et al. 2014, S. 5 - 8). Durch unterschiedlich starke Transmission und Absorption des Lichts sowie Reflexionen der Umgebung erscheint der Gegenstand nicht als einfarbiges Objekt, sondern kann dem Betrachter ebenfalls Aufschluss über dessen Materialeigenschaften und Oberflächenbeschaffenheit geben. Die vom Gegenstand ausgesandten Lichtwellen (Objektwellen)

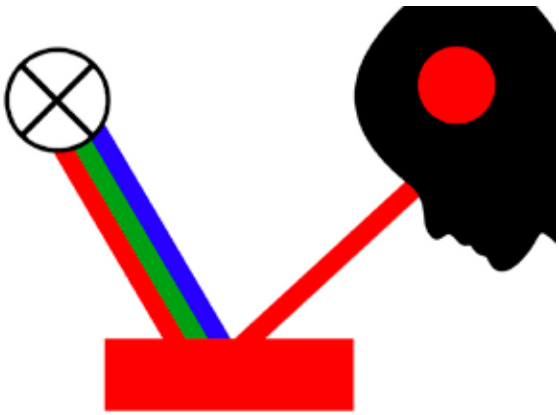

Abb. 2: *Farbsehen einer roten Fläche, blaue und grüne Lichtwellen werden absorbiert*

werden mittels der im Auge befindlichen Linse gebrochen und auf die Netzhaut projiziert (Ostrowski 1987, S. 9). Neben der Farb- und Helligkeitsinformation eines Gegenstands wird ebenfalls dessen Form durch das Licht wiedergegeben. Die Dreidimensionalität eines Objekts wird durch die sogenannte „Wellenfront“ beschrieben. Die vom Objekt ausgehenden Lichtwellen breiten sich gleichmäßig aus, wodurch die Struktur des Gegenstands über dessen sich ausbreitender Wellenfront weiter gegeben wird (Eichler / Eichler 1995, S. 174). Da die Netzhaut lediglich in der Lage ist, Helligkeits- und Farbunterschiede zu registrieren und die Wellenfront des Lichts nicht interpretieren kann, entsteht bei diesem Prozess eine zweidimensionale Abbildung des betrachteten Gegenstands. Die von den Lichtwellen ausgesandten räumlichen Informationen über den Gegenstand gehen bei der Interpretation durch das Auge verloren (Ostrowski 1987 S. 9). Dieser Informationsverlust kann allerdings durch eine zweite Abbildung ausgeglichen werden. Zu diesem Zweck ist das menschliche Augenpaar miteinander verbunden. Dadurch können beide Augen denselben Punkt im Sichtbereich fokussieren. Diese Verbindung ist eine Grundvoraussetzung einer räumlichen Wahrnehmung (Ditzinger 2013, S. 145). Bedingt durch den Abstand der Augen zueinander entstehen zwei unterschiedliche Abbildungen auf den Netzhäuten mit verschiedenen Blickwinkeln. Diese beiden Abbildungen können vom menschlichen Gehirn in Zusammenhang gebracht werden, wodurch ein räumlicher Gesamteindruck entsteht. Der Schnittwinkel zweier imaginärer Geraden, welche von den Augen in Richtung des betrachteten Objekts zeigen, wird als stereoskopische Parallaxe bezeichnet. Dieser Winkel bestimmt, wie stark oder schwach die räumliche Wirkung einer betrachteten Szene ist. Befindet sich ein betrachtetes Objekt nah am Augenpaar, ist der Schnittwinkel entsprechend groß und die beiden Bilder des Gegenstandes auf den Netzhäuten deutlich verschieden. Daraus resultiert ein hoher Grad an räumlicher Tiefenwirkung. Ist der Gegenstand jedoch weit entfernt, verkleinert sich die stereoskopische Parallaxe und die räumliche Empfindung nimmt ab (Ostrowski 1987, S. 9 - 10). Deutlich wird der Vorgang des stereoskopischen Sehens in Abbildung 3. Jedes Auge empfängt ein unterschiedliches Abbild des Gegenstandes ABCD auf der Netzhaut (Van Albada 1992, S. 7). Würde das Objekt nach oben verschoben, hätte dies eine Schmälerung der stereoskopischen Parallaxe und somit eine Verringerung der Tiefenwirkung zur Folge. Ein weiterer Indi-

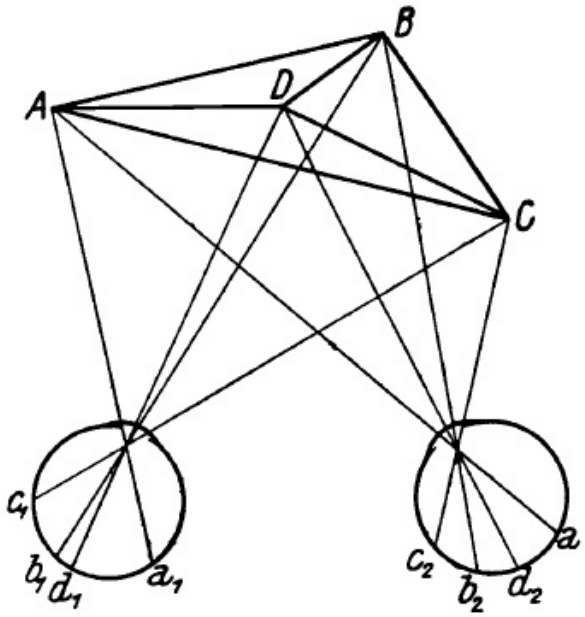

Abb. 3: *Verschiedenheit des rechten und des linken Netzhautbildes*

kator für Tiefenwirkung ist die sogenannte „Bewegungsparallaxe“. Dieser Begriff bezieht sich auf den Effekt, welcher bei der Betrachtung von unterschiedlich weit entfernten Objekten entsteht, sobald sich der Beobachter bewegt. Gegenstände, welche sich nahe des Auges befinden, scheinen sich schneller zu bewegen als ein weiter entfernter Hintergrund, sobald eine Bewegung des Kopfes ausgeführt wird. Anhand dieser Information kann auch ohne stereoskopisches Sehen eine Einordnung von Relationen der Tiefe von Objekten aufgestellt werden, wodurch ein dreidimensionaler Eindruck vermittelt wird (Jockenhövel 2014, S. 78).

2.3 Holographie

Der Begriff „Holographie“ oder „holographische Darstellung“ spielt eine entscheidende Rolle innerhalb der vorliegenden Abhandlung, daher ist es zunächst notwendig zu klären, was unter diesem Begriff zu verstehen ist. Aus diesem Grund werden im folgenden Kapitel zunächst Techniken beschrieben, mit welchen Bilder mit Tiefenwirkung erzeugt werden können, um anschließend eine Abgrenzung zur Holographie vorzunehmen. Darüber hinaus soll mittels dieses Kapitels ein Verständnis für den Begriff „Holographie“ im Allgemeinen vermittelt werden und aufzeigen, in welcher Weise das Wort „Holographie“ eingesetzt wird.

2.3.1 Bilddarstellung mit Tiefenwirkung

Es gibt viele verschiedene Möglichkeiten und Verfahren, räumlich erscheinende Bilder zu konstruieren. Eines dieser Verfahren ist die Anaglyphentechnik. Hierbei werden zwei Stereobilder (zwei verschiedene Aufnahmen mit durchschnittlichem Abstand des menschlichen Augenpaares) (Ditzinger 2013, S. 161) übereinander abgedruckt und verschieden eingefärbt. Bei der Betrachtung des Bildes durch einen entsprechenden Farbfilter wird die Abbildung in der Farbe des Filters entsprechend ausgelöscht, sodass

nur eine der Abbildungen erhalten bleibt. Werden nun beide Augen mit unterschiedlichen Farbfiltern abgedeckt, entsteht ein stereoskopischer Eindruck der Photographie (Ditzinger 2013 S. 183). Eine weitere Möglichkeit für die Erzeugung von Bildern mit Tiefenwirkung ist die Verwendung von Shutterbrillen. Bei diesem Verfahren sieht der Betrachter die Stereobilder schnell nacheinander, wodurch ein räumlicher Effekt erzielt wird (Preim / Dachselt 2015, S. 308). Da die räumliche Wirkung von Stereobildern auf einen vorbestimmten Blickwinkel beschränkt ist, kann nur von einem festen Punkt aus ein hoher Grad an dreidimensionaler Wirkung erzielt werden. Des Weiteren ist es ist dem Betrachter nicht möglich, durch Bewegungen des Kopfes, um einen Gegenstand herum zu schauen. Dementsprechend kann durch die reine Verwendung von Stereoskopischen Verfahren, kein parallaktischer Effekt erzeugt werden. Diese Beschränkungen können durch die Technik der Holographie verbessert werden (Ostrowski 1987, S. 11).

2.3.2 Das Prinzip der Holographie

Unter Holographie (griech., holo, Vorsilbe für ganz, unversehrt) (Meschede 2008, S. 212) wird in der Physik die Speicherung von Wellen aller Art verstanden. Diese können neben Lichtwellen auch aus Mikro- oder Schallwellen bestehen (Ostrowski 1987, S. 13). Ein weiterer Einsatzbereich ist die Speicherung von Daten, die in holographischer Form aufgezeichnet werden. Diese Technik erlaubt es, die Daten nicht nur auf der Oberfläche eines Trägers, sondern in seinem vollständigen Volumen zu speichern (Eichler / Eichler 1995, S. 101). Da sich die vorliegenden Ausführungen in erster Linie mit illusionistischen Effekten befassen, wird an dieser Stelle lediglich auf die grundlegenden Aspekte der visuellen Holographie eingegangen.
Das Grundprinzip zur Speicherung von Lichtwellen und die darin enthaltene vollständige Information über ein Objekt wurde 1948 von dem Physiker Dennis Gabor entdeckt. Gabor entwickelte ein Verfahren, mittels welchem Bildinformationen auf einer photographischen Platte gespeichert und anschließend wiedergegeben werden können. Neben der Intensität des vom Objekt reflektierten Lichts konnte durch sein Verfahren ebenfalls die Wellenfront gespeichert und somit ein aufgezeichnetes Objekt dreidimensional rekonstruiert werden (Lipson et al. 1997, S. 345 - 346). Für die Erzeugung eines Hologramms wird eine Lichtquelle benötigt, die kohärente Lichtwellen emittiert. Kohärentes Licht wird dadurch ausgezeichnet, dass es in gleicher Richtung strahlt und sich die Wellenzüge regelmäßig überlagern. In diesem Zusammenhang wird von „stimulierter Emission“ gesprochen (Eichler / Eichler 1995, S. 20, S. 26). Da Gabor zur Erstellung der Hologramme lediglich eine Quecksilberdampflampe zur Verfügung

stand, waren die ersten Abbildung von geringer optischer Qualität (Ostrowski 1987, S. 13). Auf Grund von besseren Kohärenz-Eigenschaften von Laserlicht gegenüber dem der Quecksilberdampflampe, war es erst mit der Erfindung des Lasers (Light amplification by stimulated emission of radiation (Böhringer et al. 2014, S. 97)) möglich, die dreidimensionalen Abbildungen wesentlich zu verbessern (Ostrowski 1987, S. 86). Um ein Hologramm mittels eines Lasers zu erzeugen, wird das vom Laser ausgehende Licht aufgeteilt (Abb. 4). Ein Teil des kohärenten Lichts wird auf das abzubildende Objekt gerichtet. Dieses reflektiert den einfallenden Laserstrahl und lenkt den selbigen in Form der sogenannten „Objektwelle“ auf eine photographische Platte. Der zweite Teil des aufgespaltenen kohärenten Lichtstrahls, die sogenannte „Referenzwelle“, wird direkt auf das Speichermedium gerichtet und nicht vom Objekt abgelenkt. Durch das Zusammenwirken von Referenzwelle und Objektwelle entsteht ein Interferenzmuster (Muster aus Überlagerungen von Lichtwellen), welches in Form einer Gitterstruktur auf der Photoplatte gespeichert wird (Meschede 2008, S. 214). Um die gespeicherten Bildinformationen für den Betrachter sichtbar zu machen, wird die Photoschicht mit der Referenzwelle beleuchtet. Dadurch wird die gespeicherte Objektwelle rekonstruiert und der zuvor gespeicherte Gegenstand erscheint als dreidimensionales Objekt auf der Photoplatte (Eichler / Eichler 1995, S. 176 - 177). Durch die Wiedergabe der Objektwelle kann der gespeicherte Gegenstand mit beiden Augen aus unterschiedlichen Perspektiven betrachtet werden, wodurch auch ohne zusätzliche Hardware ein vollständiger dreidimensionaler Eindruck entsteht. Neben den soeben beschriebenen Transmissions-Hologrammen gibt es noch weitere Hologramm-Arten, die sich durch Herstellung und Wiedergabe unterscheiden. Zusammenfassend kann gesagt werden, dass es sich bei dem durch Gabor geprägten Begriff „Holographie“ um die Speiche-

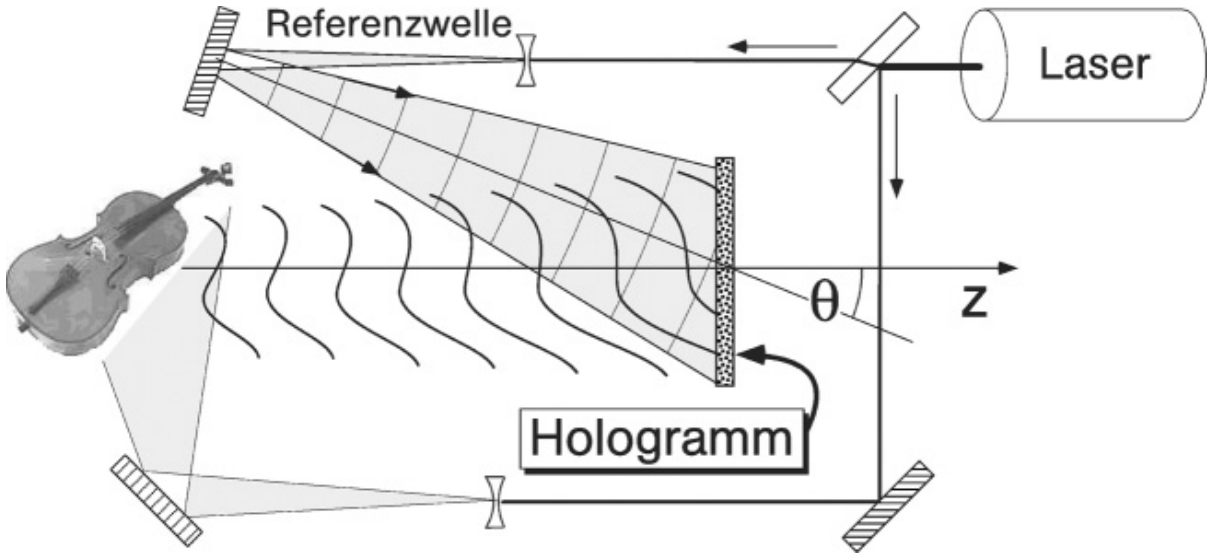

Abb. 4: *Aufnahme eines Hologramms mittels eines Lasers*

rung oder Wiedergabe von Licht-Wellenfronten eines Objekts oder Szene auf einem photographischen Trägermedium handelt.

2.3.3 Verwendung des Begriffs „Holographie“

Der Begriff „Holographie“ wird oftmals entgegen seiner physikalischen Definition (siehe 2.3.2) fälschlich verwendet, um optische Effekte und Illusionen zu beschreiben, bei denen Gegenstände dreidimensional und meist scheinbar im Raum schwebend dargestellt werden. Als ein sehr prägnantes Beispiel sei an dieser Stelle auf die Augmented Reality-Brille HoloLens von Microsoft hingewiesen. Auf der offiziellen Internetseite der HoloLens wird geschrieben:

> *„Microsoft HoloLens is the first fully untethered holographic computer running Windows 10. [...] Microsoft HoloLens allows you to place holograms in your physical environment and provides a new way to see your world“ (Microsoft 2016a).*

In diesem Artikel wird ausdrücklich auf die holographischen Darstellungs-Eigenschaften des Anzeigegeräts aufmerksam gemacht. Dementsprechend müssten die dreidimensionalen Projektionen, wie im Abschnitt 2.3.2 zum Prinzip der Holographie beschrieben, auf die Rekonstruktion von Wellenfronten der darzustellenden Objekte zurückzuführen sein. Wie aus einem weiteren Artikel über Sicherheitsvorkehrungen bei dem Umgang mit der HoloLens zu entnehmen ist, handelt es sich bei der Anzeigetechnik allerdings um ein stereoskopisches Anzeigeverfahren, bei welchem der Augenabstand über die korrekte Darstellung der dreidimensionalen Inhalte entscheidet.

> *„An interpupillary distance (an eye measurement of the distance between your two pupils) between 51 and 74 is needed to correctly and comfortably view Holograms with HoloLens. [...] Good binocular vision is required to view stereoscopic 3D content“ (Microsoft 2016b).*

Anhand der vorangegangenen Beschreibung des physikalischen Prinzips der Holographie sowie der dreidimensionalen Wahrnehmung handelt es sich bei den durch die Brille dargestellten dreidimensionalen Abbildungen lediglich um die Projektion von zwei verschiedenen zweidimensionalen Stereobildern, welche jeweils einem Auge zugespielt werden. Aus diesem Grund kann es sich, entgegen der Auslegung von Microsoft, nicht um ein echtes Hologramm handeln. Bimber und Raskar beschreiben in

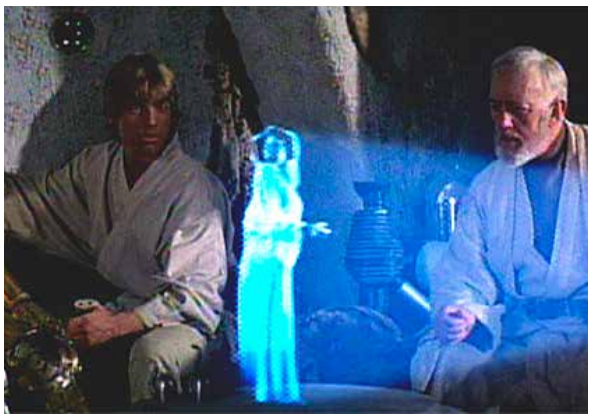

Abb. 5: *Holographie ähnliche Darstellung von Prinzessin Leia aus Star Wars*

ihrem Buch „Spatial Augmented Reality“ die räumliche Lichtdarstellung von Objekten am Beispiel von Prinzessin Leia aus Star Wars (Abb. 5) als Holographie ähnliche Projektion (Bimber / Raskar 2005, S. 2). Diese umschreibende Definition trifft den technischen Vorgang von Projektionen wie bei Microsoft’s HoloLens wesentlich besser. Wie in einem Artikel über die HoloLens in der „Zeit online“ zu entnehmen ist, wird der Begriff der „Holographie“ in vielen Medien falsch verwendet (Kühl 2015). Da sich in der Umgangssprache bei der Beschreibung mancher Illusionen durch Licht der Begriff „Hologramm“ oder „Holographie“ durchgesetzt hat, wird dieser ebenfalls für die in dem Versuchsaufbau verwendete Anzeigetechnik angeführt.

2.4 Geschichtlicher Hintergrund

Visuelle Effekte und Illusionen beschäftigen die Menschheit seit jeher. Bereits seit Erfindung der Laterna magica (eine Art altertümlicher Projektor), um 1659 (Liesegang 1992, S. 234), spielen Photo- und Filmprojektionen eine große Rolle dabei, die Aufmerksamkeit der Zuschauer zu erreichen und das Publikum in das Geschehen vor ihnen eintauchen zu lassen. Besonders die Illusion von Räumlichkeit und Dreidimensionalität trägt dazu bei, das Eintauchen der Beobachter in das Geschehen zu verbessern. So wurden bspw. Projektionen auf Rauch dazu verwendet, um geisterhafte Gestalten mitten im Raum zu erzeugen (Jockenhövel 2014, S. 167). Diese und weitere Varianten der Projektionsillusion sind heutzutage wie bereits erwähnt, fälschlicher Weise vielfach unter dem Begriff „Holographie“ bekannt. Zu sehen ist diese Definition bspw. auf der Internetseite der Firma Musion, welche holographische Projektionen für diverse Kunstprojekte sowie musikalische Aufführungen realisiert. Hierbei kommt zur Erzeugung von Illusionen frei schwebender Objekte oder Personen vermehrt der sogenannte „Pepper’s Ghost Effekt“ zum Einsatz (Musion 2016a).

2.4.1 Der Pepper's Ghost Effekt

Zurückzuführen ist der Pepper's Ghost Effekt auf eine Entdeckung des Universalgelehrten Giambattista della Porta im Jahr 1558. In seinem Buch „Magiae naturalis", welches als Standardwerk für magisches Wissen galt, beschreibt della Porta die Erzeugung von verschiedenen Illusionen mittels Spiegeln und Glasflächen (Clark 2007, S.98). Unter anderem erkannte er, dass Glas sowohl Reflexions- als auch Transmissions-Eigenschaften aufweist, durch welche sich mittels Licht, Illusionen erzeugen lassen (North / Rehak / Duffy 2015, S. 66). Auf der Basis dieser Erkenntnis entwickelten Henry Pepper und Henry Dircks eine Theaterillusion, die dem Publikum einen auf der Bühne schwebenden Geist zeigte. Bekannt wurde diese Art der Theaterprojektion unter dem Namen „Pepper's Ghost" (North / Rehak / Duffy 2015, S. 67). Um die Illusion des Geistes zu erzeugen, nutzen Pepper und Dircks eine Glasscheibe, welche in einem Winkel von 45 Grad zwischen dem Publikum und den Akteuren auf der Theaterbühne aufgebaut wurde. Die Glasscheibe wurde dabei in der Form angebracht, dass diese über die Bühne hinaus über den Orchestergraben des Theaters ragte. Unterhalb der Glasscheibe befand sich ein Schauspieler, der von einer Lichtquelle angestrahlt wurde. Mittels eines Spiegels ließ sich nun die Reflexion des als Geist verkleideten Schauspielers auf die Glasscheibe oberhalb des Orchestergrabens projizieren (Abb. 6). Da die Glasscheibe gleichzeitig die Sicht auf die Bühne zuließ und eine transparente Darstellung des Geistes unterhalb der Bühne ermöglichte, konnte ein Zusammenspiel aus Illusion und Akteuren realisiert werden (Beech 2012, S. 59).

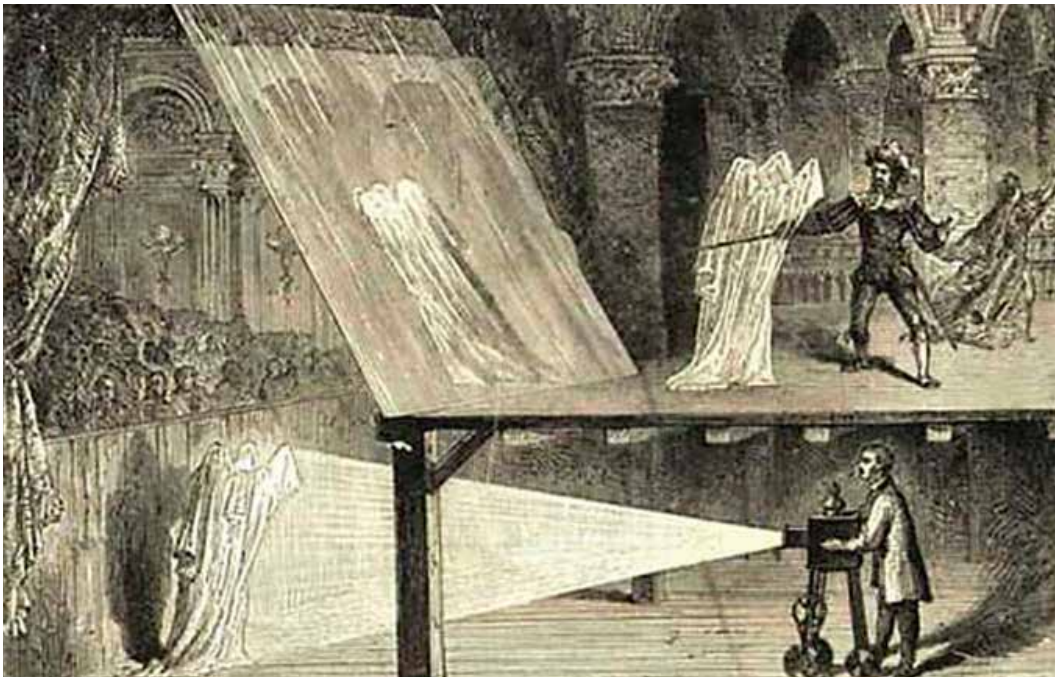

Abb. 6: *Beispiel einer Geisterprojektion mittels des Pepper's Ghost Effekts im Theater*

2.4.2 Verwendung des Pepper's Ghost Effekts

Heutzutage wird der Pepper's Ghost Effekt in vielen verschiedenen Bereichen eingesetzt. Unter anderem findet der Effekt Verwendung in der Veranstaltungstechnik. 2012 realisierte die Firma Musion eine holographische Inszenierung des verstorbenen Rappers Tupac. Hierbei kam ihre eigens entwickelte EyeLiner-Technologie zum Einsatz. Bei dieser wird eine holographische Abbildung mittels einer transparenten Spezialfolie sowie eines Projektors erzeugt. Wie bei dem Pepper's Ghost Effekt im Theater wird die Folie vor dem Publikum aufgebaut und über verschiedene Spiegel von der Projektion bespielt (Musion 2016b).
In Autos wird das Grundprinzip dieser Technik bspw. in Form von sogenannten „Head-Up-Displays" verwendet, um dem Fahrer Informationen über den Streckenverlauf einzublenden. Dies hat den Vorteil, dass die Sicht auf die Straße nicht vollständig verdeckt wird und somit weiterhin Hindernisse auf der Fahrbahn wahrgenommen werden können (Beech 2012, S. 59).

3 Versuchsaufbau

Der in dem vorangegangenen Abschnitt 2.4.1 beschriebene Pepper's Ghost Effekt kann unter anderem dazu eingesetzt werden, holographische Illusionen innerhalb einer gläsernen Pyramide zu erzeugen. Wie eine solche Illusion konstruiert werden kann, wird innerhalb des vorliegenden Kapitels genauer erläutert. Darüber hinaus erfolgt eine Beschreibung, mit welchen Techniken es möglich ist, mit den dargestellten Illusionen interagieren zu können und worin die Schwierigkeiten bei der Erzeugung von interaktiven Anwendungen für eine holographische Pyramide liegen. Wie in Abschnitt 2.1.1 beschrieben wurde, besteht ein holographisches Pyramiden-System zunächst aus drei Komponenten. Diese setzen sich im Fall des Versuchsaufbaus aus einer Plexiglaspyramide, einem Monitor sowie einem Computer zusammen, welcher für die Darstellung von interaktiven Echtzeitanwendungen verwendet wird. In einem ersten Abschnitt dieses Kapitels wird beispielhaft die Konstruktion der verwendeten Pyramide erläutert. Da die Berechnung der Pyramiden-Konstruktion von den optischen Eigenschaften des verwendeten Materials abhängig ist, erfolgt im Anschluss der Berechnung die Darlegung von physikalischen Phänomenen des zum Einsatz kommenden Plexiglases, welche zu dem gewünschten Effekt der holographischen Illusion führen. Für die rein experimentellen Versuchsdurchführungen ist die Art des Monitors sowie die technischen Eigenschaften des verwendeten Computers zweitrangig. Eine Umsetzung von interaktiven Echtzeitanwendungen kann theoretisch mit jeder Art Monitor

und Computer realisiert werden, wenn hierbei die Anforderungen, die von Seiten der verwendeten Software an die zur Verfügung stehende Hardware gerichtet sind, erfüllt werden. Aus diesem Grund entfällt eine Beschreibung der verwendeten Hardware auf Seiten des Computers. Da der genutzte Monitor allerdings mitverantwortlich für die Qualität der gezeigten Darstellungen ist, erfolgt eine theoretische Analyse von einsetzbaren Monitortechnologien außerhalb des Versuchsaufbaus in Kapitel 4 zur Verbesserung der Bilddarstellung. Nachdem die physikalischen Eigenschaften zur Erzeugung einer holographischen Darstellung sowie die Berechnung der Pyramide erläutert wurden, folgt eine Beschreibung von Anforderungen an die Umsetzung von Anwendungen für das verwendete Anzeigemedium. In einem weiteren Schritt wird die Wahl von verschiedenen Entwicklungsmöglichkeiten dargestellt, mit welchen die herausgestellten Anforderungen erfüllt werden können. Im Anschluss erfolgt die Beschreibung von testweisen Realisierungen sowie ein Vergleich der gewählten Entwicklungsumgebungen für das Pyramiden-System. Nachdem die Realisierbarkeit von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide überprüft wurde, erfolgen weitere Ausarbeitungen bezüglich der Erweiterung von programmiertechnischen Umsetzungen. Hierbei wird zunächst das Thema der Produktion von Spielen für das Pyramiden-System behandelt. Anschließend erfolgt die testweise Implementierung einer Bewegungssteuerung sowie die Verwendung eines Head-Tracking-Systems für die holographische Pyramide. Am Ende dieses Kapitels werden verschiedene, theoretische Einsatzbereiche für das im Versuchsaufbau verwendete Pyramiden-System in Bezug auf interaktive Echtzeitanwendungen dargelegt.

3.1 Aufbau und Berechnung der Plexiglaspyramide

Für den Versuchsaufbau wird eine Pyramide mit vier gleichen Flächen verwendet, theoretisch wären auch Aufbauten mit weniger oder mehr Seitenteilen denkbar. Allerdings reduziert sich die Größe der darstellbaren Projektion mit zunehmender Komplexität der Pyramide. Bei der Verwendung von vier Flächen reduziert sich die Größe des vom Monitor reflektierten Inhalts auf ein Viertel der zur Verfügung stehenden Gesamtfläche des Displays. Für die Kalkulation der einzelnen Glasflächen ist die Kenntnis über verschiedene mathematische Ausgangsgrößen erforderlich. Als erstes zu nennen ist der notwendige Winkel von 45 Grad der einzelnen Pyramidenseiten zum Abspielmedium. Die Bedeutung dieses Winkels wird im Abschnitt 3.2 genauer erläutert, spielt für die Berechnung der Pyramide an dieser Stelle allerdings eine untergeordnete Rolle. Als zweites sind die Seitenmaße des verwendeten Monitors von Bedeutung. Um eine möglichst große Anzeigefläche innerhalb der Pyramide zu erzielen, ist die Kenntnis

dieser Größen erforderlich. Handelsübliche Computermonitore, wie auch der im Versuchsaufbau verwendete Bildschirm, haben ein ungleichmäßiges Seitenverhältnis. Theoretisch ist es möglich, dieses Ungleichgewicht bei der Berechnung der Pyramide zu berücksichtigen. Durch das Einbeziehen verschieden langer Seiten des Monitors entsteht eine rechteckige Grundfläche sowie unterschiedlich große Flächen der einzelnen Seitenteile der Pyramide. Soll ein einzelnes 3D-Objekt innerhalb des Aufbaus angezeigt werden, ist die maximale Größe des Objekts entsprechend der kleinstmöglichen Seitenfläche der Pyramide. Wird dies nicht berücksichtigt, können Bildinhalte, welche auf den größeren Seitenflächen dargestellt werden, auf den kleineren Flächen abgeschnitten werden. Dadurch wird die Illusion eines frei schwebenden Objekts behindert. Eine nicht quadratische Grundfläche der Pyramide kann im Einzelfall sinnvoll sein, sobald auf den größeren Seitenflächen der Pyramide zusätzliche Informationen eingeblendet werden sollen. Da für den Versuchsaufbau keine speziellen Anforderungen an die zu realisierenden Anwendungen gestellt sind, wird ein quadratischer Pyramiden-Aufbau bevorzugt. Dieser bietet gegenüber einer rechteckigen Grundfläche gewisse Vorteile bei der Realisierung von interaktiven Anwendungen, da auf die Besonderheiten unterschiedlich großer Seitenflächen nicht eingegangen werden muss. Für das Anzeigesystem wird ein Monitor mit einem Seitenverhältnis von 4 zu 3 verwendet. Da die Grundfläche des Displays nicht quadratisch ist, diese Form allerdings Vorteile bei der späteren Umsetzung bietet, wird für die Berechnung der Pyramide das kürzeste Seitenmaß der Anzeigefläche des Displays verwendet. Bei dieser Art des Aufbaus wird zwar ein Teil der Bildschirmfläche für die Ausgabe auf der Pyramide nicht genutzt, dies ist allerdings zu Gunsten einer erleichterten Darstellung von Objekten zu vernachlässigen. Die Grundlänge (a) der einzelnen Glasflächen entspricht demzufolge der kürzesten Seitenlänge des verwendeten Displays (Abb. 7). Abbildung 7 dient zur Veranschaulichung der Berechnung der einzelnen Seitenteile. Anhand des gegebenen 45 Grad-Winkels (α) kann die Höhe der einzelnen Seitenflächen (h_a) der Pyramide berechnet werden. Hierzu wird die Grundlänge (a) halbiert und durch den Kosinus von α (45 Grad) geteilt. Anhand der Seitenlänge (a) sowie der Seitenhöhe (h_a) lassen sich somit vier gleiche dreieckige Flächen herstellen, die zu einer pyramidalen Anzeigefläche zusammen gefügt werden. Im Fall des Versuchsaufbaus wurde hierbei klares Plexiglas verwendet. Dieses bietet den Vorteil einer leichten Bearbeitung sowie einer hohen Flexibilität. Darüber hinaus ist es im Vergleich zu Glas sehr bruchsicher und ermöglicht die Beschich-

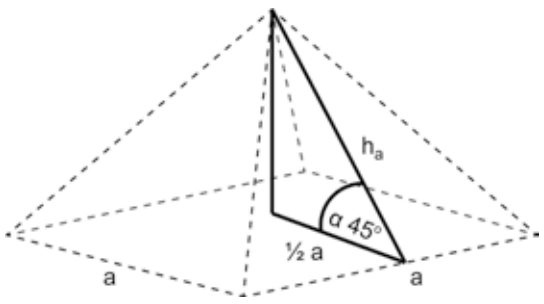

Abb. 7: *Berechnung der Flächenhöhe (h_a) der Seitenteile der Pyramide*

tung mit verschiedenen Materialien, wodurch die Reflexions- und Transmissions-Eigenschaften verändert werden können.

3.2 Reflexion und Transmission

Der zwingend erforderliche Winkel von 45 Grad der einzelnen Glasflächen zum Abspielmedium ist durch das Reflexionsgesetz begründet. Das Reflexionsgesetz besagt, dass der Winkel des einfallenden Lichts gleich seinem Ausfallswinkel ist (Bühler 2004, S. 6). Zur Veranschaulichung dieses Phänomens dient Abbildung 8. Das einfallende Licht, welches in Form von Darstellungen des Displays oberhalb der Pyramide abgegeben wird, trifft in einem Einfallswinkel (α) von 45 Grad auf die Plexiglasfläche. Anhand des Reflexionsgesetzes wird das einfallende Licht in gleicher Weise von der Glasfläche reflektiert. Dementsprechend wird das Licht in einem Ausfallswinkel (β) von ebenfalls 45 Grad von der Plexiglasfläche wiedergegeben. Auf diese Weise entsteht eine Drehung der abgebildeten Darstellungen auf dem Display um 90 Grad. Dadurch sieht der Beobachter die Inhalte des Displays bei frontaler Betrachtung des Pyramiden-Aufbaus in gerader Linie vor sich. Bei der Betrachtung eines Spiegels sieht der Beobachter die reflektierte Umgebung innerhalb der Spiegelfläche wie durch eine Art Fenster. Gleiches gilt für die Reflexion auf einer Plexiglasfläche. Somit entsteht ein Abbild des vom Display ausgesandten Lichts in Form von Objekten, innerhalb des Pyramiden-Aufbaus. Wie bereits im Abschnitt 2.4.1 über die Herkunft des Pepper's Ghost Effekts beschrieben wurde, verfügt Glas über die Eigenschaft, Licht zu reflektieren und gleichzeitig lichtdurchlässig zu sein. Die Stärke der Reflexion ist unter anderem vom Einfallswinkel des Lichts abhängig. Wird davon ausgegangen, dass ein Lichteinfall von 0 Grad einer senkrechten Beleuchtung des Plexiglases entspricht, wird der größte Teil des Lichts transmittiert, wohingegen nur ein kleiner Teil reflektiert wird. Bei zunehmendem Einfallswinkel steigt der Grad der Reflexion, wobei die Transmission des Lichts abnimmt (Arkema, Inc. 2000, S. 1). Arkema zu Folge beträgt der Grad der Reflexion bei einem angestrebten Lichteinfallswinkel von 45 Grad zum Plexiglas annähernd 10 Prozent, was eine relativ schwache Abbildung des gespiegelten Bildes darstellt. Der visuelle Eindruck der Reflexion ist ebenfalls von äußeren Einflüssen abhängig. Bei starker Umgebungsbeleuchtung wird diese

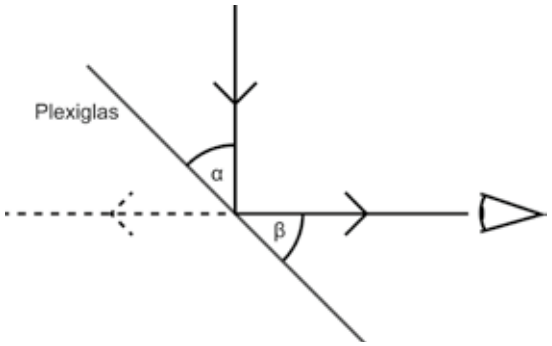

Abb. 8: *Reflexionsgesetz und Darstellung auf einer Plexiglasfläche*

ebenso auf der Glasfläche reflektiert. Dadurch wird das vom Display gespiegelte Licht reduziert bzw. überlagert. Hinzu kommt, dass Umgebungsobjekte hinter der Pyramide bei Tageslicht deutlicher zu sehen sind als bei Dunkelheit. Dadurch wird die optische Wahrnehmung eines gespiegelten Objekts innerhalb der Pyramide abgeschwächt. Ein weiteres Problem bei der Reflexion auf einer durchsichtigen Glasfläche ist die Verdoppelung des gespiegelten Bildes auf Grund der Glasdicke (Abb. 9). Bei der Spiegelung von Objekten auf der Pyramide entstehen zwei Reflexionsbilder. Die erste Spiegelung entsteht direkt an der Oberseite des Glases. Eine zweite Spiegelung wird an der dem Betrachter zugewandten Unterseite der Glasfläche erzeugt. Das im Versuchsaufbau verwendete Plexiglas hat eine Dicke von 2 Millimetern. Auf Grund der Doppelreflexion entstehen hierbei zwei zueinander versetzte Bilder mit einem Abstand von ca. 2,8 Millimetern. Die Doppelung des reflektierten Objekts führt zu einer leicht verschwommenen Ansicht des gespiegelten Inhalts, dadurch wird der optische Eindruck von feineren Darstellungen wie bspw. Schriften verschlechtert. Durch eine Reduzierung der Glasdicke kann diese Verdoppelung zwar verringert, aber nicht vollständig aufgehoben werden, da die Dicke des Glases für die Stabilität der Plexiglaspyramide mitverantwortlich ist. Neben der Bildreflexion ist die Transmissions-Fähigkeit von Plexiglas ein entscheidender Faktor für die Erzeugung eines holographischen Effekts. Die Lichtdurchlässigkeit ist bei der Erzeugung der Illusion in Zusammenhang mit den Reflexions-Eigenschaften zu betrachten. Die Qualität des reflektierten Bildes ist neben dem Lichteinfallswinkel von der Helligkeit des ausgesandten Lichts des Bildschirms abhängig. Mit zunehmender Helligkeit erhöht sich die Intensität der Reflexion, was wiederum eine Abnahme der Transmission des Glases zur Folge hat. Im Umkehrschluss führt eine fehlende Beleuchtung zu einer Reduzierung der Spiegelung, woraufhin die Oberfläche zunehmend lichtdurchlässiger wird. Daraus folgt, dass vom Monitor als schwarz ausgegebene Pixel keine Reflexion auf der Pyramide verursachen und dementsprechend an diesen Punkten ein hoher Grad an Transmission erreicht wird. Um anhand dieser Erkenntnis die Vortäuschung eines frei schwebenden Objekts innerhalb der Pyramide zu erzeugen, muss das freizustellende Objekt von schwarzem Bildinhalt (Pixeln) vollständig umgeben sein. Da die umliegenden, schwarzen Pixel keine oder eine vergleichsweise geringe Reflexion erzeugen, können sowohl die Pyramide selbst als auch Objekte hinter der Pyramide weiterhin betrachtet werden. Dadurch wird dem Betrachter der

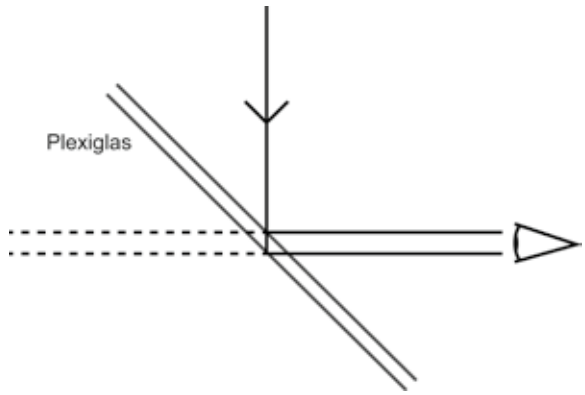

Abb. 9: *Doppelreflexion auf Grund der Dicke des verwendeten Glases*

Eindruck eines Hologramms vermittelt. Der zur Holographie-Illusion führende Effekt stellt zugleich ein großes Problem bei der Erzeugung realistischer Bilder dar. Da dunkle Bildinhalte zunehmend transparent dargestellt werden, ist es schwierig, dunkle Farben oder Schatten auf der Pyramide auszugeben.

3.3 Anforderungen an Umsetzung und Technik

Bei der Realisierung von interaktiven Echtzeitanwendungen werden an das System bzw. die Umsetzung gewisse Anforderungen gestellt. Im Fall eines pyramidenförmigen Anzeigesystems werden neben den mit der Echtzeit einhergehenden Bedingungen sehr spezielle Ansprüche an die Anwendungen gestellt. Um die besonderen Anforderungen besser beschreiben zu können, dient an dieser Stelle die beispielhafte Funktionsweise von dreidimensionalen Computerspielen als Erklärungshilfe. Ein Computerspiel ist aus verschiedenen Hauptkomponenten zusammengesetzt. Diese Komponenten können unter anderem aus einem Spieler (Benutzer), der Spielumgebung, bestimmten Spielregeln und einer Kamera, welche den darzustellenden Inhalt auf dem Monitor ausgibt, bestehen. Durch verschiedene Eingabegeräte kann ein einzelner Anwender die eigene Sicht auf die Spielumgebung oder Objekte verändern. Er hat die Möglichkeit, die Perspektive der Kamera um ein Objekt zu drehen und es von allen Seiten zu betrachten. Hierbei wird die gesamte Szene für jedes Bild pro Sekunde aus der Kameraperspektive des Spielers gerendert. Neben der Bewegung der Kamera hat der Spieler bspw. die Option, mit den einzelnen Elementen innerhalb des Spiels zu interagieren. Er kann z.B. Gegenstände in der Spielwelt aufheben und an anderer Stelle platzieren.
Wie bereits beschrieben wird bei einem PC-Spiel die Szene aus der Sicht eines einzelnen Benutzers wiedergegeben. Im Fall des Pyramiden-Systems müssen allerdings vier Ansichten der Szene dargestellt werden, welche jeweils im Abstand von 90 Grad um den Fokuspunkt innerhalb der Szene gedreht sind. Dadurch wird zwar keine echte dreidimensionale Ansicht möglich, aber die dargestellten Objekte können von vier verschiedenen Perspektiven gleichzeitig betrachtet werden. Steht der Betrachter bspw. vor der Pyramide, wird eine frontale Betrachtung der Szene deutlich. Bewegt sich der Betrachter um 180 Grad um die Pyramide, ist die Rückseite der dargestellten Szene zu sehen. Somit ist der Betrachter in der Lage, durch die rein physikalische Ausrichtung seines Körpers vier verschiedene Ansichten des Inhalts innerhalb der Pyramide zu beobachten. Im Gegensatz zu einer normalen Displayanzeige eines Spiels wäre hierfür eine Drehung der Kamera notwendig. Um vier unterschiedliche Perspektiven realisieren zu können, müssen gegenüber gewöhnlichen Computerspielen nicht nur

eine, sondern vier Kameras implementiert werden. Dementsprechend wird die darzustellende Szene aus den unterschiedlichen Winkeln mehrfach berechnet. Zwar bleibt die Auflösung der vier gerenderten Ansichten identisch zu der Auflösung einer einzelnen Perspektive, allerdings müssen die Szenen-Inhalte aus verschiedenen Winkeln berechnet werden. Dies gilt ebenfalls für das Rendern von Schatten, Licht, der dreidimensionalen Information der Objekte und die Veränderungen der Szene durch Eingaben des Benutzers. Demzufolge muss von dem Computer bei der Berechnung mehr Leistung bereit gestellt werden als bei einer einzelnen Ansicht.
Wird im Beispiel des Computerspiels davon ausgegangen, dass der Spieler eine Drehung der Kamera vollführt, wird auf der Pyramide nicht die Perspektive aus der Sicht des einzelnen Spielers verändert, sondern die Szene innerhalb der Pyramide neu positioniert. Dieses Merkmal führt dazu, dass bei der Implementierung von Benutzereingaben sehr stark darauf geachtet werden muss, in welcher Form sich eine Manipulation der Szene auf die Darstellung innerhalb der Pyramide auswirkt. Bspw. werden bei einer Kamerafahrt in Richtung eines Objekts ab einem bestimmten Ausschnitt Teile des Objekts durch den Monitor beschnitten. Im Fall des Pyramiden-Systems führt die Beschneidung der Szene dazu, dass die Illusion eines Hologramms verloren geht, da die Kanten der einzelnen Ansichten zueinander sowie die des Displays nun innerhalb der Glaspyramide zu sehen sind (Abb. 10). Ein weiteres Problem stellen Benutzereingaben dar, welche die Objekte innerhalb der Szene betreffen. Soll bspw. ein Objekt innerhalb der Pyramide von rechts nach links befördert werden, wird die Bewegung des Gegenstands von der gegenüberliegenden Seite der Pyramide als Bewegung von links nach rechts gesehen. Dadurch ist es notwendig bei dem Erstellen von Programmabläufen, die den Benutzer mit einbeziehen, darauf zu achten, dass ein intuitiver Umgang mit der Szene gegeben ist. Neben den bereits genannten Schwierigkeiten müssen die vom Monitor darzustellenden Inhalte auf der Anzeigefläche des Monitors jeweils um 90 Grad gedreht werden. Aus der normalen Ansicht ohne eine Pyramide werden Objekte demnach um den Mittelpunkt des Bildschirms rotiert, sodass ein aufrecht stehender Gegenstand bei einer Drehung von 180 Grad auf dem Kopf erscheint. Deutlich wird dies in Abbildung 11a. Darüber hinaus müssen, bedingt durch die Reflexion auf den einzelnen Plexiglasflächen, die einzelnen Viewports (Darstellungsfelder) spiegelverkehrt ausgegeben werden, damit beim Betrachter

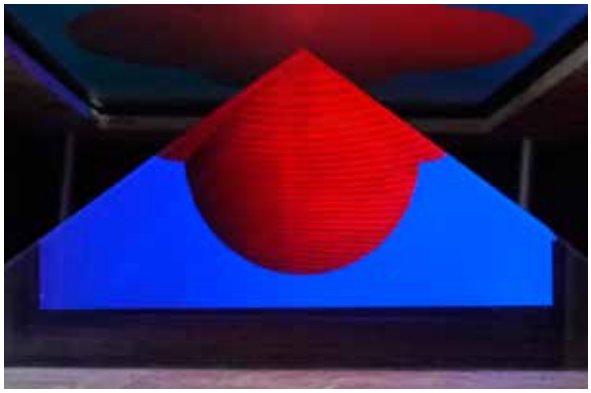

Abb. 10: *Beschnittene Objekte auf Grund einer zu großen Darstellung innerhalb der Pyramide*

ein korrektes Bild der Szene entsteht (Abb. 11b). Durch die Spiegelung der Darstellungen entsteht eine Umkehr der Räumlichkeit der angezeigten Objekte. In Abbildung 11a ist eine beispielhafte Szenen-Ausgabe auf dem Monitor zu sehen, bei welcher noch keine Korrektur der Spiegelung stattgefunden hat. In Abbildung 11b wurden die einzelnen Viewports gespiegelt. Wo nun in Abbildung 11a der rote Kreis und der grüne Halbkreis zunächst nebeneinander lagen, stehen nach der Korrektur der grüne Kreis sowie der rote Halbkreis zueinander. Demzufolge führt die Spiegelung der einzelnen Viewports zu einer fehlerhaften Ausgabe auf der Pyramide. In Abbildung 11c wurde der gesamte Aufbau an der Y-Achse gespiegelt. Demnach befinden sich nun der rote Kreis sowie der grüne Halbkreis weiterhin nebeneinander, wobei eine Spiegelung des Inhalts ebenso bestehen bleibt. Deutlicher wird dieses Prinzip bei der Umsetzung im dreidimensionalen Raum in Kapitel 3.6.2.

Die größte Herausforderung bei der Erstellung von interaktiven Anwendungen für eine holographische Pyramide ist die Implementierung von vier Kameras, welche eine dreieckige Ausgabe ermöglichen. In der realen wie auch virtuellen Welt erzeugen Kameras ein rechteckiges oder quadratisches Abbild einer Szene. Es ist zwar möglich, mit vier quadratischen Anzeigen einen holographischen Effekt zu erzeugen, allerdings gehen dabei 50 Prozent der maximal erreichbaren Darstellungsgröße verloren. Eine Veranschaulichung dieses Problems liefert Abbildung 12. Abbildung 12a zeigt die Verteilung von vier quadratischen Viewports, welche die Anzeige von vier verschiedenen Ansichten darstellen sollen. In Abbildung 12b wird die anhand der Monitorhöhe berechnete Pyramide über den Display gelegt (in blau gekennzeichnet). Durch die Zeichnung kann verdeutlicht werden, dass die Anzeige in dieser Form mit einer rechteckigen Aufteilung der Viewports nicht möglich ist. Auf diese Weise würden sich jeweils zwei Viewports eine Pyramidenfläche teilen, wodurch keine korrekte Ausgabe möglich ist. Um jedoch eine dreieckige Ausgabe zu erzeugen, kann die Pyramide um die Hälfte verkleinert und um 45 Grad gedreht werden (Abb. 12c). Dabei gehen allerdings, wie bereits erwähnt, 50 Prozent der möglichen darstellbaren Projektion verloren, da die quadratischen Viewports anhand der Drehung in jeweils zwei Dreiecke aufgeteilt wer-

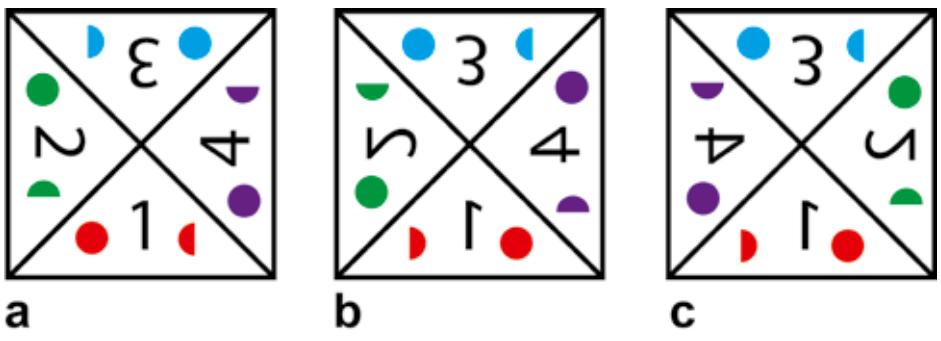

Abb. 11: *Rotation und Spiegelung der einzelnen Kameraperspektiven*

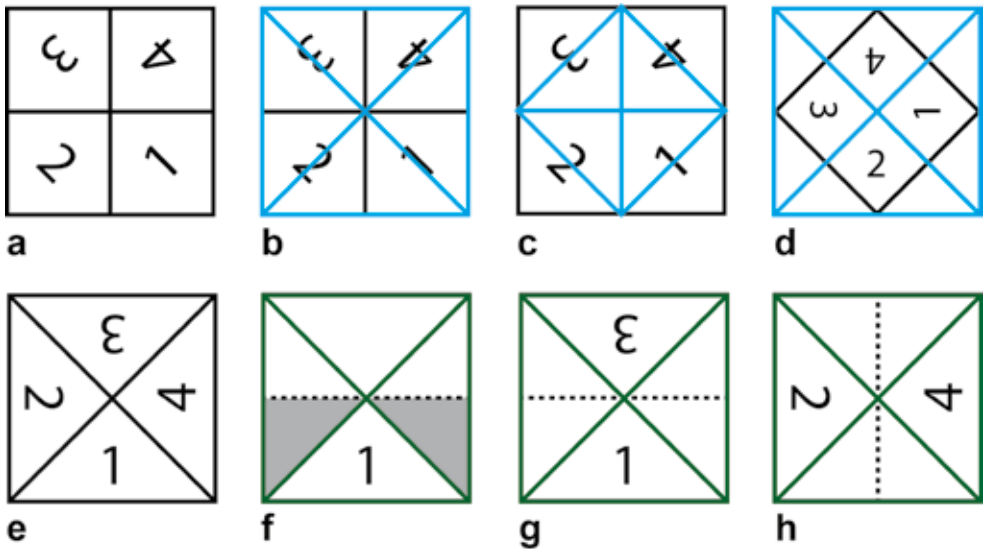

Abb. 12: *Vergleich von rechteckigen und dreieckigen Viewports auf dem Display innerhalb des Versuchsaufbaus*

den. Um die vollständige Anzeigefläche auf diese Art zu nutzen, ist eine Verdoppelung der Größe der berechneten Pyramide notwendig. Allerdings kann selbst in diesem Fall nur die Hälfte der Größe der Flächen bespielt werden (Abb. 12d). Um eine optimale Ausnutzung der Pyramidenflächen zu erzielen, müssen die Anzeigen auf dem Display selbst in dreieckige Viewports unterteilt werden (Abb. 12e). Zu erreichen ist dies nur durch eine Maskierung der Kameras bzw. Viewports. Dementsprechend wird das Format der Viewports auf die volle Breite des Bildschirms und dessen halbe Höhe eingestellt. Durch eine Beschneidung der Ecken des Formats kann eine dreieckige Anzeige erzeugt werden (Abb. 12f). Da vier Kameraperspektiven benötigt werden, müssen sich die Viewports bei dieser Art der Erstellung überlappen (Abb. 12g und 12h) Aus diesem Grund darf die Beschneidung der Bildflächen nicht einfach in schwarz erfolgen, sondern muss eine echte Transparenz aufweisen, da anderenfalls nur zwei Kameraanzeigen möglich wären.

3.4 Realisierung von interaktiven Echtzeitanwendungen

In den folgenden Abschnitten wird die programmiertechnische Umsetzung von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide dargelegt. Hierbei ist zunächst die Auswahl von geeigneten Entwicklungsumgebungen erforderlich, welche die im vorherigen Unterpunkt beschriebenen Anforderungen an die Darstellung auf einer Plexiglaspyramide erfüllen können. Nach der Auswahl von zwei möglichen Entwicklungsumgebungen, erfolgt deren Beschreibung sowie die praktische Realisierung

einer Testszenen-Ausgabe für das Pyramiden-System mit den entsprechenden Entwicklungsumgebungen. Jeweils anschließend findet eine Analyse von verschiedenen Vor- sowie Nachteilen der vorgestellten Technik statt. Nach der testweisen Realisierung von interaktiven Anwendungen für das System erfolgt eine Gegenüberstellung der unterschiedlichen Entwicklungsmethoden.

3.5 Auswahl der Entwicklungsumgebungen

Wie bereits in Abschnitt 3.3 beschrieben wurde, stellt die Erzeugung von vier dreieckigen Viewports und die damit einhergehende optimale Nutzung der Gesamtfläche des Displays ein großes Problem bei der Umsetzung von Anwendungen auf einer holographischen Pyramide dar. Bei der Erzeugung von Videos für das Pyramiden-System werden zunächst vier einzelne Perspektiven einer Szene gerendert. Anschließend können die einzelnen Filme beschnitten und zu einem abspielbaren Video zusammen gefügt werden. Dies muss bei der Umsetzung von interaktiven Echtzeitanwendungen für den Versuchsaufbau simultan geschehen, da anderenfalls keine Echtzeitinteraktionen möglich wären. Auf Grund von erlernten Kenntnissen im Studienverlauf war bekannt, dass es bei der Programmierung von Webseiten mit Hilfe von CSS3 möglich ist, sogenannte „Div-Container“ zu beschneiden. Da bereits Erfahrungen im Umgang mit HTML5, CSS3 sowie der Darstellung von 3D-Inhalten im Web vorhanden waren, wurde ein erster Testdurchlauf mittels HTML5, CSS3 und WebGL durchgeführt. Diese Methode wies jedoch verschiedene Vor- sowie Nachteile auf, sodass nach einigen testweisen Umsetzungen nach einem alternativen Lösungsweg gesucht wurde. Im Verlauf des Studiums wurden neben der Verwendung von WebGL und HTML5 ebenfalls Projekte mit der Game-Engine Unreal Engine 4 durchgeführt. Somit lagen bereits Grundkenntnisse im Umgang mit dieser Entwicklungsumgebung vor, sodass diese Technik im Verlauf der Versuchsdurchführung ebenfalls genutzt werden konnte, um Echtzeitanwendungen für den Versuchsaufbau zu realisieren.

3.6 WebGL, CSS3 und HTML5 als Entwicklungsumgebung

WebGL ist eine auf der Programmiersprache JavaScript basierende Programmierschnittstelle zur Anbindung an den OpenGL ES 2.0 Standard (Arora 2014, S. 8). Open GL ES 2.0 ist eine freie Grafikbibliothek, mit welcher sowohl 2D- als auch 3D-Grafiken auf verschiedenen Systemen wie bspw. Smartphones oder Spielekonsolen ausgegeben werden können (Khronos Group 2016). Durch die Anbindung an diese

Grafikbibliothek ist es mittels WebGL und HTML5 möglich, dreidimensionale Grafiken auf aktuellen Webbrowsern auszugeben. Hierbei ist WebGL allerdings nicht als eine Art Game-Engine zu verstehen. WebGL sorgt lediglich für das Rendern von 2D- und 3D-Objekten durch die Anbindung an den OpenGL Standard. Reines WebGL besteht aus nur wenigen Kommandos, mit welchen sich dreidimensionale Grundkörper wie Linien oder Kreise erstellen lassen. Um bspw. eine Kamera oder Kollisionen in der Szene zu implementieren, müssen eigene Klassen geschrieben werden, welche die notwendigen Informationen über die Programm-Erweiterungen enthalten (Arora 2014, S. 8). Um die soeben beschriebenen Erweiterungen nicht selbst programmieren zu müssen, können sogenannte „Frameworks“ (eine Art Baukasten) verwendet werden. Innerhalb dieser Frameworks sind einzelne Programmierbauteile wie bspw. Licht, Schatten, Kameras oder Materialien zusammengefasst. Die externen Bibliotheken enthalten alle notwendigen Beschreibungen der Grundbausteine und ermöglichen, innerhalb des eigenen Programmiergerüsts eine wesentlich einfachere Einbindung von speziellen Erweiterungen wie bspw. Licht, Schatten, Partikel oder Kameras.
Das für den Versuchsaufbau verwendete, kostenlose Framework mit dem Namen „Babylon.js“ wurde unter anderem von David Catuhe und David Rousset entwickelt und verfügt über eine große Bandbreite an fertigen Bauelementen (Babylon.js 2016a). Für die Einbindung von 3D-Objekten verwendet Babylon.js ein eigenes auf JSON basierendes Dateiformat. Um solche Dateien zu erzeugen, stellen die Entwickler ein eigenes Plugin (Erweiterung) für das 3D-Modellierungsprogramm 3ds Max bereit, mit welchem sowohl einzelne Objekte als auch komplette Szenerien mit darin enthaltenen Lichtern und Kameras exportiert werden können (Babylon.js 2016a). Für ein besseres Verständnis der folgenden Ausführungen erfolgt an dieser Stelle eine knappe Erklärung der Bildausgabe mittels des Babylon.js-Frameworks.
Die in den folgenden Abschnitten vorgenommenen CSS3- und HTML5-Anweisungen lassen sich mittels der Online-Dokumentation SELFHTML-Wiki nachvollziehen (SELFHTML e.V. 2016). Alle verwendeten Operationen, welche auf reinem JavaScript basieren, waren bereits im Vorwege durch das Studium bekannt. Die in den weiteren Abschnitten verwendeten Bausteine aus dem Babylon.js-Framework sind in der Online-Dokumentation des Frameworks aufgeführt und können an dieser Stelle eingesehen werden (Babylon.js 2016b). Für die Ausgabe von 3D-Inhalten auf einem Webbrowser mittels Babylon.js werden generell vier verschiedene Komponenten benötigt. Diese bestehen aus einer 3D-Szene, einer Kamera, einer im Framework enthaltenen Render-Engine sowie einem HTML5-Canvas-Element. Zunächst wird einem Canvas-Element, innerhalb welchem die Szene auf dem Browser ausgegeben werden soll, eine Render-Engine zugewiesen. Das Canvas-Element fungiert hierbei als eine Art Container innerhalb welchem 2D- und 3D-Grafiken gezeichnet werden können.

Dieser Container kann wie andere HTML5-Elemente mit speziellen Nummern (IDs) versehen werden, welche eine Zuordnung innerhalb des JavaScript-Konstrukts möglich machen. Darüber hinaus besteht die Option, diese Elemente wie bspw. Bilder oder Boxen mittels CSS3-Anweisungen zu verändern. Diese Veränderungen können bspw. Rotationen, Skalierungen oder Beschneidungen der Elemente enthalten. Nachdem eine Verbindung zwischen Engine und Canvas besteht, kann eine 3D-Szene implementiert werden, zu welcher eine Kamera hinzugefügt wird. Dieses Gebilde kann nun an die Render-Engine übergeben werden. Durch eine sogenannte „RenderLoop-Funktion“ kann die Szene nun wiederholt ausgegeben werden. Wie oft eine Szene pro Sekunde berechnet wird, hängt von der Leistung des Computers ab. Darüber hinaus kann die Anzahl der Bilder pro Sekunde durch bestimmte Script-Anweisungen selbst definiert werden.

3.6.1 Szenenausgabe mit Babylon.js für das Pyramiden-System

Im Folgenden soll der Grundaufbau des Babylon.js-Konstrukts erklärt werden, mit welchem sich vier dreieckige Viewports erstellen lassen. Hierzu gehört neben der JavaScript basierten WebGL-Programmierung mit Hilfe des Frameworks ebenso der Seitenaufbau mittels HTML5 sowie die Anpassung der einzelnen Viewports durch CSS3-Anweisungen. Um die Umsetzung für das Pyramiden-System zu veranschaulichen, werden die folgenden Ausführung anhand einer vereinfachten 3D-Szene beschrieben (Abb. 13). Diese beinhaltet lediglich einige 3D-Grundelemente sowie vier Ka-

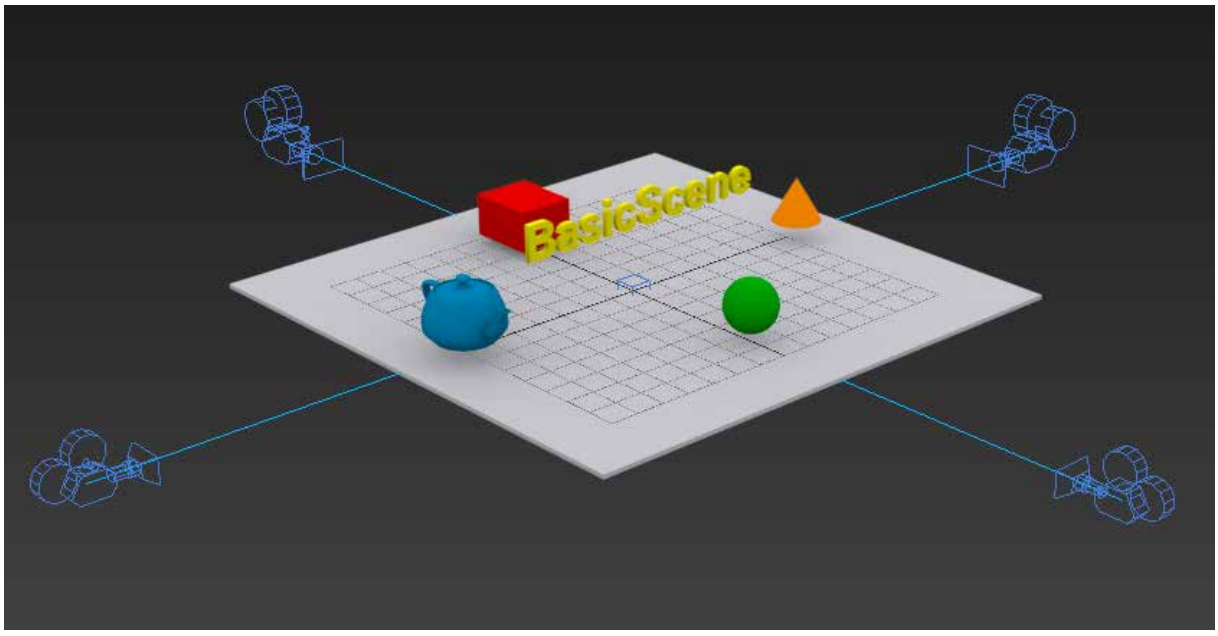

Abb. 13: *3D-Szene für die Ausgabe mittels Babylon.js, HTML5 und CSS3*

meras, welche sich um die Szene drehen lassen. Zusätzlich dienen HTML5-Texte zur Beschriftung der einzelnen Viewports. Bei den folgenden Script-Anweisungen besteht kein Anspruch auf Vollständigkeit sowie eine mustergültige Programmierung. Einzelne Anweisungen können zum Teil wesentlich vereinfacht werden, um die Ausführbarkeit des Scripts zu verbessern. Für eine vereinfachte Darlegung der notwendigen Schritte werden einige Script-Anweisungen in einer längeren Schreibweise Illustriert.

3.6.2 Aufbau einer Beispielausgabe mit HTML5, CSS3 und Babylon.js

Die im Script zur Erstellung der Viewports verwendeten CSS3-Anweisungen wurden mit dem Webbrowser „Chrome“ in der Version 50.0.2661.94 m erfolgreich getestet. Einige der Anweisungen sind mit anderen Webbrowsern nicht kompatibel und führen deshalb zu keiner korrekten Darstellung der Anzeige. Aus diesem Grund wurden ausschließlich für den Chrome-Browser definierte Anweisungen verwendet, um die Scriptlänge zu Gunsten einer besseren Verständlichkeit zu verkürzen. Für eine Kompatibilität mit anderen Browsern müssen mehrere CSS3-Anweisungen ergänzt werden.
Der einfache Grundaufbau benötigt in erster Linie drei verschiedene Dateien: Eine HTML5-Datei, welche alle erforderlichen Script-Anweisungen enthält, eine Babylon.js -Datei, innerhalb welcher die darzustellenden 3D-Elemente enthalten sind, sowie das Babylon.js-Framework. Zunächst muss das Framework in die HTML5-Datei eingebunden werden. Dies geschieht wie in Abbildung 14 zu sehen ist im Kopf der HTML5-Datei. Für eine vereinfachte JavaScript-Schreibweise wird an dieser Stelle neben dem Babylon.js-Framework ebenfalls eine JQuery-Bibliotek eingebunden, welche für den Aufbau allerdings nicht zwingend erforderlich ist. In einem weiteren Schritt werden die benötigten HTML5-Elemente definiert (Abb. 15). Zunächst erfolgt die Einbindung eines umschließenden Containers (*wrapper*) zwischen den HTML5-Body-Anweisungen (* *). Innerhalb dieses Containers können nun die vier benötigten Viewports sowie die Beschreibungstexte der einzelnen Pyramidenflächen erstellt werden. Da binnen der Canvas-Elemente keine Textanzeige integriert werden kann, werden zusätzliche Boxen benötigt, um eine Beschriftung der einzelnen Viewports zu ermöglichen (* [...] *). Zu diesem Zweck werden zunächst vier

```
<head>
    <title>BasicScene</title>
    <script type='text/javascript' src="babylon.js"></script>
    <script src="jquery.min.js"></script>
</head>
```

Abb. 14: *Einbinden des Babylon.js- und JQuery-Frameworks innerhalb des Kopfes des HTML5-Dokuments*

Div-Container deklariert, die zwecks weiterer Verwendung mit jeweils einer eigenen ID versehen werden. Der Beschreibungstext befindet sich direkt innerhalb der einzelnen Container, wobei eine Textauszeichnung mittels eines gesonderten CSS3-Abschnitts erfolgt. Innerhalb des Wrappers befinden sich neben den Textboxen die bereits erwähnten Canvas-Elemente zur WebGL-Ausgabe. Diese Elemente sind ebenfalls mit einer einmalig vorkommenden Identifikation versehen, da die vier Canvas-Elemente im weiteren Script-Verlauf einzeln angesprochen werden müssen (*<canvas id="renderCanvas1"></canvas>*).
In einem weiteren Schritt werden CSS3-Anweisungen definiert, welche notwendig sind, um eine korrekte Ausgabe der Elemente auf der Pyramide zu erzielen (Abb. 16). Die notwendigen Auszeichnungen erfolgen hierbei in einem für CSS3-Befehle gekennzeichneten Abschnitt, innerhalb des HTML5-Dokuments. Zunächst muss die Basis der HTML5-Seite angepasst werden. Dies geschieht innerhalb der ersten CSS3-Anweisungen in Abbildung 16. Da HTML5 im Standard einen äußeren Rand von mehreren Pixeln erzeugt, müssen der HTML-Anweisung und dem Body-Tag vordefinierte Ränder von 0 Pixeln zugewiesen werden. Anderenfalls kann es zu einer verzerrten Darstellung auf dem Monitor kommen, da sich die vier Viewports ansonsten nicht mehr mittig vom Display befinden. Darüber hinaus muss das Einblenden von Scrollbalken auf der Webseite blockiert werden. Diese besitzen ebenfalls eine bestimmte Pixelbreite und würden zu einer falschen Darstellung führen. Das Ausblenden der horizontalen sowie vertikalen Scrollbalken wird durch die Anweisung *overflow: hidden;* erzeugt. Da die Ausgangs-Hintergrundfarbe einer HTML5-Seite weiß ist, die Umgebung der Szene allerdings transparent auf der Pyramide ausgegeben werden soll, wird der Seitenhintergrund als schwarz definiert (*background-color: #000000;*). In einem nächsten Schritt erfolgt die Positionierung des umschließenden Div-Containers. Der Wrapper wird innerhalb des Seitenaufbaus *relativ* positioniert, da sich die Größe sowie Ausrichtung des Containers an der Seitengröße ausrichten soll. Durch einen flexiblen Aufbau

```
<body>
    <div id="wrapper">
        <div id ="text1"><div id = "text">Canvas1</div></div>
        <div id ="text2"><div id = "text">Canvas2</div></div>
        <div id ="text3"><div id = "text">Canvas3</div></div>
        <div id ="text4"><div id = "text">Canvas4</div></div>
        <canvas id="renderCanvas1"></canvas>
        <canvas id="renderCanvas2"></canvas>
        <canvas id="renderCanvas3"></canvas>
        <canvas id="renderCanvas4"></canvas>
    </div>
</body>
```

Abb. 15: *HTML5-Aufbau der benötigten Canvas- und Text-Elemente innerhalb eines umschließenden Div-Containers*

der einzelnen Elemente ist es auf verschiedenen Monitoren möglich, eine korrekte Ausgabe des Inhalts zu erzeugen. Für die fehlerfreie Darstellung auf der Plexiglaspyramide sollte ausschließlich der Vollbildmodus des Browsers verwendet werden. Andernfalls wird der Viewport-Aufbau nicht mittig auf dem Monitor ausgegeben, wodurch die Darstellung auf den einzelnen Pyramidenflächen nicht mehr korrekt angezeigt werden kann. Mit der Anweisung *margin* können Außenabstände von Div-Boxen definiert werden. In diesem Fall dient die Verwendung von automatischen Randgrößen der horizontalen Zentrierung des umschließenden Containers. Dies kann realisiert werden, indem die Abstände rechts sowie links mit dem Befehl *auto* definiert werden. Sind diese Auszeichnungen nicht vorhanden, erscheint die Anzeige linksbündig, was ebenfalls eine notwendige Verschiebung der Pyramide zur Folge hätte. In einem weiteren Schritt wird die gesamte Anzeige des Wrappers vertikal an seiner Y-Achse gespiegelt. Wie im Kapitel 3.3 „Anforderungen an Umsetzung und Technik“ beschrieben wurde, führt die Reflexion zu einer Umkehr der Anzeige. Der Befehl *transform: scaleX(-1);* skaliert den Wrapper entlang seiner breite um den Faktor -1. Durch die negative Skalierung wird die horizontale Ausrichtung des Containers umgekehrt, was eine Spiegelung des gesamten Inhalts an der Y-Achse zur Folge hat. Durch die Spiegelung an der vertikalen Achse werden die Positionen des linken und rechten Viewports vertauscht. Bei der negativen Skalierung werden ebenfalls die Elemente des oberen und unteren Viewports von links nach rechts gespiegelt. Dadurch liegen die einzelnen Perspektiven wieder in der richtigen Ausrichtung zueinander. Wie in Abbildung 17 zu sehen ist, steht die grüne Sphäre vor der Spiegelung auf der linken Anzeigefläche im Vordergrund. Nach der

```
    <style type='text/css'>

html, body {
            overflow: hidden;
            padding: 0;
            margin: 0;
            background-color: #000000;
            }

#wrapper{
            position: relative;
            margin-left: auto;
            margin-right: auto;
            transform: scaleX(-1);
            }

#renderCanvas1 {
            position: absolute;
            top: 50%;
            width: 100%;
            height: 50%;
            z-index: 1;
            touch-action: none;
            -webkit-clip-path: polygon(50% 0%, 0% 100%, 100% 100%);
            transform-origin: 50% 0;
            -webkit-transform: rotate(0deg);
            }
```

Abb. 16: *Notwendige CSS3-Anweisungen, welche das Aussehen der einzelnen Elemente innerhalb des Browsers definieren*

Transformation liegt an dieser Stelle die rote Box vorn, da Canvas2 und Canvas4 die Seiten gewechselt haben (Abb. 18). Gleichzeitig wird bei Canvas1 deutlich, dass auch dort die grüne Sphäre und die rote Box ihre Seiten getauscht haben. Simultan haben sich die Anzeigen von Kegel und Teekanne um 180 Grad gedreht und sind dementsprechend wieder richtig ausgerichtet. Bei diesem Vorgehen sollte beachtet werden, dass sich die Orientierung der Szene verändert. Das bedeutet, dass eine geplante Rotation der Kamera im Uhrzeigersinn um die Szene auf dem Monitor entgegen dem Uhrzeigersinn ausgegeben wird. Durch die Spiegelung auf der Plexiglaspyramide wird die Drehung der Kamera allerdings wie geplant im Uhrzeigersinn ausgegeben.
Als nächstes erfolgt die CSS3-Positionierung der einzelnen dreieckigen Canvas-Elemente (*#renderCanvas1*) (Abb. 16). Diese besitzen jeweils die Breite des Wrappers mit einer Angabe von 100 Prozent (*width: 100%;*). Das bewirkt, dass sich die einzelnen Flächen an die Skalierung des Wrappers anpassen. Um vier Flächen darstellen zu können, beträgt die Höhe der einzelnen Elemente jeweils 50 Prozent des umschließenden Containers (*height: 50%;*). Damit sich die dreieckigen Flächen am unteren Rand ausrichten, wird eine Verschiebung zur gegenüberliegenden äußeren Kante des Wrappers mit 50 Prozent der Höhe des Containers angegeben (*top: 50%;*). Die Zuweisung eines *z-index* ist notwendig, um die Beschriftung der Viewports mit einem höheren Index im Vordergrund anzeigen zu können. Durch den Befehl *touch-action: none;* wird verhindert, dass sich ein Klicken der Maus auf das Canvas-Element auf die Rotation der zugewiesenen Kamera auswirken kann. Damit die noch rechteckigen Viewports in eine dreieckige Form gebracht werden können, wird der Befehl *-webkit-clip-path* angewendet. Um die Spitze des Dreiecks in die obere Mitte des Canvas-Elements zu verschieben, betragen die ersten beiden Werte des Befehls 50 Prozent auf der X-Achse sowie 0 Prozent auf der Y-Achse. Ein weiterer Eckpunkt der

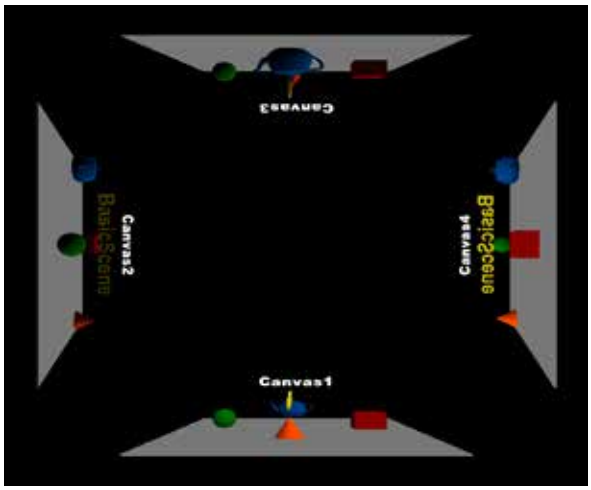

Abb. 17: *Szenen-Aufbau ohne Spiegelung an der Y-Achse*

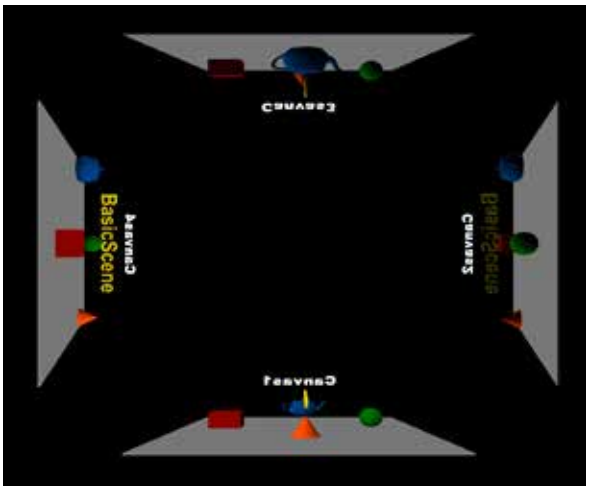

Abb. 18: *Szenen-Aufbau mit Spiegelung an der Y-Achse*

Beschneidung wird auf 0 sowie 100 Prozent gesetzt, um den linken unteren Eckpunkt zu beschreiben. Gleiches gilt für die rechte untere Seite des Dreiecks, wobei der X- und Y-Wert der Kante in beiden Fällen 100 Prozent beträgt. Um die einzelnen Viewports im Uhrzeigersinn drehen zu können, wird der Ankerpunkt der dreieckigen Flächen an die Spitze des Canvas-Elements versetzt (*transform-origin: 50% 0;*). In der letzten Code-Zeile der CSS3-Anweisung wird die Rotation der einzelnen Canvas-Elemente beschrieben (*-webkit-transform: rotate (0deg);*). Hierbei wird jeweils eine 90 Grad-Drehung im Uhrzeigersinn implementiert. Die Rotation von 0 Grad im ersten Canvas-Element dient lediglich zur Veranschaulichung. Im zweiten Element sind an dieser Stelle 90 Grad, im Dritten 180 Grad sowie im vierten Canvas-Container 270 Grad definiert. Die Textboxen für die Flächenbeschriftung werden in ähnlicher Weise beschrieben. Ein Unterschied liegt lediglich in einem höheren *z-index*, um eine Positionierung über den Viewports zu ermöglichen. Da die Erstellung der Canvas-Beschriftung für den technischen Ablauf keine Rolle spielt, wird diesbezüglich auf weitere Ausführungen verzichtet. Nachdem die notwendigen CSS3-Auszeichnungen definiert wurden, kann die technische Programmierung der Umsetzung stattfinden. Die JavaScript-Syntax erfolgt hierbei ebenfalls in einem eigens dafür ausgezeichneten Script-Teil innerhalb des HTML5-Dokuments. Zunächst muss die Größenanpassung des Wrappers realisiert werden (Abb.19). Da die Höhe des Monitors die Grundfläche der Pyramide bestimmt, kann mittels der Gesamthöhe der Seite, die Größe des Containers berechnet werden. Hierzu wird mittels JQuery die Pixelhöhe des Browser-Fensters (*$(window).height();*) an die Variable *wrapperheight* übergeben. Da der Wrapper-Container über eine eindeutige ID verfügt, kann der Wrapper mittels JavaScript angesteuert werden. Dadurch ist es möglich, die Höhe des Containers mit der Höhe des Browser-Fensters gleich zu setzen. Da die Pyramide über eine quadratische Grundfläche verfügen soll, kann die Breite des Wrappers ebenfalls der Höhe der Internetseite entsprechen. Damit sich die Maße des Wrapper-Containers bei der Skalierung des Browsers anpassen, wird die Flächenberechnung zusätzlich in einer Rezise-Funktion notiert. Anderenfalls würden die Maße der Größe des Browsers bei Erstaufruf erhalten bleiben und sich keiner Skalierung anpassen. Um eine Bewegung der Kamera bei laufender Animationsausgabe auf dem Browser zu realisieren, müssen bestimmte Variablen vordefiniert werden

```
var wrapperheight = $(window).height();
$('#wrapper').height(wrapperheight);
$('#wrapper').width(wrapperheight);

window.onresize = function(event) {
    var wrapperheight = $(window).height();
    $('#wrapper').height(wrapperheight);
    $('#wrapper').width(wrapperheight);
};
```

Abb. 19: *Auf JavaScript basierende Transformation des Wrappers, zwecks Größenanpassung an das Browserfenster*

(Abb. 20). Diese Variablen sollen per Tastendruck verändert werden, um eine Rotation oder Bewegung der Kamera auszulösen. Zu diesem Zweck werden Variablen für die Höhe des Kamerafokuspunkts (*targetheight*), der Kamerahöhe (*cameraheight*), der Distanz der Kamera zum Fokuspunkt (*cameradistance*) sowie der Kamerarotation (*camerarotation*) definiert. Mit der Zuweisung fester Werte werden die vier Kameras bei Erstaufruf der Seite auf die vergebenen Positionen gesetzt. In diesem Fall entspricht die Höhe der Kamera sowie die des Zielpunktes einer Entfernung von 50 Zentimetern zum Nullpunkt auf der Y-Achse. Im Gegensatz zu 3ds Max liegt die Höhenposition der Kamera nicht auf der Z-Achse des Koordinatensystems. Die Unterschiede in der Ausrichtung von Objekten werden allerdings beim Export durch das Plugin korrigiert, sodass sich die dargestellten Objekte in der richtigen Orientierung im Koordinatensystem von Babylon.js befinden. Um in Echtzeit mit dem System interagieren zu können, wurde wie bereits erwähnt eine Kamerasteuerung implementiert. Diese wird mittels eines JavaScript-Events *onKeyDown* realisiert. Zu diesem Zweck wurden mehrere Tasten des Keyboards mit verschiedenen Zuweisungen belegt (Abb. 21). Zur Überprüfung, welche Taste vom Benutzer gedrückt wurde, dient eine Switch-Case-Funktion. Diese überprüft die zu betätigenden Keyboard-Tasten auf ihren Keycode. Entspricht dieser bspw. der Zahl 87, wurde die W-Taste auf der Tastatur des PCs betätigt. In diesem Fall wird die bereits festgelegte Variable der Kameradistanz zu ihrem Fokuspunkt in ihrem Wert verändert. Hierbei wird die vorgegebene Variable mit sich selbst weniger 10 Einheiten überschrieben (*cameradistance = cameradistance-10;*). Dadurch verringert sich der Ausgangswert von 300 auf 290 Zentimeter, sodass die Kamera näher an ihren Fokuspunkt gerückt wird. In gleicher Weise wurden ebenso Tastaturbelegungen für die Höhe der Kamera sowie dessen Rotationswerte realisiert. Es handelt sich bei dieser

```
var targetheight = 50;
var cameraheight = 50;
var cameradistance = 300;
var camerarotation = 0;
```

Abb. 20: *Initialisierung von Variablen, deren Werte bei laufender Animation verändert werden können, um eine Bewegung der Kamera zu erzeugen*

```
var onKeyDown = function(event) {
        var key = event.keyCode;
        var ch = key;
         switch (ch) {
                        case 87: //W
                            cameradistance = cameradistance-10;
                            break;
                        case 83://A
                            cameradistance = cameradistance+10;
                            break;
                        case 65://S
```

Abb. 21: *Implementieren von Tastatureingaben, welche die Ausgabe einer Beweglichen Kamera bei laufender Echtzeitausgabe ermöglichen*

Umsetzung um die reine Demonstration von Interaktion mit der Szene. Im nächsten Schritt erfolgt die Implementierung der für die 3D-Ausgabe wichtigsten Funktionen des Babylon.js-Frameworks (Abb. 22). Zunächst werden die vier bereits im HTML5-Bereich des Dokuments eingebundenen Canvas-Elemente in vier einzelne Variablen überführt, sodass diese in das Framework integriert werden können. Dies geschieht über die Funktion *document.getElementById().* Der Befehl bildet eine Referenz zu dem Ursprungsobjekt wie in diesem Fall zu den Canvas-Containern. Da es nicht möglich ist, vier Canvas-Elemente mit einer einzelnen Render-Engine zu verbinden, muss jedem Container eine eigene Engine zugewiesen werden. Dies erfolgt mittels der Babylon.js-Anweisung *new BABYLON.Engine();*. Hierbei wird das Konstrukt aus Engine und dem zugewiesenen Canvas-Element in eine eigene Variable übergeben, um in einem weiteren Schritt darauf zugreifen zu können. Als nächstes erfolgt die Zuweisung der einzelnen Engines zu einem Babylon-Szenen-Objekt. Dieses wird im Verlauf des Scripts mit der aus 3ds Max exportierten 3D-Szene gefüllt. Des Weiteren kann das Babylon-Szenen-Konstrukt mit verschiedenen Eigenschaften versehen werden. Mit dem Befehl *BasicScene1.clearColor* kann der Szene bspw. eine Hintergrundfarbe zugewiesen werden. Im Fall des Versuchsaufbaus sollte diese schwarz sein, um einen Transparenz-Effekt auf der Pyramide zu erzeugen.
Als nächstes werden die benötigten Kameras für die einzelnen Szenen erstellt (Abb. 23). Hierbei wird eine sogenannte „*BABYLON.ArcRotateCamera*“ verwendet. Diese verfügt im Gegensatz zu einer freien Kamera über einen Zielpunkt, welcher auf den Mittelpunkt der Szene gesetzt werden kann. Dadurch ist es möglich, die Kameras durch Rotation um die Szene herum zubewegen. Bei einer freien Kamera gestaltet sich die-

```
var canvas1 = document.getElementById("renderCanvas1");
var canvas2 = document.getElementById("renderCanvas2");
var canvas3 = document.getElementById("renderCanvas3");
var canvas4 = document.getElementById("renderCanvas4");


var engine1 = new BABYLON.Engine(canvas1, true);
var engine2 = new BABYLON.Engine(canvas2, true);
var engine3 = new BABYLON.Engine(canvas3, true);
var engine4 = new BABYLON.Engine(canvas4, true);


var BasicScene1 = new BABYLON.Scene(engine1);
BasicScene1.clearColor = new BABYLON.Color3(0,0,0);

var BasicScene2 = new BABYLON.Scene(engine2);
BasicScene2.clearColor = new BABYLON.Color3(0,0,0);
```

Abb. 22: *Verknüpfung von Canvas-Element, Render-Engine und Szenen-Objekt*

```
var camera1 = new BABYLON.ArcRotateCamera("ArcRotateCamera", 1, 1, 10, new BABYLON.Vector3(0, 0, 0), BasicScene1);

var camera2 = new BABYLON.ArcRotateCamera("ArcRotateCamera", 1, 1, 10, new BABYLON.Vector3(0, 0, 0), BasicScene2);

var camera3 = new BABYLON.ArcRotateCamera("ArcRotateCamera", 1, 1, 10, new BABYLON.Vector3(0, 0, 0), BasicScene3);

var camera4 = new BABYLON.ArcRotateCamera("ArcRotateCamera", 1, 1, 10, new BABYLON.Vector3(0, 0, 0), BasicScene4);
```

Abb. 23: *Implementieren von vier Kameras mittels Babylon.js-Syntax*

ser Vorgang wesentlich schwieriger, da eine Rotation der Kamera, eine Drehung um sich selbst darstellt. Wie bereits erwähnt wurde eine Variable zur Höhenangabe des Fokuspunkts vergeben. Dadurch kann dieser bei laufender Animation verändert werden, wodurch eine Verschiebung der Szene nach oben oder unten simuliert wird. Die verschiedenen Kameras werden jeweils einer der erstellten Szenen hinzugefügt, um eine Ansicht aus vier verschiedenen Perspektiven zu ermöglichen. Um mit den Kameras weiter arbeiten zu können und deren Einstellungen über das Script anzupassen, wird die Syntax der Kameraimplementierung in eine eigene Variable überführt.
In einem weiteren Schritt werden die aus 3ds Max exportierten 3D-Objekte in die erstellten Babylon-Szenen importiert (Abb. 24). Im Fall des zu Erklärungszwecken erstellten Aufbaus befinden sich mehrere 3D-Objekte innerhalb einer einzelnen Babylon-3D-Datei. Für die Implementierung der 3D-Elemente stellt das Framework eine eigene Funktion bereit, die das 3D-Objekt für die einzelnen Szenen zugänglich macht. Hierbei wird der Funktion der Pfad zu der exportierten Babylon-3D-Datei hinzugefügt, wodurch auf die 3D-Datei zugegriffen werden kann. Bevor die vier Babylon-Szenen mit den 3D-Objekten gefüllt werden, wird eine Variable bestimmt, die das Zählen der geladenen Objekte ermöglicht (*var scenecounter = 0;*). Dieser Zähler ist notwendig, damit das Rendern der einzelnen Szenen simultan erfolgen kann. Zu diesem Zweck wird die Zählvariable mit der Operation ++ innerhalb der Import-Funktion jeweils um 1 erhöht. Dadurch enthält der Zähler nach Abschluss aller Ladevorgänge den Wert 4. Erst wenn dieser erreicht wurde, sollen die vier Szenen auf dem Canvas-Container ausgegeben werden. Verwendung findet der Vorgangszähler im folgenden Anweisungsschritt, dem Rendern der einzelnen Szenen (Abb. 25). Um die Darstellung der 3D-Objekte zu ermöglichen, wird eine Funktion benötigt, die das Rendern auf Seiten der Engines auslöst. Für das Induzieren des Rendervorgangs bietet Babylon.js eine eigene Funktion an. Die sogenannte „RenderLoop“ ist eine auf das Ausgeben von Babylon-3D-Szenen optimierte Funktion. Gestartet wird der Rendervorgang mit der Anweisung *engine.runRenderLoop()*;. Da die vier Szenen exakt gleichzeitig ausgegeben werden sollen, wird

```
var scenecounter = 0;

BABYLON.SceneLoader.ImportMesh("", "", "BasicScene.babylon", BasicScene1, function (newMeshes) {
scenecounter++;
});

BABYLON.SceneLoader.ImportMesh("", "", "BasicScene.babylon", BasicScene2, function (newMeshes) {
scenecounter++;
});

BABYLON.SceneLoader.ImportMesh("", "", "BasicScene.babylon", BasicScene3, function (newMeshes) {
scenecounter++;
});

BABYLON.SceneLoader.ImportMesh("", "", "BasicScene.babylon", BasicScene4, function (newMeshes) {
scenecounter++;
});
```

Abb. 24: *Laden der 3D-Objekte mit gleichzeitiger Zuweisung zu den einzelnen Szenen-Objekten sowie Implementierung eines Ladevorgang-Zählers*

das wiederholte Auslösen des Rendervorgangs von einer einzelnen Engine übernommen. Wenn jede der vier Engines eine eigene RenderLoop verwendet, kann nicht sicher gestellt werden, dass die Bildrate auf allen vier Canvas-Elementen gleich ist. Für die wiederholte Bildausgabe können neben der optimierten-RenderLoop-Funktion auch andere Javascript-Funktionen verwendet werden, welche ein wiederholtes Rendern der Szene erzeugen. Nachdem die RenderLoop gestartet wurde, erfolgt die Abfrage nach dem Inhalt des Ladevorgang-Zählers mittels einer If-Anweisung. Erst wenn der Zähler den Wert 4 für die abgeschlossenen Ladevorgänge erreicht hat, sollen weitere Ausführungen vollzogen werden. Ist dies der Fall, werden zunächst die Kamerapositionen, die Lage der Kamerafokuspunkte sowie die Rotationswerte der Kameras neu definiert (*camera1.setPosition*). Die Ausrichtung der Kameras erfolgt hierbei über die vordefinierten Variablen. Sobald ein Tastendruck für die Veränderung des Variableninhalts sorgt, wird der neue Wert als Positionsangabe der Kameras verwendet. Da diese Funktionen innerhalb der RenderLoop liegen, wird die Position der Kamera bei jedem gerenderten Bild erneut abgefragt. Wichtig ist an dieser Stelle die genaue Bestimmung der Alpha-Rotation, also der Rotation um den Fokuspunkt der Kameras. Die Rotationsgrade der Kameras müssen jeweils 90 Grad auseinander liegen, um vier verschiedene Perspektiven auf der Pyramide darstellen zu können. Hierbei muss die Umrechnung von Rotationsgraden in Radianten erfolgen, da Babylon.js nicht mit Gradangaben arbeiten kann. Ein weiteres Problem stellt die unterschiedliche Interpretation von Rotationsgraden von CSS3 und Babylon.js dar. Während eine Drehung um 90 Grad von CSS3 im Uhrzeigersinn dargestellt wird, arbeitet Babylon.js bei der Rotation von Kameras entgegen dem Uhrzeigersinn. Dementsprechend müssen

```
engine1.runRenderLoop(function () {
    if(scenecounter==4)
    {
    camera1.setPosition(new BABYLON.Vector3(0, kameraheight, cameradistance));
    camera1.target.position = new BABYLON.Vector3(0, targetheight, 0);
    BasicScene1.activeCamera.alpha = 0.0 + camerarotation;
    BasicScene1.render();

    camera2.setPosition(new BABYLON.Vector3(0, kameraheight, cameradistance));
    camera2.target.position = new BABYLON.Vector3(0, targetheight, 0);
    BasicScene2.activeCamera.alpha = (270 * Math.PI / 180) + camerarotation;
    BasicScene2.render();

    camera3.setPosition(new BABYLON.Vector3(0, kameraheight, cameradistance));
    camera3.target.position = new BABYLON.Vector3(0, targetheight, 0);
    BasicScene3.activeCamera.alpha = (180 * Math.PI / 180) +camerarotation;
    BasicScene3.render();

    camera4.setPosition(new BABYLON.Vector3(0, kameraheight, cameradistance));
    camera4.target.position = new BABYLON.Vector3(0, targetheight, 0);
    BasicScene4.activeCamera.alpha = (90 * Math.PI / 180)+camerarotation;
    BasicScene4.render();
    };
    });
```

Abb. 25: *Rendern der vier verschiedenen Ansichten der Szene mittels einer runRenderLoop-Funktion*

```
window.addEventListener("resize", function () {
            engine1.resize();
            engine2.resize();
            engine3.resize();
            engine4.resize();
});
```

Abb. 26: *Anpassung der Pixelauflösung der ausgegebenen Szene an die Skalierung des Browserfensters*

die Drehungen von 90 und 270 Grad vertauscht werden, um die richtige Perspektive auf dem entsprechenden Canvas-Element ausgeben zu können. Um eine Drehung der vier Kameras zu bewirken, wird auf die errechneten Gradzahlen bei Tastendruck der Wert der Variable *camerarotation* aufaddiert (*BasicScene2.activeCamera.alpha = (270 * Math.PI /180) + camerarotation;*). Dadurch rotieren alle Kameras anhand ihres Ausgangswinkels gleichmäßig um die Szene, wodurch der Eindruck entsteht, dass die sich die Szene selbst innerhalb der Plexiglaspyramide dreht. In einem letzten Schritt muss die Auflösung der ausgegebenen Szene an die Pixelauflösung des Browserfensters angepasst werden (Abb. 26). Hierfür bietet Babylon.js die Funktion *engine.rezise* an. Diese Funktion wird bei der Skalierung des Browserfensters für jede Render-Engine ausgeführt. Wird dieser Schritt nicht berücksichtigt, bleibt die Pixelauflösung der Szene bei erstmaligem Seitenaufruf erhalten. Wird die Seitenanzeige anschließend in den Vollbildmodus geschaltet, wird eine zu niedrige Auflösung angezeigt, was die Bilddarstellung der Szene verschlechtert. Durch die Implementierung der Resize-Funktion kann das Anzeigen einer falschen Auflösung verhindert werden.

3.6.3 Vor- und Nachteile von HTML5, CSS3 und Babylon.js

Durch die Verwendung von HTML5, CSS3 und dem Babylon-Framework ergeben sich verschiedene Vor- sowie Nachteile bei der Anwendungsrealisierung. Sehr positiv zu bewerten ist die Tatsache, dass bei der programmiertechnischen Umsetzungen keinerlei Kosten entstehen. Die Verwendung von Babylon.js ist völlig kostenfrei und kann mittels eines einfachen Texteditors erfolgen. Da Babylon.js für den Export der 3D-Dateien ein Plugin für die OpenSource 3D-Modellierungssoftware Blender anbietet, entstehen an dieser Stelle ebenfalls keine zusätzlichen Kosten. Babylon.js basiert auf JavaScript, daher gibt es sehr viele Möglichkeiten verschiedenste Frameworks miteinander zu kombinieren und so das Verhalten von den Objekten innerhalb der Pyramide zu verändern (Babylon.js 2016a). So ist es bspw. möglich, durch die Verwendung verschiedener Frameworks Schwerkraft innerhalb der Szene zu erzeugen. Auch Abfragen von Kollisionen der Objekte untereinander können durch entsprechen-

de Anweisungen realisiert werden. Dementsprechend ist es möglich, sehr viele verschiedene Arten von Anwendungen wie bspw. Spiele, Architekturvisualisierungen oder Produktpräsentationen zu realisieren. Babylon.js verfügt über diverse Bausteine, welche bei einer Umsetzung integriert werden können. Hierzu zählen z.B. Partikeleffekte, durch welche sich Rauch oder Feuer erstellen lassen, oder auf den 3D-Objekten frei platzierbare Aufkleber (Babylon.js 2016b). Des Weiteren könnte mit Hilfe des Frameworks eine Art visueller Editor realisiert werden, welcher es ermöglicht, 3D-Objekte in Echtzeit innerhalb der Pyramide auszutauschen. Darüber ließe sich der Arbeitsablauf bei der Realisierung von Anwendungen für die Pyramide erheblich vereinfachen. Ein entscheidender Vorteil bei der Verwendung von Browsern als Ausgabemedium ist die Möglichkeit, die Visualisierung über Webserver zu veröffentlichen. Durch die Flexibilität der Ausgabegröße und Browser-Anbindung ist es bspw. möglich, die erstellte Szene auch auf einem Mobilgerät in kleiner Form zu sehen. Somit können sich mehrere Personen gleichzeitig die Umsetzung auf verschiedenen Endgeräten anschauen und gemeinsam an einer Realisierung oder Verbesserungen arbeiten. Da WebGL vergleichsweise geringe Anforderungen an die Leistung des Computers stellt, kann im Fall eines mobilen Einsatzes ein kostengünstiger Mini-PC verwendet werden, um die Visualisierung auf der Pyramide auszugeben. Die Verwendung des Browsers als Ausgabemedium hat allerdings neben den genannten Vorteilen erhebliche Nachteile. Bei der Umsetzung von interaktiven Echtzeitanwendungen für eine holographische Pyramide mittels HTML5, CSS3 und WebGL muss sehr stark darauf geachtet werden, dass die erzeugte Datenmenge nicht zu groß wird. Bei der Verwendung von 3D-Objekten, Texturen und Effekten fallen relativ große Datenmengen an, welche über das Internet transportiert werden. Zwar kann die Umsetzung auch lokal im Browser gestartet werden, allerdings ist gerade der flexible Austausch der Visualisierungen der große Vorteil bei dieser Art der Umsetzung. Die Größe der Datenmenge ist maßgeblich für die Ladegeschwindigkeit der Szenen verantwortlich. Je umfangreicher die Darstellung desto länger sind die Ladezeiten bis zur Anzeige der Szene. Dementsprechend sind mit dieser Methode eher einfachere Anwendungen realisierbar. Darüber hinaus befindet sich das verwendete Framework noch in der Entwicklung und verfügt dementsprechend nicht über alle Funktionen wie bspw. eine Spiele-Engine. Ein weiteres Problem ist die Darstellung in verschiedenen Browsern. Nicht alle Endgeräte verfügen über die notwendigen Browser-Eigenschaften, die eine Darstellung ermöglichen. Dementsprechend müssen auf Seiten der CSS3-Syntax verschiedenste Endgeräte oder Browser berücksichtigt werden.

3.7 Unreal Engine 4 als Entwicklungsumgebung

Wie bereits in Abschnitt 2.1.5 „Game-Engine“ beschrieben wurde, handelt es sich bei der Unreal Engine 4 (UE4) um eine vollwertige Spiele-Engine zur Entwicklung von diversen Arten digitaler Spiele oder Visualisierungen. Die mit dem System erstellten Entwicklungen können für verschiedenste Plattformen bereitgestellt werden. Hierzu zählen z.B. Smartphones, Tablets, Videospiel-Konsolen oder verschiedene Betriebssysteme. Darüber hinaus können mittels der Spiele-Engine auch 3D-Anwendungen für den Browser exportiert werden. Für die Entwicklung mit der Engine fallen zunächst keine Kosten an. Bei der Veröffentlichung von Spielen erhebt der Entwickler der Engine allerdings Abgaben an das Unternehmen, sobald mit dem Spiel ein Gewinn von 3000$ pro Quartal erzielt wird. Für die Veröffentlichung von Visualisierungen, welche bei der Produktion für die Glaspyramide in erster Linie in Frage kommen, fallen keine weiteren Abgaben an (Epic Games, Inc. 2016b). Für die Entwicklung von digitalen Anwendungen verfügt die Game-Engine über einen Editor. Dieser führt die verschiedenen Bestandteile der Engine zusammen und ermöglicht ein visuelles Zusammensetzen von verschiedenen Elementen, welche für eine Anwendung nötig sind. Hierbei werden Lichter, Objekte oder auch Kameras per Drag and Drop der Szene hinzugefügt. Neben der visuellen Bearbeitung von Objektmaterialien, Animationen, Partikeleffekten oder Sounds können diverse Einstellungen, wie bspw. die Helligkeit von Lichtern, an einzelnen Objekten direkt im Editor vorgenommen werden, ohne dass hierfür Codezeilen geschrieben werden müssen. Neben der textbasierten Programmierung mit der Programmiersprache C++ bietet die UE4 die Möglichkeit visuell zu programmieren. Für die visuelle Programmierung kommen sogenannte „Blueprints“ zum Einsatz. Der Editor bietet zu diesem Zweck viele unterschiedliche Arten von Blueprints an, mit denen es möglich ist, sowohl das Level oder enthaltene Objekte wie bspw. Kameras, Lichter oder 3D-Elemente zu editieren als auch Spielmechaniken zu implementieren, ohne Codezeilen schreiben zu müssen. Zudem besteht die Möglichkeit, die visuellen Programmabläufe bei laufender Simulation der Visualisierung zu überprüfen. Auf diese Weise können Fehler im Programmablauf sehr schnell erkannt werden, ohne dass innerhalb vieler Codezeilen nach einem möglichen Problem gesucht werden muss (Epic Games, Inc. 2016c). Innerhalb des Editors können verschiedene Dateiformate verarbeitet werden. Hierzu zählen neben den Austauschformaten FBX und OBJ für den Import von 3D-Objekten bspw. das WAV-Format für Audiodateien sowie diverse Bildformate für die Texturierung von 3D-Elementen.

3.7.1 Szenenausgabe mit der Unreal Engine 4 für das Pyramiden-System

Im folgenden Abschnitt soll der Grundaufbau einer UE4-Anwendung erklärt werden, mittels welchem es möglich ist, vier dreieckige Viewports zu erzeugen. Wie im vorangegangenen Beispiel soll der Aufbau eine Art der Interaktion mit den Kameras innerhalb der Szene zulassen. Die weiterführenden Erklärungen stellen keine Musterlösung für das vorliegende Problem dar, sondern sollen eine sehr einfache Möglichkeit der Realisierung beispielhaft erläutern. Bei der folgenden Beschreibung werden zunächst einzelne Elemente des UE4-Editors grob erklärt. Anschließend erfolgt die eigentliche Realisierung einer Echtzeitanwendung für das Pyramiden-System. Da einige notwendige Schritte bereits erläutert wurden, welche für die Ausgabe von vier verschiedenen Betrachtungswinkeln notwendig sind, wird innerhalb des folgenden Abschnitts auf Erklärungen verzichtet, die sich von den vorangegangenen Ausführungen ableiten lassen. Für die Realisierung von interaktiven Echtzeitanwendungen mittels der UE4 wurde bei allen weiteren Ausführungen, welche sich auf die Engine beziehen, auf die Online-Dokumentation von Epic Games zurückgegriffen (Epic Games, Inc. 2016d).

3.7.2 Aufbau einer Beispielausgabe mit der Unreal Engine 4

Für das bessere Verständnis der folgenden Ausführungen ist eine Kenntnis vom Aufbau des Editors der Unreal Engine erforderlich. Aus diesem Grund folgt zunächst eine Beschreibung der für die Realisierung wichtigen Komponenten des Editors (Abb. 27).

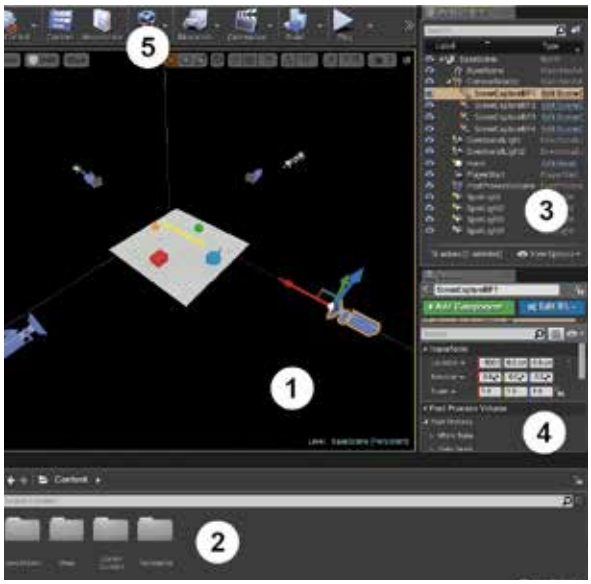

Abb. 27: *Übersicht des Unreal Engine 4-Editors*

In Position 1 ist der 3D-Viewport des Editors zu sehen. An dieser Stelle werden alle für die Ausgabe notwendigen 3D-Objekte sowie Lichter, Kameras oder Partikelsysteme positioniert und visuell angezeigt. Dem Viewport können verschiedene Arten von Objekten per Drag and Drop hinzugefügt werden. Position 2 in Abbildung 27 zeigt den sogenannten „Content Browser“. Der Content Browser stellt das Hauptareal des Editors für das Kreieren, Importieren und Organisieren von Objekten oder Blueprints dar. Per Doppelklick auf die erstellten Objekte können jeweils Untereditoren geöffnet

werden, welche die Möglichkeit bieten, weitere Einstellungen an den entsprechenden Objekten vorzunehmen. Darüber hinaus können von dieser Stelle aus Objekte in den Viewport gezogen werden. In Position 3 ist der sogenannte „World Outliner“ zu sehen. In ihm werden alle Akteure der Szene angezeigt. An dieser Position ist es möglich, verschiedene Objekte miteinander zu verknüpfen. So können bspw. mehrere Kameras mit einem bestimmten Objekt verknüpft werden. Position 4 zeigt die Detailansicht von im World Outliner ausgewählten Akteuren. Hier kann z.B. der Sichtbereich der Kamera eingestellt werden. Unter Punkt 5 befindet sich die sogenannte „Toolbar“ des Editors. Hierin lassen sich die wichtigsten Befehle des Editors finden. An dieser Stelle befindet sich unter anderem der Zugang zu einer besonderen Form von Blueprint (Level-Blueprint), in welchem sich Befehle für den gesamten Level-Inhalt verwalten lassen. Um eine Ausgabe von vier verschiedenen Perspektiven innerhalb des Editors der UE4 zu erzeugen, müssen mehrere Blueprints miteinander kombiniert werden. Die drei notwendigen Hauptkomponenten bestehen aus einer Kamera, einer Textur und einem Material, welches mit der Textur bespielt wird. Dieses Gerüst muss auf Grund der zu erstellenden vier Ansichten vier mal vorhanden sein. Die vier verschiedenen Materialien werden anschließend innerhalb einer Benutzeroberfläche (Userinterface) zusammengefasst und bei Ausführung der Anwendung über das Level-Blueprint auf dem Monitor angezeigt. An dieser Stelle erfolgt zunächst die Beschreibung der einzelnen Komponenten und deren Bezug zueinander.

In einem ersten Schritt wird eine Textur erstellt, auf welcher das Bild einer Kamera ausgegeben werden kann. Diese besondere Art der Textur wird „RenderTarget2D“ genannt und beinhaltet zunächst keine Bildinformation (Abb. 28). Wie bereits in Kapitel 3.1 beschrieben wurde, richtet sich die Größe der Pyramide nach der kürzesten Seitenlänge des Monitors. Da eine Berechnung der Maße der einzelnen Viewports in prozentualem Verhältnis zur Anzeigefläche mittels der UE4 nur erschwert zu realisieren ist, werden an dieser Stelle feste Größen verwendet. Die Anzeigefläche des Monitors oberhalb der Pyramide verfügt über eine Pixelhöhe von 1024 Pixeln. Dementsprechend ist dies die größte mögliche Breite der einzelnen, gedrehten Viewports. Da die Höhe der Anzeigeflächen der Hälfte der größten möglichen Darstellungshöhe entspricht, wird die Höhe der Textur mit 512 Pixeln

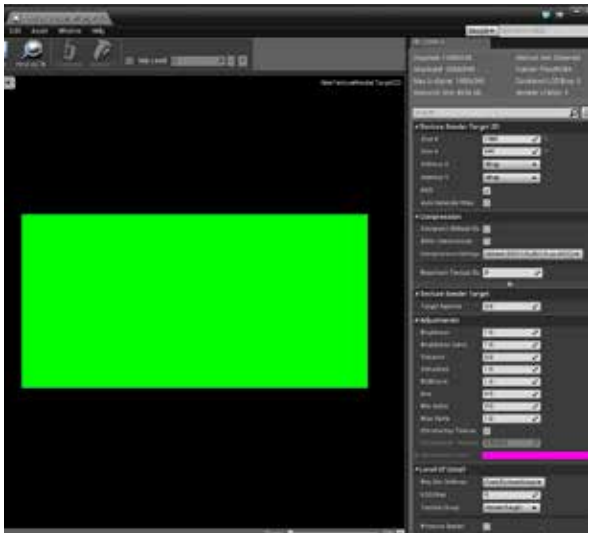

Abb. 28: *RenderTarget2D-Blueprint-Klasse ohne verknüpfte Kamera*

angegeben (Abb. 29). Die Hauptkomponente des Aufbaus ist eine sogenannte „SceneCapture2D-Blueprint-Klasse“ (Abb. 30). Diese verhält sich wie eine normale Kamera-Komponente innerhalb des Editors, verfügt darüber hinaus allerdings über besondere Einstellungsmöglichkeiten. Neben den üblichen Optionen wie bspw. der Veränderung des Aufnahmewinkels oder der Position der Kamera, besitzt die SceneCapture2D-Blueprint-Klasse die Möglichkeit, dass durch die Kamera eingefangene Abbild der 3D-Szene in einer Textur (RenderTarget2D) auszugeben. Zu diesem Zweck wird, wie in Abbildung 31 zu sehen ist, innerhalb der Kamera-Komponente eine der erstellten Texturen eingetragen. Dieser Schritt geschieht in den Detaileinstellungen der SceneCapture2D-Blueprint-Klasse innerhalb des Unterpunktes „Texture Target“. Ist an dieser Stelle eine der zuvor generierten RenderTarget2D-Texturen eingefügt, wird das Bild der Kamera-Komponente auf der entsprechenden Textur ausgegeben. Dieses Konstrukt muss vier mal vorhanden sein, da für die Pyramidenausgabe vier Kameras benötigt werden. Als nächsten Schritt werden die vier erstellten SceneCapture2D-Blueprints in den 3D-Viewport des Editors gezogen (Abb. 32). Die Kameras werden nun in einem Winkel von jeweils 90 Grad um das darzustellende Objekt, welches zuvor der Szene hinzugefügt wurde, positioniert. Auf diese Weise ist jeweils eine andere

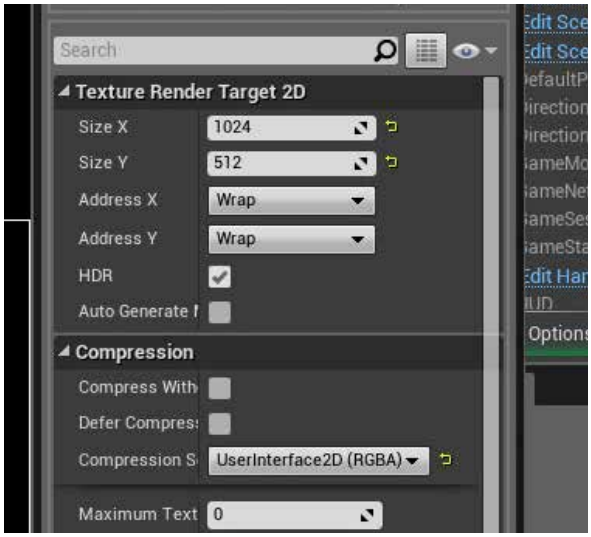

Abb. 29: *Größeneinstellung der Textur*

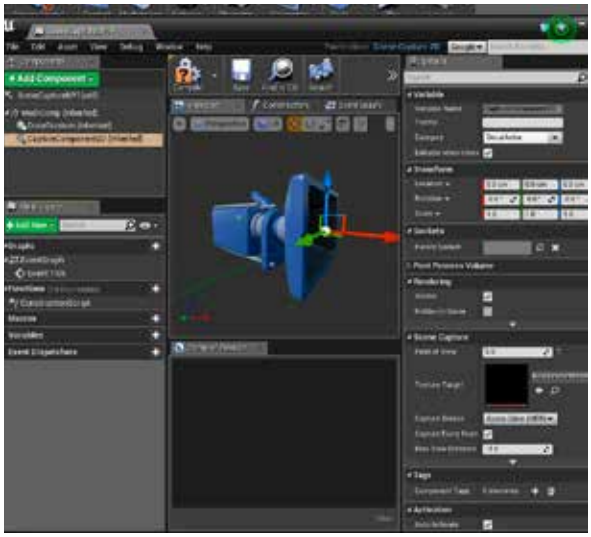

Abb. 30: *SceneCapture2D-Blueprint im Blueprint-Editor*

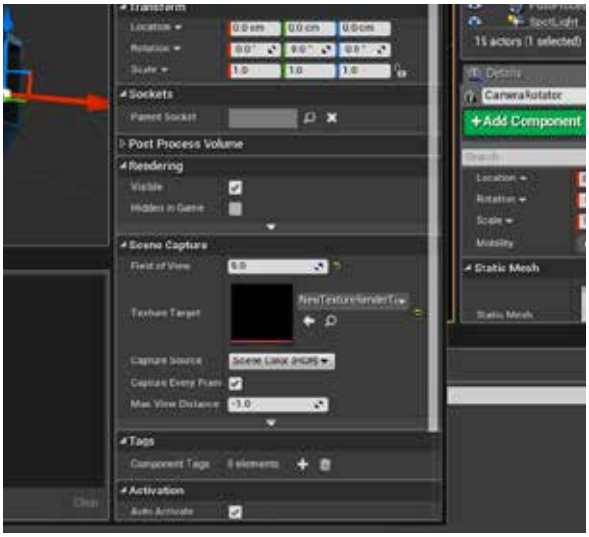

Abb. 31: *Eine Textur wird der Kamera zugewiesen*

Perspektive des Objekts zu sehen. Sobald die SceneCapture2D-Blueprints der Szene hinzugefügt wurden, erscheint deren Abbild auf den entsprechend zugewiesenen Texturen (Abb. 33). Als nächstes müssen Material-Klassen für die vier verschiedenen RenderTarget2D-Elemente erstellt werden (Abb. 34). Um die Texturen innerhalb des Materials sehen zu können, muss die Materialeigenschaft auf die Einstellung „User Interface“ gesetzt werden. Da eine dreieckige Ausgabe benötigt wird, muss an dieser Stelle der Zusatz „Masked“ ausgewählt werden. Das Material verfügt somit über zwei Knotenpunkte, wobei der Knotenpunkt „Final Color“ mit einer der Kameratexturen verbunden wird. Der zweite Knotenpunkt „Opacity Mask“ beschreibt die Beschneidung des Materials. Hierbei wird eine weitere Textur benötigt, die lediglich aus schwarzem sowie weißem Bildinhalt besteht. Diese wurde in einem weiteren Schritt separat mittels eines Bildbearbeitungsprogramms (Photoshop) angefertigt und zeigt ein weißes Dreieck auf schwarzem Grund. Hierbei werden die schwarzen Bildinhalte als transparent dargestellt, wodurch eine Beschneidung des Materials in dreieckiger Form entsteht. In einem weiteren Schritt müssen nun die vier erstellten Materialien zusammengeführt werden, um vier Ansichten ausgeben zu können. Um dies zu realisieren kann ein sogenanntes „Widget-Blueprint“ verwendet werden (Abb. 35). Widgets sind ein Bestandteil des Unreal Motion Graphics Userinterface Designer (UMG) und können diverse

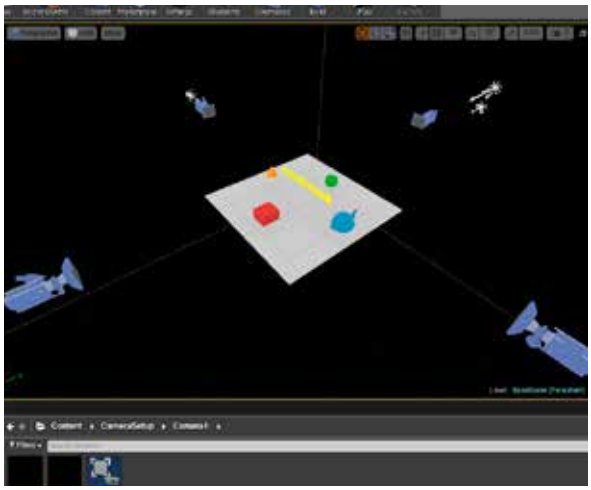

Abb. 32: *Aufbau der vier SceneCapture2D-Blueprints innerhalb des Editors*

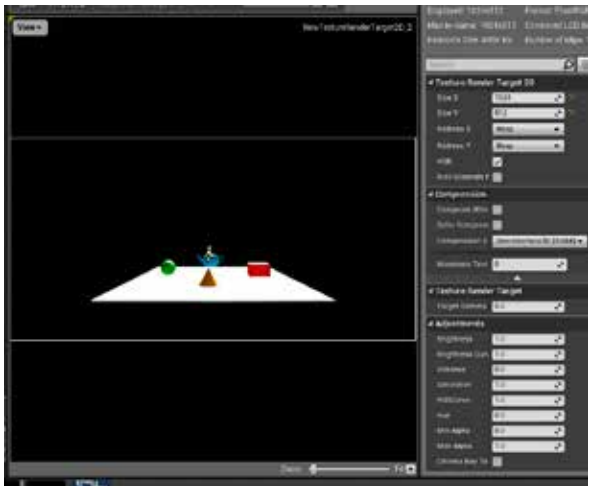

Abb. 33: *RenderTarget2D-Blueprint-Klasse mit verknüpfter Kamera*

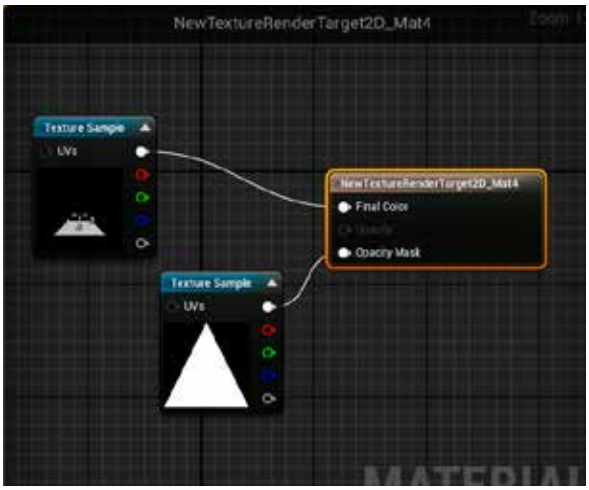

Abb. 34: *RenderTarget2D-Material mit verknüpfter Kamera*

Funktionen übernehmen. Mit ihrer Hilfe ist es bei Computerspielen bspw. möglich, Schriften, Lebensanzeigen des Spielers oder Bilder, im Vordergrund der 3D-Spielfläche, auf dem Monitor auszugeben. In diesem Fall wird ein Widget dazu verwendet, die vier verschiedenen Kameraperspektiven darzustellen. Innerhalb des Widget-Editors werden nun vier Image-Komponenten erstellt, welche die gleichen Maße wie die zuvor erstellten Texturen aufweisen (Abb. 35). Anschließend können den Image-Komponenten die verschiedenen Materialien zugewiesen werden, sodass die entsprechend beschnittenen Kameraperspektiven innerhalb der Image-Komponenten ausgegeben werden. Als nächstes werden die einzelnen Image-Elemente jeweils in einem Winkel von 90 Grad gedreht und entsprechend ihrer Kameraperspektive ausgerichtet. Wie bereits erläutert, müssen die einzelnen Anzeigen gespiegelt werden, um eine korrekte Darstellung auf der Pyramide zu erzeugen. Dies kann erreicht werden, indem die Image-Komponenten in ihrem X-Wert um -1 skaliert werden. Dadurch dreht sich die Anzeige der 3D-Szene um und wird durch die Reflexion auf der Glaspyramide richtig herum dargestellt. Da hierbei nicht alle Widgets über eine gemeinsame Achse, sondern an der Y-Achse der einzelnen Komponenten gespiegelt werden, ist es notwendig, die Perspektive eines sich gegenüberliegenden Kamerapaares zu vertauschen. Als nächstes erfolgt die Ausgabe des Widgets bei Start der Anwendung. Dieses Verhalten muss in dem Level-Blueprint visuell einprogrammiert werden. Wie in Abbildung 36 zu sehen ist, wird bei Start der Anwendung (*Event BeginPlay*) ein Widget kreiert (*Create Widget BP1 Widget*). Innerhalb des Create Widget-Befehls wird die bereits erstellte

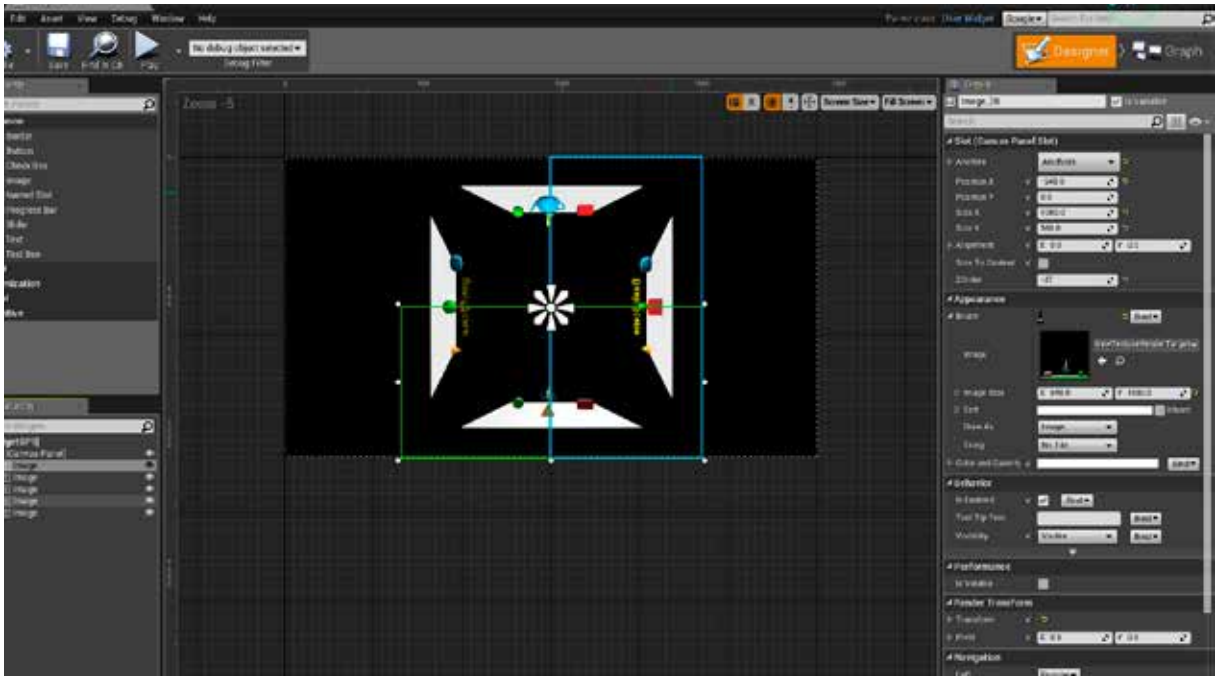

Abb. 35: *(BP1 Widget)Widget-Blueprint-Klasse zur Darstellung der vier Materialien, welche die unterschiedlichen Kameraperspektiven ausgeben*

Abb. 36: *Hinzufügen des erstellten Widgets zum Viewport des Benutzers der Anwendung*

Widget-Klasse (*BP1 Widget*) mit den darin enthaltenen vier Image-Komponenten eingetragen. Anschließend erfolgt die Ausgabe des Wigdets innerhalb des Sichtbereichs des Benutzers der Anwendung durch den Befehl *Add to Viewport*. Da nun die vier unterschiedlichen Anzeigen auf dem Monitor ausgegeben werden können, ist es möglich, bewegliche Kameras visuell zu implementieren. Zu diesem Zweck wurden die vier Kamera-Blueprints innerhalb des World Outliners mit einem unsichtbaren 3D-Objekt (CameraRotator) verbunden (Abb. 37). Auf diese Weise können bei der Programmierung alle Kameras mit einem einzelnem Objekt angesprochen werden. Da die in den verschiedenen SceneCapture2D-Blueprints enthaltene Kamera keinen Zielpunkt besitzt, ist eine Rotation der Kameras um einen bestimmten Punkt nur erschwert zu realisieren. Durch die Verknüpfung mit dem CameraRotator-Objekt, welches sich mittig zwischen den Kameras befindet, drehen sich bei der Rotation des Objekts alle Kameras gleichmäßig um den CameraRotator. Dementsprechend muss eine Kamerabewegung innerhalb des Level-Blueprints anhand der erstellten zusätzlichen Komponente erfolgen. Zu diesem Zweck werden zunächst zwei Variablen für die Rotation sowie die Auf- und Abbewegung des Hilfsobjekts vergeben. Anschließend sollen die Werte dieser Variablen mittels Tastatureingaben verändert werden (Abb 38). Wie in Abbildung 38 zu sehen ist, wird beim Druck auf die D-Taste die Variable *Rotation* auf den Wert 1 gesetzt. Beim Loslassen der Taste erfolgt eine Zuweisung auf den Wert 0. Dies geschieht ebenfalls bei Druck auf die A-Taste. Da die Kameras hierbei in die entgegengesetzte Richtung rotieren sollen, wird die Variable auf -1 gesetzt. Gleiches gilt für die Tastaturbelegung einer Auf- und Abwärtsbewegung des Hilfs-

Abb. 37: *Verknüpfung der einzelnen SceneCapture2D-Blueprints mit einem gemeinsamen 3D-Objekt (CameraRotator)*

objekts. Hierbei bewirkt der Druck auf die Tasten W und S des Keyboards jeweils eine Erhöhung oder Reduzierung der Variable *Position*. Um die Anwendung bei Ausführung beenden zu können, wurde an dieser Stelle ebenfalls die Escape-Taste des Keyboards mit dem Beenden der Anwendung belegt. Zur Auslösung einer Bewegung des CameraRotator-Objekts bzw. der einzelnen SceneCapture2D-Blueprints müssen die gesetzten Variablen mit der Rotation sowie Position des Hilfsobjekts in Verbindung gebracht werden (Abb. 39). Hierzu wird zunächst ein *Tick Event* benötigt. Dieses löst bei jedem gerenderten Bild (Frame) ein bestimmtes Ereignis aus. Um die in Abbildung 39 beschriebene Funktion zu verdeutlichen, erfolgt an dieser Stelle die Erklärung eines bestimmten Funktionsdurchlaufes. Bei Start der Anwendung übermittelt die Funktion *AddRelativeRotation* ihrem Zielobjekt (*Target*) den Rotationsgrad 0, da der Variable *Rotation* bei Start der Anwendung der Wert 0 zugewiesen wurde. Wie in Abbildung 39 zu sehen ist, ist unter dem Knotenpunkt *Target* der Funktion *AddRelativeRotation*, das Objekt *CameraRotator* eingetragen. Dementsprechend wird dem *CameraRotator*, welcher die Verbindung zu den Kameras darstellt, der Rotationsgrad 0 übergeben. Da dieser zunächst in jedem Frame 0 beträgt, bleiben

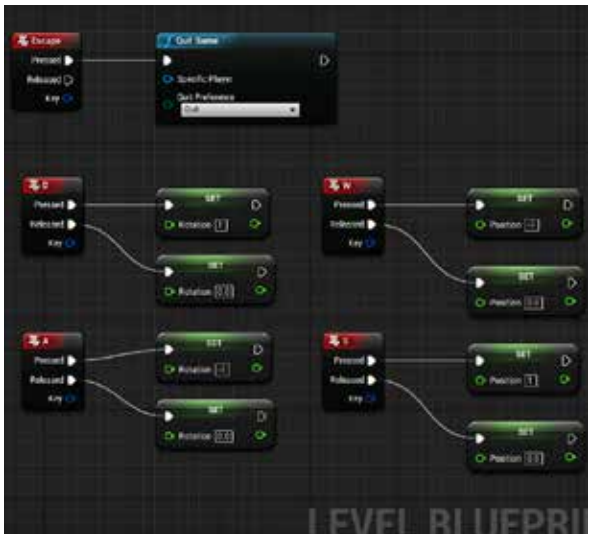

Abb. 38: *Tastaturbelegung für eine Bewegung der verschiedenen Kameras*

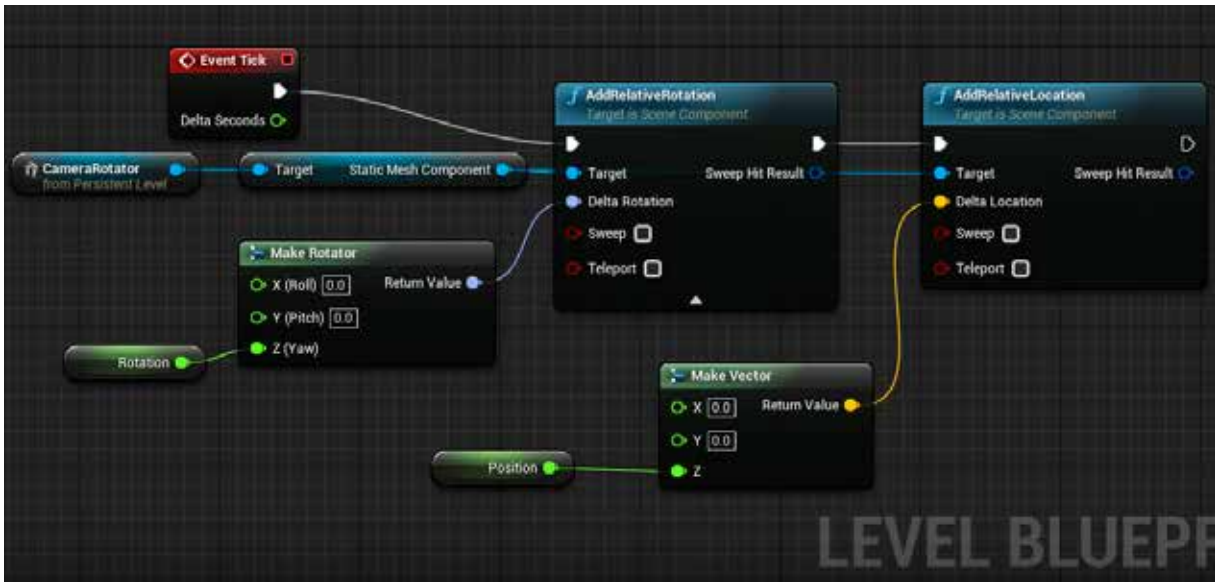

Abb. 39: *Veränderung der Kamerabewegung, durch Einsetzen verschiedener Variablen auf Tastendruck*

die Kameras auf dieser Winkelangabe stehen. Wird nun die D-Taste betätigt, wird die Variable *Rotation* mit dem Wert 1 überschrieben. Nun addiert die Funktion *AddRelativeRotation* den Wert 1 in jedem Frame auf den Z-Rotationsgrad des CameraRotator-Objekts. Wird die D-Taste wieder losgelassen, wird der Wert der Variable wieder durch 0 ersetzt. Dadurch wird auf den Rotationsgrad des CameraRotator-Objekts nun wieder der Wert 0 addiert. Dementsprechend bleiben die Kameras an dem zuletzt erreichten Rotationswert stehen. Wird die D-Taste bei einer Bildwiederholfrequenz von 60 Frames bspw. eine Sekunde lang betätigt, wurde der Rotationswert des *CameraRotators* 60 mal um ein Grad addiert und dementsprechend auf 60 Grad um seine Z-Achse gedreht. Der soeben beschriebene Vorgang kann ebenfalls auf die A-Taste übertragen werden. Allerdings wird hierbei die Variable *Rotation* auf -1 gesetzt. Eine Addition mit einem negativen Wert führt zu einer Reduzierung des Rotationsgrades des Hilfsobjekts. Gleiches gilt für die Position des *CameraRotators*. Diese steht zu Beginn der Anwendung auf jeweils 0 Einheiten in X, Y und Z. Wird nun eine Taste für die Positionsveränderung des Objekts betätigt, addiert sich der Wert der Variable *Position* in jedem Frame auf den Positionswert auf der Z-Achse des Objekts auf, wodurch die Kameras nach oben bewegt werden.
Neben dem Testen innerhalb des Editors, kann die erstellte Anwendung für verschiedene Formate exportiert werden. Bei dem sogenannten „Packaging“ werden alle notwendigen Dateien und Programmteile zusammengeführt, sodass ein Ordner ausgegeben werden kann, welcher die Anwendung enthält. Diese kann nun außerhalb des Editors gestartet werden, ohne dass es nötig ist, zusätzliche Software zu installieren.

3.7.3 Vor- und Nachteile der Unreal Engine 4

Ein Vorteil der Verwendung der Unreal Engine 4 ist, dass die Game-Engine völlig kostenfrei ist und je nach Verwendung keine weiteren Kosten entstehen. Darüber hinaus verfügt sie über einen integrierten Editor, welcher mit diversen Bestandteilen ausgerüstet ist, die dem Benutzer das Erstellen von Anwendungen enorm erleichtern. So müssen einzelne Komponenten nicht erst umständlich einprogrammiert werden, sondern stehen direkt nach dem Import visuell zur Verfügung. Zudem liefert der Editor bestimmte Grundelemente wie Lichter, Partikelsysteme oder primitive Objekte von vornherein. Durch die Verwendung des Editors gestaltet sich ein Szenen-Aufbau als sehr benutzerfreundlich und kann schnell erweitert werden. Darüber hinaus besteht die Möglichkeit, den Quellcode der Engine zu verändern. Dies könnte in der Form geschehen, dass eine Aufbereitung der Szene durch verschiedene Kameras und die Verwendung von Widgets überflüssig wird, wenn die Engine generell darauf ausge-

richtet ist, vier verschiedene Viewports auszugeben. Ein weiterer Vorteil ist die mögliche Verwendung von Blueprints. Mit ihrer Hilfe ist es nicht nötig bei der Umsetzung C++ Codezeilen zu schreiben. Daher muss keine entsprechende Programmiersyntax erlernt werden, was die Arbeit an Anwendungen erheblich vereinfacht. Vorteilhaft ist darüber hinaus, dass die erstellten Anwendungen für viele unterschiedliche Systeme ausgegeben werden können, wodurch die Anwendungen sehr flexibel einsetzbar sind. Ein Nachteil hingegen ist, dass für die Entwicklung das gesamte Paket der Engine inklusive des Editors installiert werden muss. Dadurch fallen relativ große Datenmengen an, die für die Entwicklung von recht kleinen Anwendungen nicht notwendig wären. Außerdem ist beim Export von HTML5-Anwendungen eine Anbindung an mögliche Erweiterungen, wie z.B. die Verwendung von Datenbanken mittels PHP nur erschwert zu realisieren. Der beim Packen des Projekts produzierte Code der Engine ist nur schwer nachzuvollziehen und stellt somit eine Hürde bei der Implementierung von Erweiterungen dar.

3.8 Gegenüberstellung der Lösungsmöglichkeiten

Wie aus den vorangegangenen Kapiteln ersichtlich ist, kann mit beiden Varianten eine interaktive Echtzeitanwendung zur Ausgabe auf der Plexiglaspyramide umgesetzt werden. Die Wahl der Entwicklungsumgebung hängt dabei stark vom Zweck der Anwendung ab. Generell bietet sich aber eine Umsetzung mit der UE4 an. Die Möglichkeit, die darzustellenden Objekte oder Szenen in einem Editor zusammensetzen zu können, ist gegenüber der Realisierung mit WebGL ein enormer Vorteil. Es können erst alle Elemente visuell angeordnet werden, um anschließend deren Wirkungsweise zu überprüfen. Mittels WebGL müssen zunächst viele Codezeilen geschrieben werden, um überhaupt eine Ausgabe zu ermöglichen. Ein Anpassen der Objekte in Position oder Verhalten erfordert sehr viel Zeit, da immer wieder Codezeilen angepasst werden müssen, wohingegen mit der UE4 die Objekte sichtbar verschoben werden können. Da es sich bei der Unreal Engine um eine vollwertige Spiele-Engine handelt, sind die möglichen Anwendungsumsetzungen enorm vielfältig. Soll bspw. ein Spiel für die Pyramide erstellt werden, ist dies mit der Engine wesentlich komfortabler als mit HTML5, CSS und Babylon.js. Um Objekte miteinander kollidieren zu lassen oder eine Interaktion mit einzelnen Objekten in der Szene zu realisieren, sind mit der Engine nur wenige Einstellungen zu verändern. Im Gegensatz dazu stellt sich die Implementierung von Physik oder Kollisionen mittels Babylon.js als wesentlich umständlicher dar. Da die UE4 eine Ausgabe der Anwendungen für diverse Plattformen anbietet und an dieser Stelle auch die Ausgabe für einen Webbrowser enthalten ist, wird die Verwendung

von HTML5, CSS3 und Babylon.js nahezu überflüssig. Allerdings ist es nur bedingt möglich, Änderungen an den von der Engine erstellten HTML5 und JavaScript Dateien vorzunehmen. Soll bspw. eine Anwendung erstellt werden, bei welcher dem Benutzer Interaktion mit den Objekten sowie eine Übermittlung von Daten an eine Datenbank ermöglicht werden soll, kann die Verwendung des Babylon.js-Frameworks in Betracht gezogen werden, da hierbei eine Datenbankanbindung mittels PHP wesentlich leichter zu realisieren wäre. Abschließend ist zu sagen, dass bei der Umsetzung zunächst der Zweck und Einsatzbereich der Anwendung im Vordergrund steht. Nachdem klar ist, welche Anforderungen an die Anwendung gestellt werden, kann die Auswahl der geeigneten Entwicklungsumgebung erfolgen.

3.9 Umsetzung von Spielen für das Pyramiden-System

Da es mittels der verwendeten Entwicklungsumgebungen möglich ist, interaktive Echtzeitanwendungen zu kreieren, kann eine Art der Umsetzung die Realisierung von Spielen für das Pyramiden-System sein. Zu diesem Zweck wurde ein Testspiel mittels des Babylon.js-Frameworks erstellt, bei welchem sich verschiedene Probleme äußerten. Ziel dieses Abschnitts ist es aufzuzeigen, für welche Art Spiel sich das Pyramiden-System eignen kann und an welchen Stellen die Verwendung des Pyramiden-Aufbaus gewisse Schwierigkeiten aufweist. Um die tatsächliche Nutzbarkeit des Pyramiden-Aufbaus in Bezug auf die Spiele-Entwicklung darlegen zu können, müssten an dieser Stelle weitere Versuche durchgeführt werden, was allerdings im Rahmen dieser Abhandlung nicht möglich ist.
Die Erstellung von Spielen für das System wird durch verschiedene Umstände erschwert. Um die mögliche Ansicht der Pyramide von vier verschiedenen Seiten innerhalb eines Spiels nutzbar zu machen, stand zunächst die Überlegung im Vordergrund, bei welchen Spielen eine Ansicht von mehreren Seiten von Vorteil sein kann. Bei einem Labyrinth, durch welches der Spieler einen Gegenstand steuern muss, kann die Spielfigur unter Umständen verdeckt sein, sobald sie sich hinter einer Wand befindet. In diesem Fall war eine Überlegung, dass der Spieler um den Aufbau herumgehen kann, um die Figur aus einem besser einsehbarem Winkel zu betrachten und so im Labyrinth weiter voranzukommen. Diese Art von logischer Spielweise ist durch den Versuchsaufbau allerdings stark eingeschränkt. Wie in Abbildung 40 dargestellt ist, wurde die Kamera oberhalb der Spielfläche positioniert. Dadurch hat der Betrachter zwar einen guten Blick auf das Spielgeschehen, allerdings wirkt die Spielfläche innerhalb der Pyramide verzerrt, sobald der Betrachter nicht den gleichen Blickwinkel wie die Kamera einnimmt. Es kann davon ausgegangen werden, dass die Pyramide in den

meisten Fällen frontal auf Höhe der Anzeige betrachtet wird, da die Wirkung der holographischen Illusion an dieser Position am besten zur Geltung kommt. Demzufolge müssten die Kameraperspektiven ebenfalls gerade auf die Szene ausgerichtet werden. Wie in Abbildung 41 zu beobachten ist, wird das Labyrinth dadurch zunehmend zweidimensional, eine Tiefenwirkung geht verloren und einzelne Wände überlagern sich, sodass eine Steuerung des Objekts innerhalb des Labyrinths nicht mehr möglich ist. Der Pyramiden-Aufbau birgt bei der Umsetzung von Spielen ein weiteres Problem. Im Beispiel des Labyrinths wird die Spielfigur (Kugel) mit den Tasten W, A, S und D des Keyboards gesteuert. Der Druck auf die W-Taste lässt die Kugel bei frontaler Betrachtung der Pyramide vom Spieler weg rollen. Bei Betätigung der A-Taste rollt die Kugel nach links. Sobald der Betrachter um 180 Grad um den Aufbau herum geht, um in eine andere Richtung zu steuern, müsste sich ebenfalls die Tastaturbelegung umdrehen. Wird nun die Taste W vom Spieler betätigt, rollt die Kugel auf den Beobachter zu anstatt wie erwartet von ihm weg. Dem Problem der Steuerung kann entgegen gewirkt werden, indem eine Steuerung verwendet wird, bei welcher sich die Spielfigur lediglich geradeaus bewegen kann. Um die Richtung zu wechseln, muss die Spielfigur um die eigene Achse gedreht werden, um sich dann in Richtung der Rotation der Figur fort zu bewegen. Dabei ist allerdings eine genaue Kenntnis des Rotationsgrades der Figur für den Spieler entscheidend, um die Spielfigur durch die Umgebung bewegen zu können. Auf Grund der genannten Probleme ist es nur erschwert möglich, Spiele zu entwickeln, welche die Ansicht von vier Seiten ausnutzen. Weniger problematisch sind Spielkonzepte, bei denen es keine direkte Verbindung zu einer dreidimensionalen Räumlichkeit mit den Objekten gibt. Ein Beispiel wäre das Einsammeln von frei schwebenden Objekten innerhalb der Pyramide. Durch das Freistellen der Elemente besteht kein direkter Bezug zu einer Grundfläche, wodurch ein dreidimensionaler

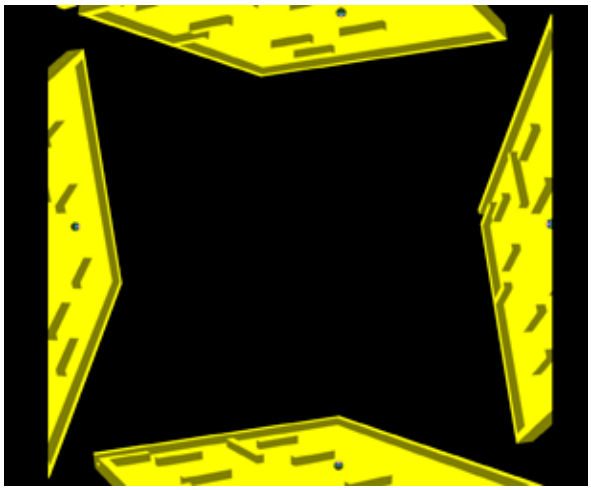

Abb. 40: *Labyrinthspiel mit einer Kameraansicht oberhalb der Spielfläche*

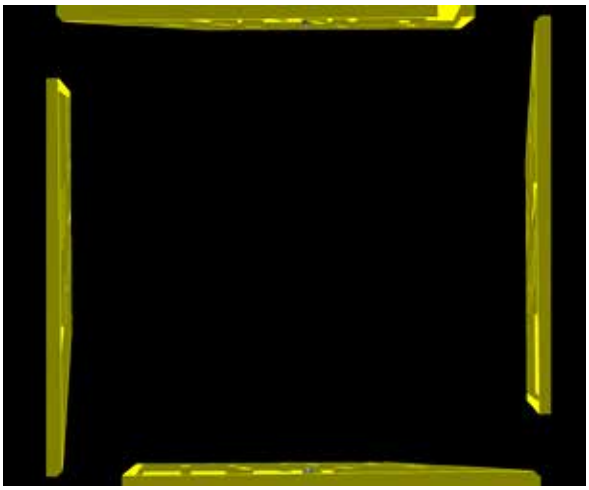

Abb. 41: *Labyrinthspiel mit einer Kameraansicht seitlich der Spielfläche*

Eindruck erhalten bleibt, auch wenn der Betrachter seine Perspektive verändert. Auf Grund der Steuerungsprobleme innerhalb der Pyramide ist es bei der Entwicklung von Spielen nahezu unmöglich, nicht für jede Anzeigefläche eine eigene Steuerung zu implementieren. Anhand dieser Überlegung kann es sinnvoll sein, Spiele für zwei oder mehr Spieler zu entwickeln. Hierbei wäre jeder Spieler bezüglich seiner Steuerung der Spielabläufe auf eine bestimmte Anzeigefläche beschränkt, wodurch es nicht zu Irritationen bei der Steuerung kommt. Fraglich ist dabei, in wie fern sich das Spielen auf dem Pyramiden-System von der Betrachtung eines normalen Monitors unterscheidet. Ein Vorteil kann dabei sein, dass es möglich ist, die Mitspieler durch die Anzeigefläche sehen zu können. Welche Vor- oder Nachteile sich daraus ergeben, muss in weiteren Testdurchläufen in Erfahrung gebracht werden. Eine detaillierte Untersuchung von Spiele-Umsetzungen für das Pyramiden-System führt innerhalb dieser Abhandlung jedoch zu weit.

3.10 Implementieren einer Bewegungssteuerung

Die Verwendung einer Tastatur oder eines Game-Controllers als Eingabemedium ist im Fall einer öffentlichen Ausstellung eines Pyramiden-Systems mit interaktivem Charakter nur bedingt nutzbar. Es ist zwar möglich, die dargestellten Objekte mittels Tastatureingaben zu drehen oder bestimmte Ereignisse auszulösen, allerdings ist der Umgang mit den Illusionen per Knopfdruck eher für Spiele anstatt für Visualisierungen auf dem System geeignet. Für den Fall, dass die Visualisierung als interaktives Element lediglich eine Drehung des Objekts darstellt, wäre das Auslösen einer Rotation per Knopfdruck eher mit dem Starten oder Stoppen eines Videos vergleichbar. Demzufolge wird der interaktive Anwendungscharakter in einem solchen Szenario nicht klar herausgestellt. Ebenso ist die Verwendung einer Computermaus als Eingabegerät nicht effektiv zu nutzen, da die Pyramide ein räumliches Abbild einer Szene darstellt und eine Maus lediglich zweidimensional arbeitet. Da die Verwendung von herkömmlichen Eingabegeräten für eine praktische Anwendung im Bereich der Produktvisualisierung nicht empfehlenswert ist, wurde in diesem Zusammenhang die Implementierung einer Bewegungssteuerung überprüft. Diese hat gegenüber anderen Eingabegeräten den Vorteil, dass die Steuerung einer Anwendung zunehmend vereinfacht wird und dadurch von einer weit gefächerten Zielgruppe verwendet werden kann. Besonders bei Produktvisualisierungen kann die Bewegungssteuerung einen schnellen Zugang zur dargestellten Anwendung ermöglichen. Darüber hinaus wird der holographische Effekt durch die Bewegungssteuerung hervor gehoben, da die händische Manipulation der gezeigten Objekte die Illusion von Berührungen mit der

Szenerie erzeugen kann. Zu Testzwecken wurde die Implementierung einer Bewegungssteuerung mittels der beschriebenen HTML5, CSS3 und Babylon.js Entwicklungsumgebung durchgeführt. Für die Aufnahme von Handbewegungen kommt dabei der Bewegungssensor Leap Motion des gleichnamigen Herstellers zum Einsatz (Leap Motion, Inc., 2016a). Der Bewegungssensor erkennt die Anzahl und Position von Händen, welche sich im Sichtbereich des Sensors befinden. Dabei wird neben der räumlichen Position der Hände ebenso die der einzelnen Gliedmaßen per USB an den angeschlossenen Computer übermittelt. Das Abtasten der Umgebung über dem Sensor geschieht 60 mal in der Sekunde, kann aber durch JavaScript-Anweisungen an die Anwendung angepasst werden. Die übermittelten Daten pro Abtastung werden wie bei dem Rendern von Bildern als Frames definiert (Leap Motion, Inc., 2016b). Innerhalb des Testdurchlaufs soll eine Rotation sowie Distanzierung der Kamera mittels der Bewegung einer Hand im Abtastbereich der Leap Motion erzielt werden. Um dem Grundaufbau der Entwicklungsumgebung für die Pyramiden-Ausgabe die Möglichkeit der Bewegungssteuerung von einzelnen Objekten hinzuzufügen, muss eine Zusatzbibliothek in das HTML5-Script eingebunden werden. Das sogenannte „Leap.js-Framework“ ermöglicht den Zugang zu den Aufzeichnungsdaten, welche von dem Bewegungssensor übermittelt werden. Da sich der Grundaufbau des Scripts nur unwesentlich von dem Aufbau der Beispielszene in Kapitel 3.6.2 unterscheidet, wird an dieser Stelle nur auf die wichtigsten Befehle zur Implementierung der Bewegungssteuerung eingegangen. Diese können mittels der Online-Dokumentation des Leap.js-Frameworks nachvollzogen werden (Leap Motion, Inc., 2016c). Das Leap.js-Framework enthält mehrere Funktionen, mit welchen eine Bewegungssteuerung eingebunden wird. Hierzu gehört unter anderem auch das Erkennen von bestimmten Gesten wie bspw. das Wischen mit der Hand. Um solche Gesten verwenden zu können, muss zunächst die Gestenerkennung aktiviert werden (*var controllerOptions = {enableGestures: true};*) (Abb. 42). Diese finden in dem in Abbildung 42 gezeigten Script-Block zwar keine Anwendung, illustrieren aber an dieser Stelle die Möglichkeiten des Frameworks. In einem weiteren Schritt wird eine Loop-Funktion implementiert (*Leap.loop(controllerOptions, function(frame){};*). Diese überprüft die vom Sensor aufgenommen Positionsdaten und Bewegungen der Hände mit 60 Frames in der Sekunde. Innerhalb der Loop-Funktion können die einzelnen Frames auf ihren Inhalt abgefragt werden. In einem ersten Schritt wird zunächst durch die Anweisung *frame.hands.length > 0* überprüft, ob sich eine Hand über dem Sensor befindet. Die Operation *> 0* registriert, ob mindestens eine Hand über dem Sensor liegt. Wird die Anweisung *frame.hands.lenght* auf *> 1* geprüft, werden die Operationen innerhalb der implementierten If-Anweisung nur dann ausgeführt, sobald sich mehr als eine Hand über dem Sensor befindet. Sobald eine Hand erkannt wurde, wird eine For-Schleife ausgeführt, welche den Zähler

i bis zum Wert 4 hoch zählt. Dies ist notwendig, da die für den Aufbau verwendeten Kameras innerhalb eines Arrays aufgebaut wurden. Dies vereinfacht das Script, da die Anweisungen bezüglich der Kamerabewegung nur einmalig implementiert werden müssen. Als nächstes erfolgt die Zuweisung der im Frame zuerst erkannten Hand zu der Variable *hand* mittels der Scriptzeile *var hand = frame.hands[0];*. Diese Variable enthält nun alle Positions- und Rotationsangaben der einzelnen Bestandteile der vom Sensor erkannten Hand. Um die Bewegung der Hand auf die Bewegung der Kamera zu übertragen, wird die Geschwindigkeit der Handfläche verwendet. Die Geschwindigkeit in X, Y und Z wird mittels der Funktion *hand.palmVelocity[];* ermittelt, wobei die Zahlen zwischen 0 und 3 innerhalb der eckigen Klammern die Bewegungsachse der Hand bestimmen. Anschließend wird die entsprechende Achsengeschwindigkeit der Handfläche in jeweils eine eigene Variable geschrieben. In einem nächsten Schritt erfolgt die Addition der Handgeschwindigkeit zu einer Kamerabewegung. Hierbei wird die Geschwindigkeit der Hand auf der X-Achse auf die Alpha-Rotation der Kamera übertragen (*cameras[i].alpha += Xvelocity*). Die Alpha-Rotation stellt dabei eine horizontale Drehung um den Fokuspunkt der Kamera dar. In gleicher Weise wird die Handbewegung auf der Y-Achse auf die Beta-Rotation der Kamera addiert. Der Radius der Kamera stellt die Entfernung zum Fokuspunkt dar. Dementsprechend wird die Z-Geschwindigkeit der Hand auf diesen Parameter addiert. Da die Geschwindigkeiten sowohl positiv als auch negativ werden können, löst bspw. eine Handbewegung nach rechts eine Addition auf den Alpha-Rotationswert der Kamera aus, wohingegen eine Bewegung nach links diesen Wert reduziert. Dadurch entsteht eine Hin- und Herbewegung der Kamera durch die Positionsveränderung der Hand. Damit die Kamera nicht zu nah an den Mittelpunk der Szene kommt, muss der Bewegungsradius der Kameras

```
var controllerOptions = {enableGestures: true};
Leap.loop(controllerOptions, function(frame) {

    if(frame.hands.length > 0)
        {
            for (i = 0; i < 4; ++i) {

                var hand = frame.hands[0];

                var Xvelocity = hand.palmVelocity[0]/3000;
                var Yvelocity = hand.palmVelocity[1]/3000;
                var Zvelocity = hand.palmVelocity[2]/20;

                cameras[i].alpha += Xvelocity;
                cameras[i].beta += Yvelocity;
                cameras[i].radius += Zvelocity;

            }
        }
});
```

Abb. 42: *Bewegungssteuerung mit dem Leap Motion-Sensor*

```
engine1.runRenderLoop(function () {

    for(i=0;i<4;i++)
    {

        pointhelpers[i].rotation.y -= 0.004;

        if (cameras[i].radius <= 400)
        {
            cameras[i].radius = 400;
        }

        if (cameras[i].radius >= 950)
        {
            cameras[i].radius = 950;
        }

scenes[i].render();

    }
});
```

Abb. 43: *Rendern der Szenen sowie Beschränken des Zoomfaktors der Kamera*

beschränkt werden. Dies erfolgt mittels der If-Abfrage *if(cameras[i].radius <= 400){cameras[i].radius = 400;}(Abb. 43)*. Eine Radius-Anpassung ist notwendig, um eine Beschneidung der Bildfläche zu vermeiden. Sobald die Kamera zu nah an ihren Fokuspunkt heranreicht, werden die Ränder der vom Bildschirm reflektierten Abbildung sichtbar, wodurch die Illusion eines frei schwebenden Objekts zerstört wird. Wie in Abbildung 43 zu sehen ist, erfolgt die Überprüfung des Kameraradius innerhalb der RenderLoop und wird daher bei jedem gerenderten Bild ausgeführt. Die Anweisungen der Radius-Beschränkung befindet sich zusätzlich innerhalb einer For-Schleife, um die Beschränkung für alle vier Kameras ausführen zu können. Neben der Ansprache der Handflächen können alle Teile der Hände mittels des Leap.js-Frameworks erfasst werden. So ist es bspw. möglich, die Position von 3D-Objekten auf die räumliche Position der Finger zu übertragen. Dadurch werden die 3D-Elemente bei einer Bewegung der Hand mitgeführt. Auf diese Weise können verschiedene Objekte über eine Kollisonsabfrage innerhalb der Szene gegriffen werden. Ein praktisches Beispiel wäre ein Puzzlespiel, bei welchem Boxen innerhalb der Plexiglaspyramide übereinander gestapelt werden.
Wie mittels dieses Abschnitts dargestellt wurde, ist die Umsetzung einer Bewegungssteuerung für die Pyramide leicht zu realisieren. Gleiches gilt für die Unreal Engine 4 als Entwicklungsumgebung. Hierzu stellt Leap Motion ein spezielles Plugin zur Verfügung, welches die vom Sensor aufgenommenen Daten an den UE4-Editor überträgt. Hierzu wurden allerdings keine weiteren Versuche durchgeführt.

3.11 Implementieren eines Head-Tracking-Systems

Im folgenden Kapitel soll ein weiteres Beispiel für die Vielseitigkeit der Realisation von Anwendungen mit der Unreal Engine 4 liefern. In diesem Abschnitt geht es nicht um die Einsetzbarkeit der dargelegten Umsetzung im professionellen Bereich, sondern soll lediglich ein Beispiel für die Verbindung von Head-Tracking mit einer Umsetzung für die Pyramide geben sowie die sich daraus ergebenden Vor- sowie Nachteile aufzeigen. In diesem Fall ist das Ziel der Verwendung von Head-Tracking, eine Bewegungspar-

allaxe bei der Betrachtung der Szene innerhalb der Pyramide zu erzeugen. Dadurch erscheint die dargestellte Szene innerhalb der Pyramide wesentlich räumlicher als bei einer einfachen Anzeige. Aufgabe des Head-Trackings ist es, die Bewegungen des Kopfes des Betrachters abzugreifen und auf die Bewegungen der Kamera innerhalb der Szene zu übertragen. Für die Umsetzung des Head-Trackings wird zunächst zusätzliche Hardware sowie Software benötigt. Ein wichtiger Teil des Versuchs war die Herstellung eines Infrarotsenders, welcher am Kopf des Beobachters der Darstellung zu befestigen ist (Abb. 44). Dieser besteht aus drei Infrarot-LEDs (IR-LEDs) sowie zwei Batterien, um mobil einsetzbar zu sein. Die IR-LEDs sind über Widerstände in einer speziell, von der verwendeten Tracking-Software vorgegebenen Position zueinander angeordnet. In welcher Form und wie viele IR-LEDs zum Einsatz kommen, kann durch die Software bestimmt werden. Um das von den IR-LEDs ausgesandte Licht einfangen zu können, wurde eine handelsübliche Webcam verwendet. Zur Filterung des Umgebungslichts dient ein IR-Filter, welcher vor der Kameralinse angebracht ist. Für die Analyse der einzelnen Infrarot-Lichtpunkte sowie deren räumlicher Lage und Rotation wurde die freie Tracking-Software OpenTrack 2.3 verwendet (Abb. 45). Innerhalb der Software können verschiedene Tracking-Modi ausgewählt werden. Im Fall der Versuchsdurchführung wird das Tracking von einem sogenannten „PointTracker“ übernommen. Für eine Anpassung an den verwendeten Infrarotsender können hierbei verschiedene Einstellungen vorgenommen werden. Ebenso ist es möglich, das von der Kamera eingefangene Bild zu justieren. Sobald die Kamera mit dem PC verbunden ist, kann das Tracking-Programm zur Aufzeichnung der empfangenen Daten gestartet werden. Innerhalb des Programms können die abgefangenen Daten an die räumlichen Gegebenheiten des Senders angepasst und eingestellt werden. So ist es bspw. möglich, verschiedene Rotationswinkel zu invertieren, um eine korrekte Datenausgabe zu erhalten. Als nächstes erfolgt die Überführung der gewonnenen

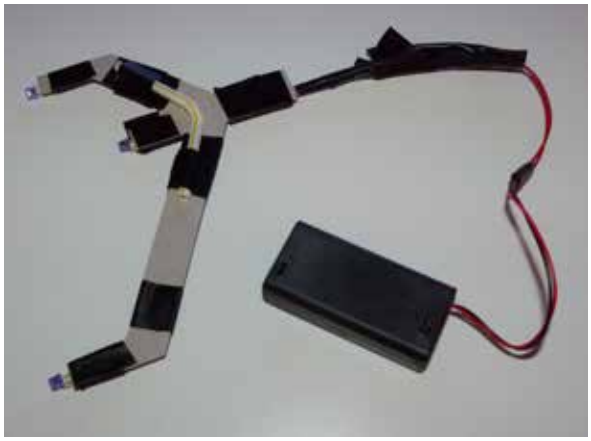

Abb. 44: *Infrarot-Sender bestehend aus drei IR-LEDs und Batterie*

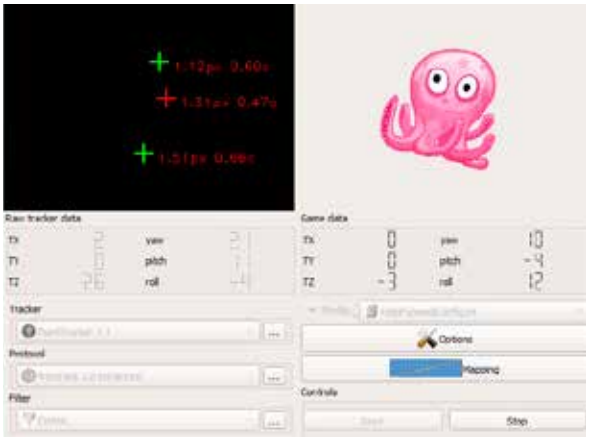

Abb. 45: *OpenTrack 2.3 mit empfangenen Tracking-Daten des Senders*

Lageinformationen des Kopfes in den Script-Teil der Anwendung. Zu diesem Zweck muss dem zu erstellenden UE4-Projekt ein zusätzliches Plugin hinzugefügt werden. Das Freetrack-Plugin wird hierbei in das Verzeichnis des erstellten Projekts kopiert. Nachdem dieser Schritt erfolgt ist, kann innerhalb des UE4-Editors auf spezielle Funktionen zugegriffen werden, welche die von dem Tracking-Programm gewonnenen Daten verarbeiten. Mit der Funktion *getFreetrackData* werden die Lageinformationen des Kopfes innerhalb des Level-Blueprints zugänglich gemacht (Abb. 46). Diese Funktion wird mittels eines *Event Ticks* in jedem neu gerenderten Frame ausgeführt, um eine Positionsveränderung aufzuzeichnen. Im Fall des Testversuchs spielt die Rotation des Kopfes des Betrachters keine Rolle. Für die Veränderung des Blickwinkels auf das darzustellende Objekt ist lediglich die Position des Kopfes entscheidend. Wird die Szene von oben betrachtet, muss die Kamera entsprechend von oben auf die dargestellte Szene blicken, um einen dreidimensionalen Effekt zu erreichen. Eine Betrachtung von der Seite führt folglich zu einer Rotation der Kamera auf horizontaler Ebene. Dementsprechend muss die räumliche Lage des Kopfes auf die Rotation der Kamera bzw. des Kamera-Hilfsobjekts (CameraRotator) übertragen werden. Zu diesem Zweck wird der vom Tracking-Programm übertragene Positionsvektor (*Translation*) in drei verschiedene Werte für die X-, Y- und Z-Lage des Kopfes aufgeteilt (*Break Vector*). Hierbei spielt die Positionierung des Kopfes auf der Tiefenebene (in diesem Fall X) keine Rolle. Würde diese mit berücksichtigt, würde das Objekt bei einem Heranfahren des Kopfes in Richtung des Aufbaus zu einer Vergrößerung des Objekts führen. Da das 3D-Element allerdings ein reales Abbild eines Gegenstands darstellen soll, würde dieses Verhalten zu Irritationen auf Seiten des Betrachters führen. In einem weiteren Schritt werden die empfangenen Positionsdaten in Rotationsdaten für den CameraRotator umgerechnet (Abb. 47). Bei der Verknüpfung der Daten zwischen Tracking-Software und Kamera kommt es darauf an, in welcher Ausrichtung die verschiedenen Koordinatensysteme liegen. Dementsprechend müssen zusätzliche Rechenoperationen und Umwandlungen der empfangenen Daten erfolgen, um eine korrekte Kamerabewegung zu erzeugen. Anschließend werden die Daten auf die Rotation des Kamera-Hilfsobjekts mit der

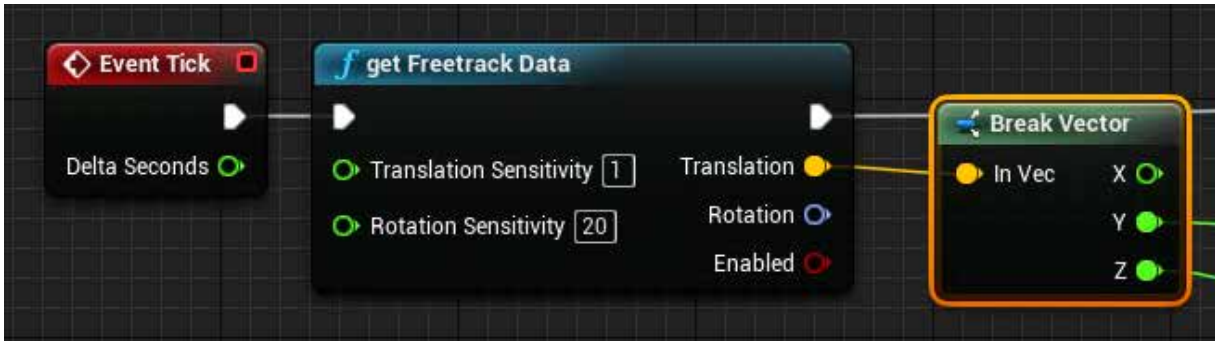

Abb. 46: *Einbinden der von der Tracking-Software übermittelten Daten*

Funktion *SetRelativeLocationAndRotation* übertragen.
Das Head-Tracking für den Pyramiden-Aufbau ist, wie durch den Testdurchlauf dargelegt wurde, relativ einfach umzusetzen. Es bedarf allerdings einiger Einstellungen, um eine korrekte Abbildung des dreidimensionalen Raumes der dargestellten Szene anzeigen zu können. Durch das Einbinden von Kopfbewegungen kann der dreidimensionale Eindruck der holographischen Darstellung gesteigert werden, da es somit theoretisch möglich ist, ein Objekt in einer 360 Grad-Ansicht darzustellen anstatt nur aus vier verschiedenen Winkeln. Innerhalb des Versuchsaufbaus wurde für die Darstellung lediglich eine Anzeigefläche verwendet. Für das Bespielen von allen vier Flächen müssten mehrere Kameras verwendet werden. Hierbei ist fraglich, ob eine solche Umsetzung überhaupt realisierbar ist. Auf Grund der verwendeten Hardware war es nicht möglich, ein korrektes Tracking aufzubauen. Die Bewegungen des Kopfes wurden zwar auf die Kamera und somit auf die Szene übertragen, allerdings führten schnelle Bewegungen auf Grund des eingeschränkten Sichtfeldes der Kamera und einer zu schwachen Lichtemission der Infrarot-LEDs, zu Übertragungsfehlern der Tracking-Daten. Abschließend kann gesagt werden, dass ein Head-Tracking in einzelnen Fällen sinnvoll sein kann. Allerdings muss für die Betrachtung ein entsprechender Aufsatz getragen werden. Darüber hinaus kann die Anzeige nur für eine Person pro Fläche korrekt dargestellt werden. Wenn ein pyramidaler Aufbau für Messezwecke und somit für ein breites Publikum eingesetzt werden soll, ist die Verwendung von Head-Tracking in diesem Zusammenhang eher hinderlich als dass der gewonnene 3D-Effekt eine Bereicherung für das System darstellt.

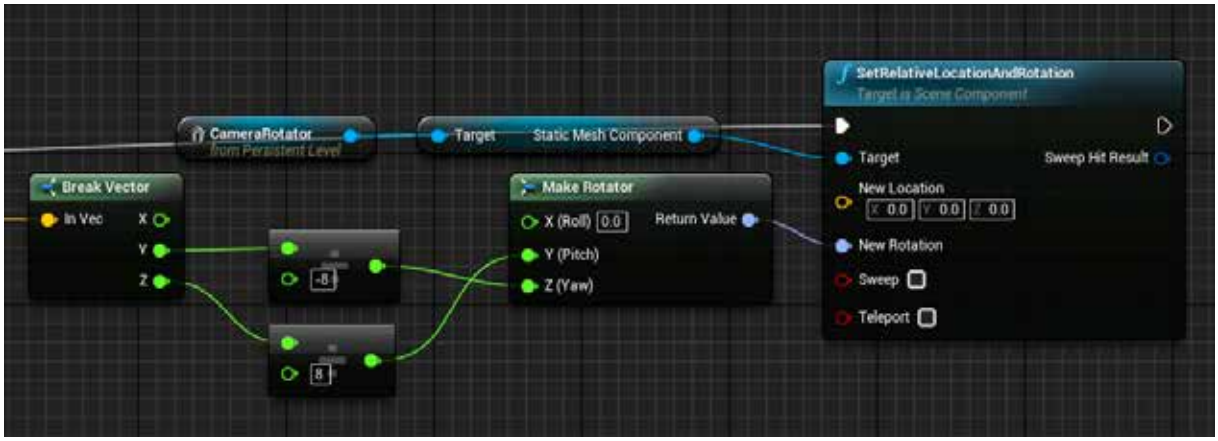

Abb. 47: *Umrechnung der empfangenen Daten auf die Bewegung des CameraRotators*

3.12 Mögliche Einsatzbereiche

Anhand der Resultate, welche sich durch die Umsetzung ergeben haben, können Vermutungen über mögliche Einsatzbereiche von interaktiven Echtzeitanwendungen auf holographischen Pyramiden-Displays angestellt werden. Da die Analyse von Einsatzbereichen sehr umfassend ist und hierzu weitere Tests und Ausarbeitungen stattfinden müssen, erfolgt an dieser Stelle lediglich eine grobe Einschätzung von Einsatzbereichen.
Ein Anwendungsfeld könnte die Nutzung von interaktiven Anwendungen zu Präsentationszwecken von Produkten sein. Holographische Pyramiden kommen bereits seit längerer Zeit bspw. auf Messen zur Produktvisualisierung zum Tragen. Hierbei wird meist ein realer Gegenstand in der Mitte der Pyramide platziert und von einem Video umspielt (Magic Holo 2015a). Mit der Verwendung von interaktiven Lösungen werden an dieser Stelle neue Formen der Produktvisualisierung möglich. Ein Beispiel wäre die Präsentation eines neuen Mobiltelefons. Während das reale Objekt in mitten der Pyramide liegt, könnte es den Beobachtern möglich sein, mittels einer Wischbewegung durch holographisch dargestellte Menüs zu scrollen ohne das Handy zu berühren. So entsteht eine Interaktion mit einem einzelnen Gegenstand, welcher allerdings gleichzeitig von mehreren Beobachtern von vier Ansichten gesehen werden kann. Hierbei dient der pyramidale Aufbau hauptsächlich als Blickfang. Die Interaktion mit der Illusion sowie die Illusion selbst könnte für eine gesteigerte Immersion beim Betrachter sorgen, welcher hierdurch ein positives Erlebnis mit der Marke erfährt. Durch die verwendete Entwicklungsumgebung könnten sich hierbei ebenso Rückschlüsse auf das Verhalten der Rezipienten beim Umgang mit der Produktvisualisierung ergeben. Beispielsweise ist es möglich, die Bewegungen bzw. die Länge der betrachteten Menüs des Mobiltelefons aufzuzeichnen, um so einen Verständnis für das Nutzungsverhalten des Rezipienten zu erhalten. Ein weiteres Anwendungsfeld könnte die dreidimensionale Darstellung von Objekten zu Bildungszwecken darstellen. Da es möglich ist 3D-Objekte in allen denkbaren Formen anzuzeigen und über verschiedene Eingabegeräte zu manipulieren, können schwer verständliche Themen veranschaulicht werden. Ein Beispiel hierfür wäre die Präsentation von Elementen auf molekularer Ebene. Hierbei können die Moleküle von verschiedenen Seiten betrachtet werden, um deren Aufbau deutlich zu machen. Darüber hinaus wäre es bspw. möglich, einen Editor für Moleküle zu entwickeln. Hierbei könnten mittels Bewegungssteuerung einzelne Moleküle aus verschiedenen Grundbausteinen nachgebaut werden. Durch die Interaktion mit den Elementen könnte das Verständnis für die Materie verbessert und Gruppenlernziele erreicht werden. Ein drittes Einsatzfeld wäre im Bereich der Video- bzw. Computerspiele denkbar. Durch die Ansicht von vier Seiten könnten sich völlig neue Spielprin-

zipien entwickeln lassen, welche die Räumlichkeit der Objekte anders ausnutzen als dies bei Spielen auf herkömmlichen Bildschirmen der Fall ist. Da sich die Spieler durch die Pyramide hindurch anschauen können, könnten neue Konzepte in Bezug auf die Interaktion mit dem Spiel aber auch der Spieler untereinander entwickelt werden.

4 Verbesserung der Bilddarstellung

Die Qualität des reflektierten Bildes innerhalb des Versuchsaufbaus ist von verschiedenen Einflussgrößen abhängig. In dem vorliegenden Kapitel sollen Möglichkeiten aufgezeigt werden, in welcher Form sich die optische Wiedergabe von Darstellungen auf der Pyramide verbessern lässt. Grundlegend sind für die optische Qualität zwei Faktoren verantwortlich, welche teilweise beeinflusst werden können. Als erstes ist die Qualität des verwendeten Displays zu nennen. Verschiedene Technologien verfügen über unterschiedliche Vor- und Nachteile bei der Verwendung als Anzeigefläche für einen Pyramiden-Aufbau. Die verschiedenen Technologien werden im Abschnitt 4.2 „Theoretische Verwendung verschiedener Monitortechnologien“ weiter erläutert. Da für den Versuchsaufbau lediglich eine der möglichen Technologien zur Verfügung stand, erfolgt eine Analyse der verschiedenen Techniken auf einer rein theoretischen Basis. Der zweite Faktor ist das Plexiglas der Pyramide. Wie bereits beschrieben wird die Abbildung auf dem Plexiglas durch verschiedene Umstände wie bspw. das Umgebungslicht oder die Doppelreflexion auf der Glasoberfläche beeinträchtigt. In welcher Art und Weise diesen Beeinträchtigungen entgegengewirkt werden kann und welche Folgen daraus resultieren, wird in dem folgenden Abschnitt „Beschichtung des Plexiglases“ näher erläutert.

4.1 Beschichtung des Plexiglases

Um die Qualität des reflektierten Bildes zu verbessern, wurde die Beschichtung eines Plexiglas-Teststreifens mit verschiedenen Folien durchgeführt. Für einen Vergleich der unterschiedlichen Folien sowie deren Wirkungsweise wurden zwecks transparenter Dokumentation, Bilder des Aufbaus sowohl vor als auch nach der Beschichtung aufgenommen. In Abbildung 48 ist der Versuchsaufbau ohne Beschichtung bei leicht abgedunkeltem Tageslicht zu sehen. Wie auf diesem Bild zu erkennen ist, ist der Eindruck des reflektierten Bildes sehr transparent. Die Objekte innerhalb der Szene sind nur oberflächlich wahrzunehmen. Zum einen liegt dies daran, dass das Plexiglas nur einen gewissen Anteil des einfallenden Lichts reflektiert. Zum anderen wird neben dem

Licht des Displays ebenfalls das Umgebungslicht auf der Glasfläche gespiegelt. Darüber hinaus ist der Hintergrund der Pyramide sehr gut sichtbar, wodurch sich die Reflexion mit dem Hintergrund überblendet. In Abbildung 49 ist die Pyramide bei geringem Umgebungslicht zu sehen. Hierbei wird deutlich, dass die Lichtstärke des reflektierten Lichts bei geringerer Umgebungsbeleuchtung enorm zunimmt. Da der Hintergrund der Pyramide wesentlich dunkler ist als in Abbildung 48, treten die dargestellten Objekte deutlicher heraus und heben sich vom Hintergrund ab. Darüber hinaus wird die Umgebung weniger stark auf dem Plexiglas reflektiert. Daraus folgt, dass die Qualität der Abbildung gesteigert werden kann, wenn die Reflexions-Eigenschaften des Glases verändert werden und der Hintergrund der Pyramide abgedunkelt wird. Um die Qualität des reflektierten Bildes zu steigern, wurden Folien mit verschiedenen Transmissions- und Reflexionswerten getestet. Da für das verwendete Plexiglas keine Reflexions- sowie Transmissionswerte vorliegen, dienen hierbei zum Vergleich die Angaben des Chemiekonzerns Arkema. Laut Arkema hat reines Plexiglas bei einem senkrechten Lichteinfall eine Lichtdurchlässigkeit von ca. 90 Prozent, wohingegen ca. 5 Prozent des Lichts reflektiert werden (Arkema, Inc. 2000, S. 1). Diese Angaben dienen lediglich der groben Einschätzung und dem Vergleich

Abb. 48: *Photographie des Pyramiden-Aufbaus ohne Beschichtung bei leicht abgedunkeltem Tageslicht*

Abb. 49: *Photographie des Pyramiden-Aufbaus ohne Beschichtung bei geringem Umgebungslicht*

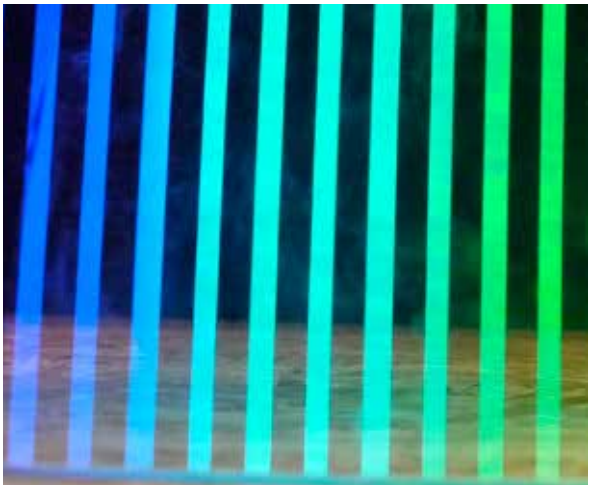

Abb. 50: *Reflektiertes Testbild ohne Beschichtung des Plexiglases*

der verwendeten Folien. Abbildung 50 stellt eine Referenz zu den weiteren Beschichtungen dar und zeigt den Plexiglas-Teststreifen ohne eine Beschichtung der Oberfläche. In Abbildung 51 wurde die Folie *Quantum 14* von Bruxafol auf das Glas aufgezogen (Bruxafol 2015a). Diese Folie verfügt über eine sehr geringe Lichttransmission. Dementsprechend wird nur wenig Licht durch die Folie hindurch gelassen, wodurch der Hintergrund nur noch sehr schlecht zu erkennen ist. Wie man in Abbildung 51 aber sehr deutlich erkennen kann, ist das reflektierte Bild des Monitors besser zu sehen als ohne Beschichtung in Abbildung 50, da sich der Hintergrund weniger deutlich abzeichnet (Abb. 50 und Abb. 51). Da die Lichtdurchlässigkeit von nur 13 Prozent (Bruxafol 2015a) sehr gering ist, wird der Eindruck einer holographischen Illusion entsprechend behindert. Die Wirkung dieser Folie entspricht eher der eines schwarzen Spiegels. Wie in Abbildung 52 zu sehen ist, wurde die Lichtdurchlässigkeit im Vergleich zu einer Beschichtung mit der Folie *Quantum 14* um 6 Prozent erhöht (Bruxafol 2015b). Diese vergleichsweise geringe Erhöhung ist jedoch optisch bereits gut zu erkennen. Das Abbild wirkt zunehmend transparenter bei fast gleichbleibender Reflexionsstärke. Durch die Beschaffenheit der Folie, wirkt das reflektierte Bild jedoch leicht verwaschen, was vermutlich auf Unterschiede in der

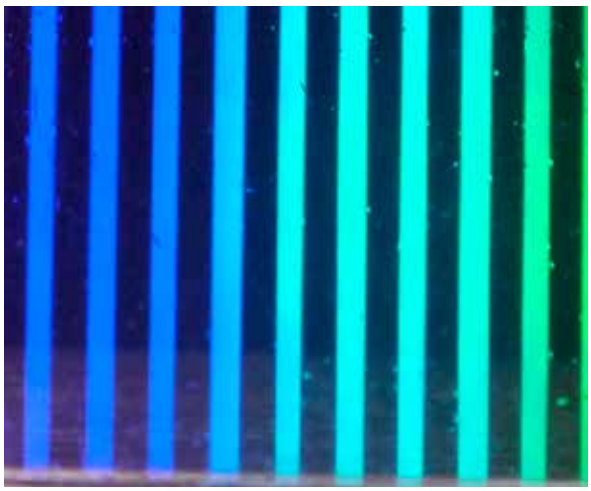

Abb. 51: *Beschichtung mit Quantum 14, Transmission: 13%, Reflexion: 17%, Absorption: 70%*

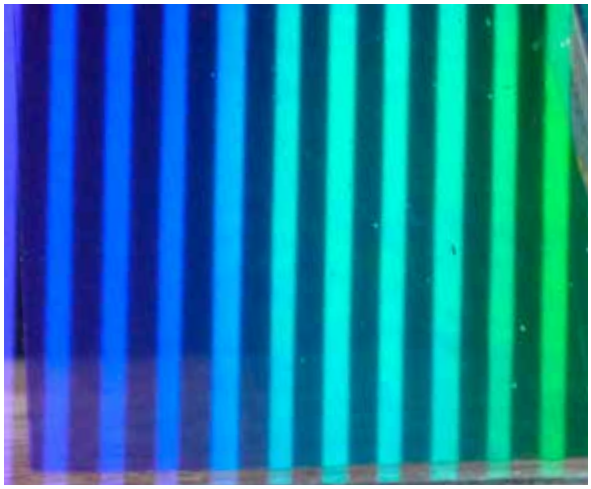

Abb. 52: *Beschichtung mit Quantum 15, Transmission: 19%, Reflexion: 14%, Absorption: 67%*

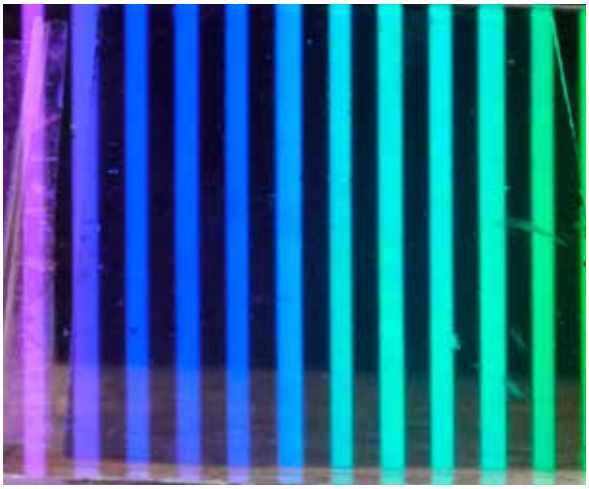

Abb. 53: *Beschichtung mit Quantum 16, Transmission: 32%, Reflexion: 11%, Absorption: 57%*

Herstellung zurückzuführen ist. Bei dem Test mit der Folie *Quantum 16* liegt gegenüber *Quantum 15* eine Steigerung der Transmission von 13 Prozent vor, wohingegen die Reflexion nur leicht abgenommen hat (Bruxafol 2015c) (Abb. 53). Gegenüber der Folie *Quantum 15* ist eine Durchsicht auf den Hintergrund deutlich zu erkennen, wobei die Lichtstärke des Bildes nahezu gleich geblieben ist. Dadurch wirkt die Spiegelung wesentlich holographischer als bei *Quantum 14*. Die in Abbildung 54 zu sehende Beschichtung verfügt über einen Transmissionswert von 50 Prozent. Dementsprechend wird die Hälfte des einfallenden Lichts hindurch gelassen. Die Reflexion der Fläche ist mit 8 Prozent sehr gering und nahezu gleich der Reflexion des reinen Plexiglases (Bruxafol 2015d). Die Reflexion ist durch eine geringere Lichtdurchlässigkeit zwar deutlich zu sehen aber im Vergleich zu der Folie *Quantum 16* nicht so lichtstark. Da die optischen Ergebnisse im Vergleich zum reinem Plexiglas bei der Folie *Quantum 16* am besten waren, wurde nach dem Test die gesamte Pyramide mit dieser Folie bezogen. Wie in Abbildung 55 und 56 zu sehen ist, wird die Qualität des reflektierten Bildes durch die Beschichtung mit *Quantum 16* enorm gesteigert, wobei eine Durchsicht durch die Pyramide weiterhin gut möglich ist. Durch die Beschichtung wurde darüber hinaus die Doppelreflexion erheblich re-

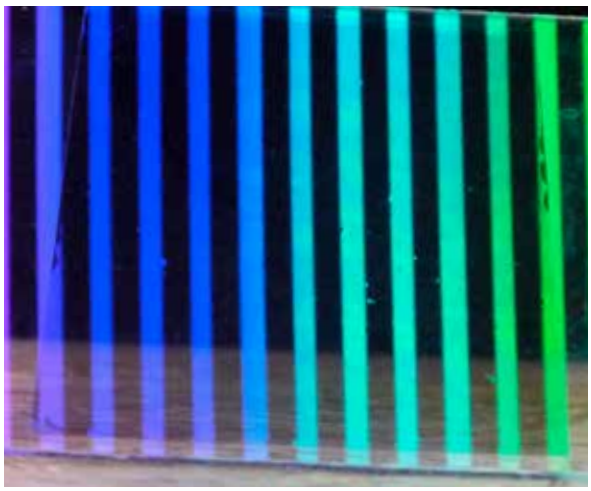

Abb. 54: *Beschichtung mit Quantum 25, Transmission: 50%, Reflexion: 8%, Absorption: 39%*

Abb. 55: *Photographie des Pyramiden-Aufbaus ohne Beschichtung des Plexiglases*

Abb. 56: *Photographie des Pyramiden-Aufbaus mit Beschichtung des Plexiglases mit Quantum 16*

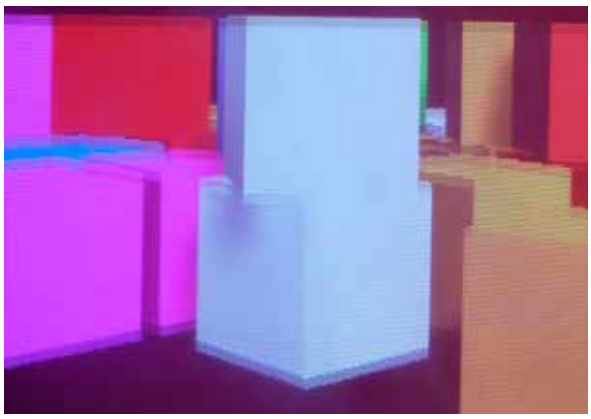

Abb. 57: *Starke Doppelreflexion ohne Beschichtung des Glases*

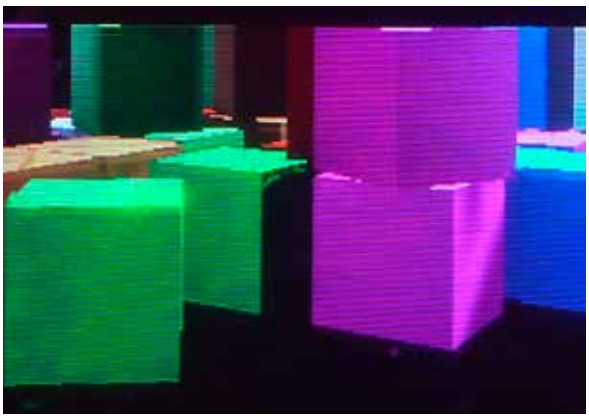

Abb. 58: *Reduzierte Doppelreflexion mit Beschichtung des Glases*

duziert. Gut zu erkennen ist die Reduzierung bei dem Vergleich zwischen Abbildung 57 und Abbildung 58. Da weniger Licht durch die obere Seite des Plexiglases hindurch gelassen wird, fällt eine erneute Reflexion an der Unterseite des Plexiglases deutlich geringer aus, wodurch das Abbild wesentlich schärfer dargestellt wird.

4.2 Theoretische Verwendung verschiedener Monitortechnologien

Theoretisch kann der Versuchsaufbau mit jeder Art von Monitor oder Display realisiert werden. Allerdings ist das Gewicht des verwendeten Bildschirms ein entscheidender Faktor, da der Display oberhalb der Plexiglaspyramide angebracht ist und durch ein höheres Gewicht eine stärkere Unterkonstruktion benötigt wird. Darüber hinaus verfügen verschiedene Technologien über unterschiedliche Eigenschaften der Bilddarstellung. Innerhalb dieses Abschnitts soll dargelegt werden, welcher Monitortyp für die Darstellung von Bildinhalten auf der Pyramide am besten geeignet ist und welche Faktoren über die Qualität des reflektierten Bildes entscheidend sind.
Eines der Hauptentscheidungskriterien ist das Kontrastverhältnis des Displays. Dieses gibt Auskunft über das Verhältnis von möglicher maximaler sowie minimaler Helligkeit des Monitors (Böhringer et al. 2014, S. 486). Das Kontrastverhältnis ist ein entscheidender Indikator für die Bildqualität bzw. die Reflexion auf der Plexiglaspyramide. Ist das Kontrastverhältnis des Displays zu gering, geben ebenfalls schwarze Bildbereiche Licht ab, was zu Reflexionen auf der Plexiglaspyramide führt, wodurch der holographische Effekt beeinträchtigt wird. Diese Reflexionen führen dazu, dass die Illusion verloren geht und der Betrachter nicht den Eindruck eines freistehenden Objekts hat, sondern die Bildspiegelung dunkler Pixel als Reflexion einer Bildfläche wahrnimmt. Da bei der Bildspiegelung ein Teil des Lichts transmittiert wird, ist das Bild auf dem Plexig-

las abgeschwächt. Um dennoch ein gut sichtbares Abbild zu erhalten, müssen helle Bildbereiche dementsprechend lichtstark sein, um der Transmission entgegen zu wirken. Der mögliche Betrachtungswinkel des Bildschirms spielt auf Grund des Aufbaus keine Rolle. Da das Bild senkrecht auf dem Plexiglas gespiegelt wird, kann die holographische Illusion auch mit einem kleinen Betrachtungswinkel des Monitors erzeugt werden. Der für den Versuchsaufbau verwendete Monitor ist ein Siemens SyncMaster LCD TFT-Display mit einer Auflösung von 1280 x 1024 Pixeln und einem Kontrastverhältnis von 700 zu 1. Im Vergleich zu CRT-Displays (Cathode Ray Tube), welche ein Kontrastverhältnis von 2000 zu 1 erreichen können (Schmidt 2013, S. 481), ist dieser Wert relativ gering und damit als Wiedergabemedium nicht ideal. Um eine theoretische Technologieauswahl zu ermöglichen, mit welcher sich die Abbildung verbessern lässt, folgt an dieser Stelle zunächst eine knappe Erklärung einzelner Monitortechnologien. An Hand der dargelegten Vor- sowie Nachteile der verschiedenen Techniken findet eine theoretische Technologieauswahl am Ende dieses Abschnitts statt.

4.2.1 Plasmadisplays

Plasmadisplays erzeugen die Darstellung bunter Pixel durch Gas, welches durch elektrische Spannung zu Plasma gewandelt wird, wodurch dieses UV-Strahlung abgeben kann. Jedes Pixel besteht hierbei aus drei verschiedenen Gaskammern, die mit leuchtenden Phosphoren in den Grundfarben Rot, Grün und Blau beschichtet sind. Bei der Entzündung des Gases regt die UV-Strahlung das Phosphor zur Abgabe von Licht in der entsprechenden Farbe an. Da hierbei die einzelnen Bildpunkte selbstständig Licht abgeben, kann ein hohes Kontrastverhältnis erzielt werden, da keine dauerhafte Hintergrundbeleuchtung benötigt wird. Darüber hinaus ist eine flache Bauweise und die Produktion von sehr großen Bildflächen möglich. Ein Nachteil gegenüber der im nächsten Abschnitt erläuterten LCD-Technik ist, dass sich das Bild bei statischen Bildinhalten einbrennen kann und somit weitere Bilddarstellungen verschlechtert werden. Des Weiteren wird zur Entzündung des Gases eine relativ hohe Spannung benötigt, woraus ein höherer Stromverbrauch im Vergleich zu LCDs resultiert. Darüber hinaus entsteht im Unterschied zu anderen Technologien durch die Entzündung des Gases und die angelegte Stromspannung eine höhere Wärmeentwicklung, was je nach Gerät eine zusätzliche Kühlung der Bauteile erforderlich macht (Schmidt 2013, S. 477 - 479).

4.2.2 LCD und LED-LCD

Sogenannte „LCDs“ (Liquid Christal Display) arbeiten mit Polarisationsfiltern, welche Licht in bestimmten Schwingungsebenen herausfiltern können. Hierbei werden zwei Polarisationsfilter mit einer Drehung von 90 Grad übereinander angeordnet. Durch diesen Aufbau wird gleichschwingendes Licht einer Hintergrundbeleuchtung durch den ersten Filter hindurch gelassen, aber durch die Drehung von 90 Grad durch den weiteren Polarisationsfilter absorbiert. Zwischen den beiden Filtern befindet sich eine Schicht aus Flüssigkristallen, welche bei Anlegen von elektrischer Spannung ihre Ausrichtung derart verändern, dass einfallendes Licht in dessen Schwingungsebene um 90 Grad verdreht werden kann, sodass die Lichtstrahlen durch beide Filter hindurch gelassen werden. Dementsprechend wird bei einem solchen System von einem passiven Monitor gesprochen, da die einzelnen Pixel nicht selbst Licht emittieren, sondern eine dauerhafte Hintergrundbeleuchtung abdunkeln oder hindurch lassen (Schmidt 2013, S. 470). Bei LCDs besteht ein Pixel aus jeweils einem roten, grünen und blauen Subpixel mit einem entsprechenden Farbfilter. Jedes Subpixel verfügt über einen sogenannten „TFT“ (Thin Film Transistor), welcher für das Anlegen von elektrischer Spannung an den Flüssigkristallen der einzelnen Subpixel sorgt (Welsch / Liebmann 2012, S. 337). Bei älteren LCD-Modellen kommen für die Hintergrundbeleuchtung sogenannte „Cold Cathode Fluorescent Lamps“ (CCFL) zum Einsatz. Diese sind am Rand der Bildfläche angebracht und verteilen das Licht über diffuse Kunststoffflächen hinter den Polarisationsfiltern. Auf Grund der konstanten Hintergrundbeleuchtung besitzen auch abgedunkelte Pixel einen gewissen Grad an Helligkeit, wodurch die Qualität von Kontrast und schwarzen Bildinhalten verschlechtert wird. Bei neueren LCD-Geräten kommt bei der Hintergrundbeleuchtung ein Gitter aus LEDs zum Einsatz. Diese haben den Vorteil, dass sie für dunklere Bildbereiche geschwächt oder abgeschaltet werden können, so ergeben sich höhere Kontrast- und Schwarzwerte bei der Bildwiedergabe. Darüber hinaus zeichnen sich LCD-Monitore durch einen sehr geringen Stromverbrauch aus (Schmidt 2013, S. 470 - 477).

4.2.3 OLED

OLED-Displays (Organic Light Emitting Diode) stellen eine Erweiterung der LED-Technologie dar. Im Gegensatz zu LCD oder LED-LCD-Monitoren wird bei dieser Technik keine Hintergrundbeleuchtung benötigt, da die Bildwiedergabe durch eigenständig leuchtende Dioden realisiert wird. Aus diesem Grund handelt es sich bei dieser Monitortechnologie um ein aktives Display-System. Eine OLE-Diode besteht aus zwei

Elektroden, zwischen welchen eine organische Schicht liegt. Beim Anlegen von elektrischer Spannung zwischen den Elektroden werden positive sowie negative Ladungen an die jeweilige Gegenelektrode ausgesandt. Das Anlegen der elektrischen Ladung erfolgt ähnlich der LCD-Technik durch eine zeilenweise Adressierung der einzelnen Bildpunkte. Beim Aufeinandertreffen der Ladungen innerhalb der organischen Schicht entsteht Energie in Form einer elektromagnetischen Welle, welche von dem Material abgestrahlt wird. Die für bunte Pixel benötigten Grundfarben können durch die Wahl des in der OLED verwendeten, organischen Materials erzielt werden. Durch einen sehr flachen Aufbau der einzelnen OLED-Schichten ist es möglich, Displays mit einer sehr geringen Tiefe im Millimeterbereich herzustellen. Da die einzelnen Dioden selbstleuchtend sind und komplett dunkel geschaltet werden können, kann mit der OLED-Technologie ein sehr hohes Kontrastverhältnis zwischen hellen und dunklen Bildpunkten erreicht werden (Schmidt 2013, S. 479 - 481). Darüber hinaus sind OLED-Displays durch die verwendete Technik im Vergleich zu anderen Technologien sehr stromsparend und daher auch für den mobilen Einsatz gut geeignet. Ein großer Nachteil ist allerdings die geringere Lebensdauer der Leuchtdioden, da diese nach einer gewissen Zeit an Leuchtkraft verlieren. Darüber hinaus ist die verwendete Technik anfällig gegen äußere Einflüsse wie Sauerstoff und Wasser, wodurch eine sorgfältige Verkapselung der einzelnen Leuchtzellen notwendig wird, was wiederum vergleichsweise hohe Produktionskosten nach sich zieht (Heinecke 2012, S. 151).

4.2.4 Auswahl der Technologie

Generell ist die Verwendung der vorgestellten Technologien als Wiedergabemedium von Inhalten auf einer Plexiglaspyramide stark abhängig von dem Zweck zu welchem das System eingesetzt werden soll. Wenn die interaktive Visualisierung bspw. als Präsentationsmedium auf einer Messe Anwendung findet, sollte der verwendete Bildschirm eine gewisse Größe haben, um von möglichst vielen Betrachtern gesehen zu werden. Deshalb käme an dieser Stelle der Einsatz eines Plasmadisplays in Betracht, da diese mit einer sehr großen Bilddiagonale hergestellt werden können. Anhand der vorangegangenen Erläuterungen wären OLEDs für die Verwendung als Anzeigemedium innerhalb des Pyramiden-Aufbaus am besten geeignet. Durch ihre flache Bauweise ist der Platzverbrauch oberhalb der Pyramide am geringsten, so dass der Betrachter weniger von der eigentlichen Illusion abgelenkt wird. Darüber hinaus stellen diese Displays eine enorme Gewichtsreduktion dar, was einen mobilen Einsatz des Pyramiden-Displays erleichtert. Ebenso bieten OLEDs durch das selbstständige Emittieren von Licht ein sehr gutes Kontrastverhältnis, was für eine gute Bilddarstel-

lung auf der Pyramide förderlich ist. Da die Herstellung dieser Monitore allerdings sehr kostenintensiv ist und sich noch in der Entwicklung befindet, ist fraglich, ob die Vorteile dieser Technik den hohen Kostenaufwand ausgleichen können. Auf Grund der Tatsache, dass LCD Monitore unter der Verwendung von CCFLs ein sehr schlechtes Verhältnis von hellen zu dunklen Pixeln besitzen und schwarze Bildinhalte gräulich wirken können, sind diese Displays nur bedingt für die Verwendung der Bildwiedergabe geeignet. Für den reinen Versuchsaufbau, bei welchem die Realisierbarkeit von Echtzeitanwendungen auf einem Pyramiden-System im Vordergrund steht und die Qualität der Wiederzugebenen Inhalte eine untergeordnete Rolle spielt, stellt die Verwendung von LCDs mit CCFLs auf Grund der niedrigen Anschaffungskosten eine gute Alternative zu anderen Systemen dar. Nach Abwägung der einzelnen Vor- und Nachteile der unterschiedlichen Technologien ist der Einsatz von LED-LCDs am besten geeignet. Neben dem geringen Stromverbrauch sowie der flachen und leichten Bauweise dieser Displays lassen sich im Vergleich zu LCD CCFls gute Bildergebnisse erzielen. Da diese Technik bereits seit längerem in verschiedenen Bereichen angewandt wird, ist eine vergleichsweise kostengünstige Produktion der Displays möglich.

5 Theoretische Einordnung des Versuchsaufbaus

In der Fachliteratur lassen sich verschiedene Definitionen der im Versuchsaufbau verwendeten Anzeigetechnik finden (Xu et al. 2012, S. 395 - 396). Ebenso verwenden einige Hersteller solcher Display-Systeme fälschlicher Weise den Begriff „Hologramm“ oder eine Umschreibung dieser Bezeichnung, um die pyramidiale Anzeige zu beschreiben (Magic Holo 2015b). Aus diesem Grund erfolgt in Abschnitt 5.1 der Ansatz einer Zuordnung der verwendeten Technik in Bezug auf verschiedene dreidimensionale Anzeige-Systeme, um anhand der Zuordnung eine eindeutigere Bezeichnung aufstellen zu können. In Unterpunkt 5.2 wird eine Einordnung des Versuchsaufbaus innerhalb des Realitäts-Virtualitäts-Kontinuums angestrebt. Zu diesem Zweck werden anhand von Beispielen verschiedene Positionen innerhalb des Kontinuums beschrieben, um den Versuchsaufbau anschließend darin einordnen zu können.

5.1 Dreidimensionale Anzeigetechniken

Es gibt eine ganze Reihe an technischen Entwicklungen, mit welchen Bildinhalte dreidimensional dargestellt werden können. Diese unterscheiden sich zum Teil durch die Art der Anzeige sowie in deren Möglichkeiten der Bildwiedergabe und technischen

Umsetzungen. Innerhalb dieses Abschnitts werden Techniken vorgestellt, mit welchen es ohne zusätzliche Hardware möglich ist, einen möglichst großen dreidimensionalen Betrachtungswinkel zu erzielen. Zum einen soll dieser Abschnitt Auskunft darüber geben, welche Entwicklungen auf diesem Gebiet bestehen und zum anderen, in welcher Weise sich der Versuchsaufbau zu einem der dargestellten Display-Systeme zuordnen lässt, um daraufhin eine genauere Definition der Anzeigetechnik aufzustellen.

5.1.1 Volumetrische Displays

Volumetrische Anzeigetechniken zeichnen sich dadurch aus, einen Gegenstand oder eine Szene in ihrer natürlichen Dreidimensionalität abbilden zu können. Dementsprechend können die dargestellten Inhalte von diversen Betrachtungswinkeln korrekt wiedergegeben werden. Im Gegensatz zu stereoskopischen Alternativen werden den Augen bei volumetrischen Displays keine zwei verschiedene Bilder zugespielt, es handelt sich vielmehr um die tatsächliche dreidimensionale Abbildung eines Objekts mit korrekter Tiefenwirkung und Bewegungsparallaxe. Um dies zu realisieren, gibt es unterschiedliche technische Lösungsansätze, welche sich in zwei verschiedene Kategorien einordnen lassen (Lueder 2012, S. 237). Die Unterschiede von verschiedenen volumetrischen Displays lassen sich am besten anhand von spezifischen Beispielen verdeutlichen.

5.1.2 Static Volume Displays

Bei statischen volumetrischen Displays erfolgt die dreidimensionale Bilddarstellung anhand eines starren Gebildes, verschiedenen Flüssigkeiten oder mittels gasförmiger Substanzen, welche entweder eigenständig Licht abgeben oder aber von einer Projektion bespielt werden (Blundell / Schwarz 2006, S. 293). Ein Beispiel einer statisch, volumetrischen Anzeigetechnik ist das Projekt Fairy Lights in Femtoseconds (Abb. 59). Das Forschungsprojekt entstand aus der Zusammenarbeit verschiedener japanischer Universiäten und basiert auf einem sogenannten „Femtosecond Laser“. Diese Art von Laser ist in der Lage, das Potenzial von Atomen zu verändern (Ionisieren). Hierbei fokussiert

Abb. 59: *Fairy Lights in Femtoseconds*

der Laser Luftmoleküle, welche mit einem Lichtimpuls von 30 bis 269 Femtosekunden (eine Femtosekunde entspricht 10^{-15} Sekunden) beschossen werden. Das bei der Ionisation entstehende Plasma erzeugt einen Lichtpunkt (Voxel) an der vom Laser fokussierten Stelle im Raum. Getestet wurde dieses Verfahren mit zwei unterschiedlichen Femtosecond Lasern, welche 1000 sowie 200.000 Lichtpunkte pro Sekunde erzeugen können. Mittels verschiedener Linsen und Spiegel sowie der hohen Pulsrate des Lasers ist es möglich, dreidimensionale Lichtobjekte schwebend in einen Raum zwischen 1 und 5 Kubikzentimetern zu projizieren. Die bespielbare Fläche ist bei dieser Art der Darstellung von der verwendeten Technik und der Leistung der einzelnen Komponenten abhängig. Ein Vorteil des beschriebenen Verfahren ist, im Gegensatz zu vielen anderen Techniken, die Möglichkeit, die Lichtdarstellungen anfassen zu können. Bei der Berührung des Objekts sendet das Plasma eine Druckwelle aus, worüber ein haptisches Feedback entsteht. Ebenso sind Interaktionen bei der Berührung mit dem Objekt möglich. Problematisch bei diesem Anzeigeverfahren ist die Gefahr von Verletzungen an Haut und Augen beim Umgang mit den Laserdarstellungen. Zwar wird die Produktion von Lichtpunkten bei einer Berührung innerhalb von 17 Millisekunden gestoppt, wodurch ein Anfassen ungefährlich wird, allerdings können bei direktem Lichteinfall Schäden an den Augen entstehen (Ochiai et al. 2015).

5.1.3 Swept Volume Displays

Sogenannte „Swept Volume Displays“ sind mechanische Systeme, welche durch schnelle Bewegungen einer Anzeigefläche eine volumenartige Oberfläche erzeugen, auf welcher eine Szene in dreidimensionaler Weise dargestellt werden kann. Hierbei werden einzelne Voxel synchron zur Bewegung der Anzeigefläche erzeugt. Die Bewegung der Oberfläche findet typischerweise in Form einer Rotation oder einer Auf- und Abbewegung statt. Die Darstellung der einzelnen räumlichen Bildpunkte kann von auf der beweglichen Fläche montierten Lichtquellen oder durch zweidimensionale Projektionen auf der Oberfläche erzeugt werden. Durch die schnelle Bewegung der Anzeigefläche und differenzierter, positionsabhängiger Bildwiedergabe lassen sich 360-Grad-Ansichten von computergenerierten Elementen darstellen (Blundell / Schwarz 2006, S. 290). Dieses Prinzip lässt sich anhand von aktuellen Entwicklungen näher erläutern. Das Swept Volume Display mit dem Namen „Voxiebox“ der Firma Voxon, welches sich zur Zeit noch in der Entwicklung befindet, ist in der Lage, Animationen oder statische 3D-Objekte auf einer sich auf- und abbewegenden transparenten Bildfläche zu rendern (Abb. 60). Dabei wird das darzustellende Objekt in hunderte horizontale Scheiben aufgeteilt und synchron zur Bewegung auf die Anzeigefläche

Abb. 60: *Voxon Voxiebox*

projiziert. Für die Berechnung der einzelnen Bildebenen sorgt laut Voxon eine sogenannte „Volume Rendering Engine“ (Voxon 2016). Mittels des von Voxon entwickelten Prototypen lassen sich über eine halbe Milliarde Voxel pro Sekunde darstellen, welche auf Grund von Nachleuchten der Lichtpunkte zu einem 3D-Objekt verschmelzen (Voxon 2016). Mit der Voxiebox sind sowohl einfarbige als auch mehrfarbige Ausgaben möglich. Laut Schriftverkehr mit einem Mitarbeiter bei Voxon kann durch die Überblendung der drei Grundfarben Rot, Grün und Blau, eine Vielzahl unterschiedlicher Farben angezeigt werden (Tamblyn 2016).

5.1.4 Autostereoscopic Displays

Autostereoskopische Displays erzeugen eine dreidimensionale Wahrnehmung über das im Abschnitt 2.2 „Dreidimensionale Wahrnehmung“ bereits erklärte stereoskopische Verfahren. Allerdings wird hierbei keine zusätzliche Hardware benötigt, um den dreidimensionalen Effekt sichtbar zu machen. Darüber hinaus ist bei teureren Geräten der Blickwinkel im Gegensatz zu herkömmlichen stereoskopischen Displays nicht auf einen festen Punkt bestimmt, sodass bei einer Kopfbewegung verschiedene Ansichten eines Objekts gesehen werden können und die Erzeugung von parallaktischen Effekten möglich ist. Um dies zu realisieren, gibt es verschiedene Ansätze. Einer ist die Verwendung von linsenförmigen Zylindern vor dem Bildschirm. Durch die Lichtbrechung der zweidimensionalen Pixelbilder in verschiedene Richtungen können den Augen unterschiedliche Bilder zugespielt werden, wodurch ein dreidimensionales Bild entsteht. Hierbei ist es möglich, die Bildfläche so aufzuteilen, dass verschiedene Betrachtungswinkel stereoskopisch dargestellt werden können. Allerdings sinkt die Pixelauflösung mit zunehmenden 3D-Ansichten, außerdem ist durch die zylindrische Form der Optiken lediglich eine horizontale stereoskopische Darstellung möglich (Lueder 2012, S 73-

Abb. 61: *Sony RayModeller*

74). 2010 stellte die Firma Sony einen Prototypen eines autostereoskopischen Bildschirms im Rahmen der SIGGRAPH Messe in Los Angeles vor. Der Prototyp mit dem Namen „RayModdeler“ ist in der Lage 360 verschiedene Ansichten eines Objekts autostereoskopisch darzustellen (Abb. 61). Auf dem 27 x 13 cm großen zylinderförmigen LED Bildschirm können sowohl statische als auch interaktive 3D Szenen wiedergegeben werden. Der RayModeller ist laut Sony in verschiedenen Bereichen wie z.B. der Medizin, Bildung, Unterhaltung oder Telekommunikation einsetzbar (Sony 2010).

5.1.5 Zuordnung des Versuchsaufbaus

Eine Zuordnung des Versuchsaufbaus zu einem der vorgestellten Display-Systeme ist nur bedingt möglich, da lediglich bestimmte Aspekte der Systeme für den Aufbau zutreffend sind. Generell hängt es stark von einzelnen Bestandteilen der Technik ab, zu welchem System eine Technik zählen kann. Im Beispiel des RayModellers ordnet Sony den Prototypen den volumetrischen Displays zu, obwohl bei dieser Technik keine Voxel erzeugt werden und es sich dabei lediglich um autostereoskopische zweidimensionale Bilder handelt (Sony 2010). Da der Versuchsaufbau ebenfalls zweidimensionale Bilder eines Objekts darstellt und keine volumetrischen Pixel generiert werden, ist die Zuordnung zu einem volumetrischen System nicht gegeben. Es ist zwar möglich, ein Objekt in vier verschiedenen Ansichten innerhalb der Anzeigefläche des Versuchsaufbaus darzustellen, allerdings entsteht durch den großen Abstand von 90 Grad der Bildflächen zueinander kein stereoskopischer Effekt. Theoretisch könnte der Versuchsaufbau als ein Head-up-Display bezeichnet werden, bei welchem dem Betrachter zusätzliche Informationen ins Sichtfeld eingeblendet werden, wobei die Wahrnehmung der realen Umgebung sichtbar bleibt (Beech 2012, S. 59). Allerdings ist diese Einordnung sehr oberflächlich und beschreibt nicht die Möglichkeit, ein virtuelles Objekt von verschiedenen Seiten zu betrachten und mit ihm interagieren zu können. In dem Buch „Meet the Kinect: An Introduction to Programming Natural User Interfaces“ werden Wiedergabetechniken wie der Versuchsaufbau als Pepper’s Ghost-based Display bezeichnet. Hierbei werden verschiedene Möglichkeiten, diesen Illusionseffekt zur Anzeige von Objekten zu nutzen, zusammengefasst. Neben dem pyramidalen Aufbau zählen nach Kean et al. auch parallele Anzeigetechniken, welche eine Bewegungsparallaxe ermöglichen, oder die bereits beschriebenen großflächigen Projektionen auf Veranstaltungen zu dieser Kategorie (Kean et al. 2011, S. 181 - 184). Darüber hinaus lassen sich sehr weitreichende Oberbegriffe wie „Reflection Based Display“ oder „Pyramid Display“ finden (Xu et al. 2012, S. 395 - 396). Da die genannten Zuordnungen für den Versuchsaufbau nicht aussagekräftig genug sind, möchte ich an dieser Stelle

einen eigenen Begriff zur Beschreibung anführen. Auf Grund der Tatsache, dass der interaktive Echtzeitcharakter im Fokus des Aufbaus liegt, sollte dieser Bestandteil der Benennung sein. Um eine Verwechslung mit anderen Anzeigetechniken auszuschließen, bietet sich die Verwendung des Begriffs „Pepper's Ghost" an, da hierin das spezielle Prinzip der Reflexion auf einer transparenten Fläche als Anzeigesystem beinhaltet ist. Darüber hinaus ist die Form der Anzeigefläche für die Umsetzung von bestimmten Anwendungen entscheidend. Eine vierflächige Pyramide hat im Gegensatz zu anderen Konstruktionen den Vorteil, dass die virtuellen Elemente von mehreren Seiten betrachtet werden können, woraus sich unterschiedliche Anwendungsbereiche ergeben, aber auch Anforderungen an Technik und Umsetzung gestellt werden. Da eine Unterscheidung zu Pyramiden mit drei Seiten lediglich einen Unterschied des möglichen Betrachtungswinkels ausmacht, ist eine genaue Benennung der Flächenanzahl nicht erforderlich. Aus den eben genannten Gründen kann der Versuchsaufbau als Interaktiver Pepper's Ghost Pyramiden-Display bezeichnet werden.

5.2 Realitäts-Virtualitäts-Kontinuum

Um den Versuchsaufbau in das Realitäts-Virtualitäts-Kontinuum einordnen zu können, erfolgt zunächst eine Erklärung von verschiedenen Begrifflichkeiten, welche den Zusammenhang von Realität und Virtualität näher klassifizieren.

5.2.1 Mixed Reality

Der Begriff „Mixed Reality" (MR) beschreibt die lückenlose Verbindung zwischen realen Umgebungen und virtuellen Welten. Innerhalb dieses Konstruktes, dem Realitäts-Virtualitäts-Kontinuum, bestehen von der vollständig realen Umgebung zur vollständig virtuellen Umgebung keinerlei feste Abstufungen. Der Begriff „Mixed Reality" umfasst diverse Arten von Systemen. Diese können von holographischen Aufnahmen bis hin zu räumlichen Wandprojektionen alle denkbaren Formen von Aufbauten beinhalten (Grau 2003, S. 245 und Preim / Dachselt 2015, S. 248). Anhand verschiedener Begrifflichkeiten innerhalb der MR-Definition ist es möglich, verschiedene Systeme und Arrangements mit unterschiedlichen Graden der Immersion, welche sich zwischen Realität und Virtualität bewegen, genauer einzuordnen.

Abb. 62: *SOMNIACS SA Birdly*

5.2.2 Virtual Reality

Mit dem Begriff „Virtual Reality“ können virtuelle Welten beschrieben werden, welche sich durch die Anzeige von realitätsnahen dreidimensionalen Darstellungen auszeichnen. Diese Umgebungen verfügen über einen hohen Grad an Immersion, welcher durch verschiedene technische Lösungen realisiert werden kann. Neben der Verwendung von Spezialbrillen, den sogenannten „Head-Mounted-Displays“, für die Anzeige von stereoskopischen Bildern, können verschiedenste Arten von Eingabe- und Anzeigegeräten zum Einsatz kommen (Heinecke 2012, S. 9). Am besten lässt sich VR anhand von konkreten Beispielen beschreiben. 2013 startete die Züricher Hochschule der Künste (ZHdK) ein Forschungsprojekt, bei welchem mittels eines eigens entwickelten VR-Systems die Simulation eines Vogelflugs erforscht wurde. Die ersten Prototypen des virtuellen Vogelflugsimulators Birdly erschienen 2014 (Abb.62) (SOMNIACS SA 2015a). Die Installation bietet eine Ganzkörpererfahrung, welche den Benutzer in die virtuelle Umgebung einbindet. Auf dem Bauch liegend werden Armbewegungen als Flügelschläge umgesetzt und dienen darüber hinaus der Fortbewegung und Körperausrichtung in der computergenerierten Welt. Durch das Anheben und Neigen der Vorrichtung werden physikalische Kräfte in der virtuellen Umgebung auf den Körper des Anwenders übertragen. Die Darstellung der Landschaft wird mittels eines Head-Mounted-Displays umgesetzt (SOMNIACS SA 2015b).

5.2.3 Augmented Reality

Im Gegensatz zu VR, bei welcher versucht wird, den Anwender in eine virtuelle Welt eintauchen zu lassen, wird unter dem Begriff „Augmented Reality“ das Einbetten von computergenerierten Inhalten in die reale Umgebung verstanden (Bimber / Raskar 2005, S. 1 - 2). Es erfolgt also eine Anreicherung der Realität um virtuelle Inhalte und Darstellungen. In verschiedener Fachliteratur wird bei der AR davon ausgegangen, dass der Nutzer mit den dargestellten Zusatzinhalten interagieren kann. Des Weiteren spielt die räumliche oder örtliche Position des Benutzers sowie die der virtuellen Anwendung eine entscheidende Rolle (Heinecke 2012, S. 9 - 10).

„Augmented Reality bezeichnet die Überlagerung der realen Welt durch computergenerierte Informationen [...]. AR-Systeme erfordern, dass die Position und Lage von Objekten in der realen Welt korrekt erfasst werden [...]. Wie die virtuelle Realität, basiert auch die Augmented Reality auf der Möglichkeit, in Echtzeit zu interagieren“ (Preim / Dachselt 2015, S. 248).

Abb. 63: *Microsoft HoloLens*

Um ein Beispiel für die Anwendung von AR zu geben, erfolgt an dieser Stelle eine knappe Beschreibung der bereits erwähnten HoloLens (Abb. 63). Im Fall des Head-Mounted-Displays von Microsoft, werden zwei transparente Displays vor den Augen des Trägers mit stereoskopischen Bildern bespielt. Dadurch ist es möglich, räumlich wirkende Objekte darzustellen, bei denen der Träger jedoch im Gegensatz zu einer reinen VR-Brille die reale Umgebung weiterhin sehen kann. Darüber hinaus verfügt die Brille über Lagesensoren, welche Kopfbewegungen des Trägers registrieren, sodass es möglich ist, die auf den Displays erscheinenden Inhalte, an der realen Umgebung auszurichten. Des Weiteren hat der Träger die Möglichkeit, mit den in die Umwelt projizierten Darstellungen zu interagieren (Microsoft 2016c).

5.2.4 Einordnung des Versuchsaufbaus

Bei dem verwendeten Versuchsaufbau handelt es sich eindeutig um eine Form der Mixed Reality, da computergenerierte Inhalte mit realen Objekten kombinieren werden können. Selbst wenn dies nicht der Fall sein sollte, befindet sich die technische Umsetzung innerhalb des Realitäts-Virtualitäts-Kontinuums, da sie nicht aus rein reellen Inhalten sondern Computergrafiken besteht. Eine Untereinordnung in die virtuelle Realität kann ausgeschlossen werden. Es besteht bei der Projektion von 3D-Inhalten auf der holographischen Pyramide im Vergleich zu einem Head-Mounted-VR-Display nur ein geringer Grad an Immersion. Des Weiteren befindet sich der Betrachter nicht innerhalb einer virtuellen Umgebung, sondern betrachtet und interagiert lediglich mit virtuellen Objekten. Laut der sehr strikten Definition von AR-Systemen von Preim und Dachselt kann der Versuchsaufbau nicht in vollem Umfang der Augumented Reality zugeordnet werden. So wie der Versuchsaufbau aktuell konzipiert ist, spielt die Position des Betrachters keine Rolle für die Anzeige der Inhalte. Wie allerdings im Abschnitt

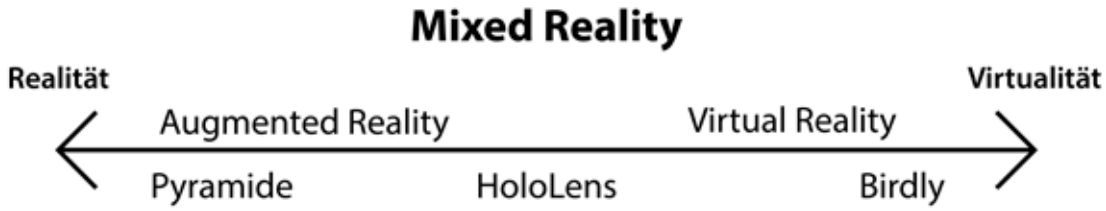

Abb. 64: *Grafische Darstellung von Mixed Reality und Einordnung des Versuchsaufbaus innerhalb des Realitäts-Virtualitäts-Kontinuums*

3.11 beschrieben wurde, kann der Aufbau mittels eines Head-Tracking-Systems dahingehend verbessert werden, dass sich die Ausrichtung des Objekts dem Betrachtungswinkel des Beobachters anpasst. Interaktionsmöglichkeiten in Echtzeit können wie zuvor beschrieben durch verschiedene Eingabegeräte ermöglicht werden. Ebenso ist eine ständige Vermischung aus realen und virtuellen Inhalten gegeben, sobald die Projektionen ausgeführt werden. Des Weiteren besteht die Möglichkeit, reale Gegenstände innerhalb der Glaspyramide mit Projektionen zu umspielen, wodurch eine Kombination aus Realität und Virtualität entsteht. Da sich in der Fachliteratur Beispiele für AR-Systeme finden lassen, bei denen nicht alle Funktionen von AR erfüllt werden, welche von Preim und Dachselt definiert wurden und der Versuchsaufbau die Möglichkeit der Erweiterung durch Head-Tracking bietet, kann die holographische Projektion im Fall der beschriebenen Installation als ein AR-System bezeichnet werden. Eine Verdeutlichung der Einordnung des Versuchsaufbaus innerhalb des Realitäts-Virtualitäts-Kontinuums zeigt Abbildung 64.

6 Fazit

Wie anhand der Zielsetzung definiert wurde, konnte die Realisierung von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide umgesetzt werden. Darüber hinaus konnten Kenntnisse über das verwendete System gesammelt werden, welche eine nähere Beschreibung des Versuchsaufbaus ermöglichen. Des Weiteren wurden Möglichkeiten und Grenzen der Technik sowie der verwendeten Entwicklungsumgebungen für den pyramidalen Aufbau aufgezeigt. Bei der Entwicklung des Pyramiden-Systems sowie der Realisierung von interaktiven Echtzeitanwendungen konnte festgestellt werden, dass es sich hierbei um relativ komplexe Strukturen handelt, die durchdrungen werden müssen. Diese beziehen sich sowohl auf die Wirkungsweise von Darstellungen innerhalb der Pyramide als auch auf deren Umsetzung, mögliche Einsatzbereiche und die Zuordnung des Systems zu einer Display-Technologie.
Im Rahmen dieser Ausarbeitung konnten nicht alle Bereiche, welche den Versuchs-

aufbau betreffen, ausreichend behandelt werden. Vielmehr konnte ein Überblick über verschiedene Teilbereiche des Systems gegeben werden. Um die verschiedenen Themenfelder rund um den Versuchsaufbau genauer erfassen zu können, müssen weitere Versuche durchgeführt werden. Die Einschätzung, für welche Bereiche und in welcher Form interaktive Echtzeitanwendungen auf einer holographischen Pyramide zum Einsatz kommen können, wäre ein Themenpunkt, welcher einer genaueren Analyse und testweisen Untersuchung bedarf. Darüber hinaus müsste eine detaillierte Hinterfragung der Wirkungsweise von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide näher untersucht werden um bspw. Rückschlüsse auf den Einfluss auf den Rezipienten einer Anwendung im Bereich der Produktpräsentation ziehen zu können. Zudem könnte untersucht werden, inwiefern sich die Interaktion mit dargestellten Elementen auf das Verhalten eines Rezipienten gegenüber einer Marke oder eines Produktes auswirkt. Ebenso könnten weitere Versuche im Bereich der Spielentwicklung durchgeführt werden, um herauszufinden, für welche Art von Spielen sich ein solcher Aufbau anbieten kann und ob diese Anklang bei entsprechenden Zielgruppen finden. Ein weiteres Untersuchungsfeld ist im Bereich des Home Entertainment denkbar. Hierbei wäre es möglich zu überprüfen, ob sich ein solches System auch im privaten Bereich für bspw. die Telekommunikation, Unterhaltung oder zu Lernzwecken einsetzen lässt.

Anhand der gewonnenen Erkenntnisse komme ich zu der Schlussfolgerung, dass sich interaktive Echtzeitanwendungen für holographische Pyramiden vermutlich am besten zur Unterstützung im Bereich Produktpräsentation einsetzen lassen. Die immersive Wirkung, welche durch die Illusion einer holographischen Abbildung entsteht, kann meines Erachtens innerhalb relativ kurzer Zeit durchschaut werden. Sobald dem Rezipienten klar ist, in welcher Weise der holographische Effekt erzeugt wird, geht die überraschende Wirkung der Illusion zurück. Daher muss der Einsatz im privaten Bereich eher durch die spezielle Nutzung von vier transparenten Anzeigeflächen anstatt durch die holographische Illusion begründet sein. Wie durch diese Ausarbeitung verdeutlicht wurde, ergeben sich hierbei gewisse Vorteile gegenüber einer Darstellung auf einem einzigen Monitor, allerdings ist fraglich, ob sich eine holographische Pyramide neben dem Einsatz eines herkömmlichen Displays behaupten kann. Während die vorliegende Arbeit die Realisierbarkeit von interaktiven Echtzeitanwendungen auf einer holographischen Pyramide aufgezeigt hat, bedarf es der Erhebung weiterer Studien, um die Frage nach dem alltäglichen Einsatz des Pyramiden-Systems in Bezug auf interaktive Anwendungen näher analysieren zu können.

Literaturverzeichnis

Austinat, Roland / Gieselmann, Hartmut / Janssen, Jan-Keno (2015): Ausprobiert: Zukünftige Virtual-Reality-Brillen im Praxis-Check, in: Heise, Christian / Heise, Ansgar / Persson, Christian (Hrsg.), c‘t wissen: Virtual Reality, 2015, Sonderheft, Hannover: Heise Medien GmbH & Co. KG, S. 42 - 48.

Arora, Sumeet (2014): WebGL Game Development: Gain insights into game development by rendering complex 3D objects using WebGL, Birmingham: Packt Publishing, Ltd..

Beech, Martin (2012): The Physics of Invisibility: A Story of Light and Deception, New York u. a.: Springer Science+Business Media, LLC..

Bimber, Oliver / Raskar, Ramesh (2005): Spatial Augmented Reality: Merging Real and Virtual Worlds, Wellesley MA: A K Peters, Ltd..

Blundell, Barry G. / Schwarz, Adam J. (2006): Creative 3-D Display and Interaction Interfaces: A Trans-Disciplinary Approach, Hoboken NJ u. a.: John Wiley & Sons, Inc..

Busby, Jason / Parrish, Zak / Wilson, Jeff (2009): Mastering Unreal Technology: Volume I: Introduction to Level Design with Unreal Engine 3, aus der Reihe: Mastering Unreal Technology, Bd. 1, Indianapolis: Sams Publishing.

Böhringer, Joachim / Bühler, Peter / Schlaich, Patrick / Sinner, Dominik (2014): Kompendium der Mediengestaltung: II. Medientechnik, 6. überarb. u. erw. Aufl., aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag.

Bühler, Peter (2004): MediaFarbe - analog und digital: Farbe in der Medienproduktion, 2. überarb. u. erw. Aufl., aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag.

Clark, Stuart (2007): Vanities of the Eye: Vision in Early Modern European Culture, Oxford u. a.: Oxford University Press.

Ditzinger, Thomas (2013): Illusionen des Sehens: Eine Reise in die Welt der visuellen Wahrnehmung, 2. überarb. u. erw. Aufl., Berlin u. a.: Springer-Verlag.

Eichler, Hans Joachim / Eichler, Jürgen (1995): Laser: High-Tech mit Licht, Berlin u. a.: Springer-Verlag.

Grau, Oliver (2003): Virtual Art: From Illusion to Immersion, Cambridge MA u. a.: The MIT Press.

Heinecke, Andreas M. (2012): Mensch-Computer-Interaktion: Basiswissen für Entwickler und Gestalter, 2. überarb. u. erw. Aufl., aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag.

Jockenhövel, Jesko (2014): Der digitale 3D-Film: Narration, Stereoskopie, Filmstil, aus der Reihe: Film, Fernsehen, Medienkultur: Schriftenreihe der Hochschule für Film und Fernsehen „Konrad Wolf“, Wiesbaden: Springer Fachmedien.

Kean, Sean / Hall, Jonathan / Perry, Phoenix (2011): Meet the Kinect: An Introduction to Programming Natural User Interfaces, aus der Reihe: Technology in Action, New York: Apress.

Liesegang, F. Paul (1992): Das Projektionswesen, in: Hay, Alfred (Hrsg.) weitergeführt von M. v. Rohr, Wissenschaftliche Anwendungen der Photographie: Erster Teil: Stereophotographie · Astrophotographie · Das Projektionswesen, Stuttgart: Lindemanns-Verlag, S. 234-283.

Lipson, Stephen G. / Lipson, Henry S. / Tannhauser, David S. (1997): Optik, übersetzt von Holger Becker, Berlin u. a.: Springer-Verlag.

Lueder, Ernst (2012): 3D Displays, aus der Reihe: Wiley-SID Series in Display Technology, Chichester u. a.: John Wiley & Sons, Ltd..

Meschede, Dieter (2008): Optik, Licht und Laser, 3. Aufl., Wiesbaden: Vieweg+Teubner | GWV Fachverlage GmbH.

North, Dan / Rehak, Bob / Duffy, Michael S. (2015): Special ws: New Histories, Theories, Contexts, London u. a.: Palgrave.

Ostrowski, Juri I. (1987): Holografie: Grundlagen, Experimente und Anwendungen, 2. Aufl., aus der Reihe: Kleine Naturwissenschaftliche Bibliothek, Bd. 64, übersetzt von Dr. W. Osten, Leipzig: BSB B. G. Teubner Verlagsgesellschaft.

Overschmidt, Gordian / Schröder, Ute B. (2013): Kapitel 1: Einleitung, in: Overschmidt, Gordian / Schröder Ute B. (Hrsg.), Fullspace-Projektion: Mit dem 360°lab zum Holodeck, aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag, S. 3 - 10.

Preim, Bernhard / Dachselt, Raimund (2015): Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces, 2. Aufl., aus der Reihe: eXamen.press, Berlin u. a.: Springer-Verlag.

Ritter, Marko (2013): Kapitel 8: Design interaktiver und immersiver Erlebnisse, in: Overschmidt, Gordian / Schröder Ute B. (Hrsg.), Fullspace-Projektion: Mit dem 360°lab zum Holodeck, aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag, S. 101 - 120.

Schmidt, Ulrich (2013): Professionelle Videotechnik: Grundlagen, Filmtechnik, Fernsehtechnik, Geräte- und Studiotechnik in SD, HD, DI, 3D. 6. Aufl., Berlin u. a.: Springer-Verlag.

Scholz, Peter (2005): Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung, aus der Reihe: Xpert.press, Berlin u. a.: Springer-Verlag.

Schröder, Ute B (2013): Kapitel 6: Erforschung von 360°-Welten: Überlegungen und sozialwissenschaftliche Rekonstruktion von Handlungspraxen in der Auseinandersetzung mit dem 360°-Medium, in: Overschmidt, Gordian / Schröder Ute B. (Hrsg.), Fullspace-Projektion: Mit dem 360°lab zum Holodeck, aus der Reihe: X.media.press, Berlin u. a.: Springer-Verlag, S. 67 - 85.

Van Albada, L. E. W. (1992): Stereophotographie, in: Hay, Alfred (Hrsg.), weitergeführt von M. v. Rohr, Wissenschaftliche Anwendungen der Photographie: Erster Teil: Stereophotographie · Astrophotographie · Das Projektionswesen, Stuttgart: Lindemanns-Verlag, S. 1 - 100.

Wegener, Claudia / Jockenhövel, Jesko / Gibbon, Mariann (2012): 3D-Kino: Studien zur Rezeption und Akzeptanz, aus der Reihe: Film, Fernsehen, Medienkultur: Schriftenreihe der Hochschule für Film und Fernsehen „Konrad Wolf“, Wiesbaden: VS Verlag für Sozialwissenschaften | Springer Fachmedien.

Welsch, Norbert / Liebmann, Claus Chr. (2012): Farben: Natur, Technik, Kunst, 3. verb. u. erw. Aufl., Heidelberg: Spektrum Akademischer Verlag.

Xu, Shuhong / Wu, Bin / Ge, Dongyun / Chen, Lei / Yang, Hongyan (2012): Building an Inverted Pyramid Display for Group Learning, in: Xiao, Tianyuan / Zhang, Lin / Fei, Minrui (Hrsg.), AsiaSim 2012: Asia Simulation Conference 2012: Shanghai, China, October 27-30, 2012: Proceedings, Part III, aus der Reihe: Communications in Computer and Information Science, Bd. 325, Heidelberg u. a.: Springer-Verlag, S. 394 - 404.

Zeman , Nicholas Bernhardt (2015): Essential Skills for 3D Modeling, Rendering, and Animation, Boca Raton FL u. a.: CRC Press.

Zerbst, Stefan / Düvel, Oliver (2004): 3D Game Engine Programming, Boston MA: Premier Press.

Online-Quellen

Arkema, Inc. (2000): Plexiglas: Optical & Transmission Characteristics, [online] http://www.plexiglas.com/export/sites/plexiglas/.content/medias/downloads/sheet-docs/plexiglas-optical-and-transmission-characteristics.pdf [05.05.2016].

Babylon.js (2016a): WebGL: Simple Powerful A complete JavaScript framework for building 3D games with HTML5, WebGL and Web Audio, [online] http://www.babylonjs.com/ [10.05.2016].

Babylon.js (2016b): Inhaltsverzeichnis, [online] https://doc.babylonjs.com/overviews/Documentation_Category_Map [31.05.2016].

Bruxafol (2015a): Quantum 14, [online] http://bruxsafol.de/wp-content/uploads/2015/06/AGF-QUANTUM-14-de-20130212.pdf [20.05.2016].

Bruxafol (2015b): Quantum 15, [online] http://bruxsafol.de/wp-content/uploads/2015/06/AGF-QUANTUM-15-de-20130212.pdf [20.05.2016].

Bruxafol (2015c): Quantum 16, [online] http://bruxsafol.de/wp-content/uploads/2015/06/AGF-QUANTUM-16-de-20130212.pdf [20.05.2016].

Bruxafol (2015d): Quantum 25, [online] http://bruxsafol.de/wp-content/uploads/2015/06/AGF-QUANTUM-25-de-20130212.pdf [20.05.2016].

Epic Games, Inc. (2016a): Introduction to C++ Programming in UE4, [online] https://docs.unrealengine.com/latest/INT/Programming/Introduction/index.html [29.04.2016].

Epic Games, Inc. (2016b): IF YOU LOVE SOMETHING, SET IT FREE: Unreal Engine is now FREE, [online] https://www.unrealengine.com/what-is-unreal-engine-4 [17.05.2016].

Epic Games, Inc. (2016c): Unreal Engine Features, [online] https://www.unrealengine.com/unreal-engine-4 [17.05.2016].

Epic Games, Inc. (2016d): Unreal Engine 4 Documentation, [online] https://docs.unrealengine.com/latest/INT/ [31.05.2016].

Khronos Group (2016): The Standard for Embedded Accelerated 3D Graphics, [online] https://www.khronos.org/opengles/ [10.05.2016].

Kühl, Eike (2015): Microsoft Hololens: Vorsprung durch Trickserei, [online] http://www.zeit.de/digital/mobil/2015-10/microsoft-hololens-brille-blickfeld-kritik [04.04.2016].

Leap Motion, Inc. (2016a): Reach into new worlds, [online] https://www.leapmotion.com/ [28.05.2016].

Leap Motion, Inc. (2016b): Frame, [online] https://developer.leapmotion.com/documentation/javascript/api/Leap.Frame.html [21.05.2016].

Leap Motion, Inc. (2016c): JavaScript SDK Documentation, [online] https://developer.leapmotion.com/documentation/javascript/index.html [21.05.2016].

Magic Holo (2015a): Faszinierende USP‘s, [online] http://magic-holo.com/ [05.06.2016].

Magic Holo (2015b): Holographische Displays: Der innovative Eyecatcher, [online] http://magic-holo.com/?gclid=CPeu4fazn80CFRKNGwodvYIMuA [05.06.2016].

Microsoft (2016a): General Frequently Asked Questions: What is Microsoft HoloLens and how does it work?, [online] https://www.microsoft.com/microsoft-hololens/en-us/faq [04.04.2016].

Microsoft (2016b): Health & Safety: Use requirements, [online] https://www.microsoft.com/microsoft-hololens/en-us/legal/health-and-safety-information [04.04.2016].

Microsoft (2016c): Why Hololens, [online] https://www.microsoft.com/microsoft-hololens/en-us/why-hololens [11.04.2016].

Musion (2016a): EyeLiner, [online] http://musion.com/eyeliner/ [08.04.2016].

Musion (2016b): Tupac hologram, [online] http://musion.com/?portfolio=tupac-coachella-2012-hologram [24.03.2016].

Ochiai, Yoichi / Kumagai , Kota / Hoshi, Takayuki / Rekimoto, Jun / Hasegawa, Satoshi / Hayasaki, Yoshio (2015): Fairy Lights in Femtoseconds: Aerial and Volumetric Graphics Rendered by Focused Femtosecond Laser Combined with Computational Holographic Fields, [online] https://arxiv.org/ftp/arxiv/papers/1506/1506.06668.pdf [21.04.2016].

SELFHTML e.V. (2016): Inhaltsverzeichnis, [online] https://wiki.selfhtml.org/wiki/Startseite [31.05.2016].

SOMNIACS SA (2015a): From Design Research to the Market, [online] http://www.somniacs.co/about.php [04.04.2016].

SOMNIACS SA (2015b): The Ultimate Dream of Flying, [online] http://www.somniacs.co/ [04.04.2016].

Voxon (2016): Technology, [online] http://www.voxiebox.com/tech/ [16.04.2016].

Online-Videos

Sony (2010): Sony RayModeler: a 360-Degree Autostereoscopic Display Prototype, [online auf YouTube] https://www.youtube.com/watch?v=6BFKC-NKRFw [24.04.2016].

Sonstige Quellen

Tamblyn, Will (2016): Chief Operating Officer at VOXON, Schriftverkehr per Facebook: brief information about how many colors can be displayed by the Voxiebox, Kiel [16.04.2016].

Anhang - DVD

Auf der beigefügten DVD befinden sich verschiedene Testprogramme, welche mittels HTML5, CSS3 und dem Babylon.js-Framework für die holographische Pyramide erstellt wurden. Darüber hinaus sind verschiedene Unreal Engine 4 Realisierungen für das Pyramiden-System enthalten. Für die Erklärung der Bedienung der Programme befindet sich eine Liesmich-Datei innerhalb der verschiedenen Ordner auf der DVD. Ebenso auf dem Datenträger enthalten ist die schriftliche Ausarbeitung als Word- und PDF-Dokument sowie Nachweise für die genutzten Online-Quellen.