UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

DIN: NewTechnologies

Stereoskopische Video-Panoramen

Bachelorarbeit

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

vorgelegt von
Nico Winkler

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Nils Höhner M.Sc. Institut für Computervisualitik, Abteilung Computergrafik

Koblenz, im November 2018

Erklärung

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

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

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

In dieser Arbeit wird ein System zur Erzeugung und Darstellung stereoskopischen Video-Panoramen vorgestellt. Neben der theoretischen Grundlagen werden der Aufbau und die Funktionsweise dieses Systems erläutert. Dazu werden spezielle Kameras verwendet, die Panoramen aufnehmen können und zur Wiedergabe synchronisiert werden. Anschließend wird ein Renderer implementiert ,welcher die Panoramen mithilfe einer Virtual-Reality Brille stereoskopisch darstellen kann. Dafür werden separate Aufnahmen für die beiden Augen gemacht und getrennt wiedergegeben. Zum Abschluss wird das entstandene Video-Panorama mit einem Panorama eines schon bestehenden Systems verglichen.

Abstract

In this thesis a system for creating and displaying stereoscopic video panoramas is presented. Besides the theoretical basics, the structure and operation of this system will be explained. For this special cameras will be introduced for capturing panoramas. These panoramas need to be synchronized for displaying. Then a renderer is implemented which can stereoscopically display the panoramas using a virtual reality system by taking separate shots for the two eyes. Finally, the resulting video panorama is compared with a panorama of an already existing system.

Inhaltsverzeichnis

1 Einleitung

Über die letzten Jahre ist die Anzahl an verkauften Virtual-Reality Brillen weiter gestiegen, da die benötigten Hardware Anforderungen zur Verwendung einer Virtual-Reality Brille erschwinglicher werden.
Dadurch stehen durch Virtual-Reality Brillen neue Möglichkeiten zur Darstellung von Grafiken und interaktiven Programmen zur Verfügung. So können Bilder auch mit Tiefeneindruck, ähnlich einer 3D Aufnahme, dargestellt werden, die auch stereoskopische Bilder genannt werden. Für einen stereoskopischen Effekt müssen zwei Bilder mit einem Abstand aufgenommen werden, der sich ungefähr an dem menschlichen Augenabstand orientiert. Eine beliebte Art von Bildern in diesem Rahmen sind Panoramabilder. Bei einem Panoramabild handelt es sich um einen Rundumblick der gesamten Szene, welcher durch Kopfbewegungen innerhalb der Virtual-Reality Brille angeschaut werden kann. Dabei soll dem Benutzer vermittelt werden in der Szene zu stehen, als würden sie sich in einer echten Umgebung befinden.
Zur Erstellung solcher Panoramen wird entweder ein Kamerasystem oder eine sich rotierende Kamera benötigt. Dabei wird das Panorama aus mehreren aufgenommenen Bildern zusammengesetzt. Eine Herausforderung liegt hierbei in der Aufnahme sogenannter Video-Panoramen, bei welchen nun dynamische Objekte aufgenommen werden sollen. Bisweilen existierende Systeme zur Aufnahme von Video-Panoramen verwenden hierfür mehrere Kameras, die jeweils einen Teil der Szene aufnehmen und die einzelnen Frames dieser dann zusammenfügen. Solche Systeme sind aber vor allem für die Forschung und professionelle Aufnahmen gedacht, da sie preislich für Viele unerschwinglich sind. Deshalb wird in dieser Arbeit ein prototypisches System zur Aufnahme von Video-Panoramen vorgestellt, welches günstig realisiert werden kann. Dafür werden zwei Kameras mit speziellen Objektiven benutzt, die in der Lage sind weite Bereiche der Szene aufzunehmen.
Um die Aufnahmen des Panoramas in der Virtual-Reality Brille darzustellen, wird ein Programm benötigt, welches die Panoramen stereoskopisch wiedergeben kann. Es muss sichergestellt werden, dass die beiden Stereopaare keinen Zugriff aufeinander haben und nur auf ihrem Auge der Virtual-Reality Brille wiedergegeben werden. Ein solcher Renderer sollte dabei im Idealfall mit allen Virtual-Reality Brillen kompatibel sein.
Zum Abschluss wird das hier vorgestellte System mit einem derzeitigen Forschungsprodukt gegenübergestellt und ein Fazit gezogen.

2 Grundlagen

2.1 Stereoskopie

Stereoskopie nach [Wim04] ist die Wiedergabe zweier Bilder, wobei durch bestimmte Anordnung der Bilder ein Tiefeneindruck entsteht. Der Mensch nimmt seine Umgebung mit zwei Augen wahr, die in einem Abstand von zirka 6,5cm parallel angeordnet sind. Dadurch kommt es zu leicht unterschiedlichen Abbildungen auf der Netzhaut, wodurch Entfernungen abgeschätzt werden können.
In Abbildung 1 stellt Punkt 1 den Fixpunkt dar, also den Punkt, der auf der Netzhaut mit der höchsten Sehschärfe abgebildet wird. Punkt 3 wird zwar in beiden Augen links des Fixpunktes abgebildet, allerdings zu unterschiedlichen Abständen. Diese Abstandsunterschiede werden im Zusammenhang mit Stereoskopie Parallaxe genannt. Die Parallaxe kann das menschliche Gehirn als Tiefeninformation verwenden.
Ein Beispiel für Stereoskopie sind 3D Filme. Dabei werden die Bilder für den Film aus 2 Perspektiven aufgenommen und überlagert. Durch eine spezielle Brille wird dann das eine Bild nur von dem linken Auge wahrgenommen und das andere Bild nur von dem rechten Auge.[Wim04].

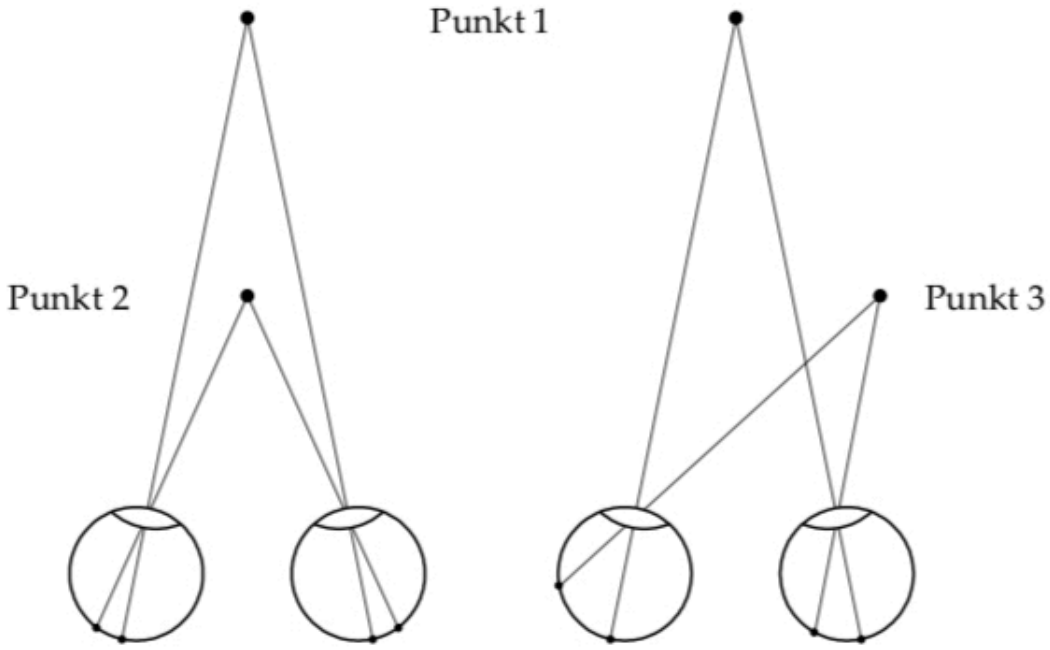

Abbildung 1: Parallaxe nach[Wim04]

2.2 Stereoskopische Panoramen

Ein stereoskopisches Panorama ist eine 360 Grad Rundumsicht einer Szene mit Tiefeneindruck. Dafür werden 2 Panoramen aus leicht unterschiedlichen Positionen aufgenommen, wobei sich dabei meistens an dem durchschnittlichen Augenabstand von 6,5cm orientiert wird. Um bei einem Menschen nun den Tiefeneindruck darzustellen, wird jeweils ein Panorama einem Auge zugeordnet. Eine Herausforderung von stereoskopischen Panoramen besteht in der Aufnahme dynamischer 360 Grad Videos, auf die im Folgenden eingegangen wird. Dabei wird unterschieden ob die Panoramen mit einer oder mehreren Kameras aufgenommen wurden.

2.2.1 Aufnahme mit einer Kamera

Um eine 360 Grad Aufnahme mit einer Kamera zu bewerkstelligen, benötigt man entweder eine spezielle Art von Spiegel oder man benutzt eine Technik die als Mosaicing bekannt ist.
Bei Mosaicing nach [PBE99] und [PPBE00] wird ein Panoramabild erzeugt indem mehrere Aufnahmen zu einem kompletten Panorama zusammengefügt werden. Abbildung 2 verdeutlicht das Grundprinzip von Mosaicing.

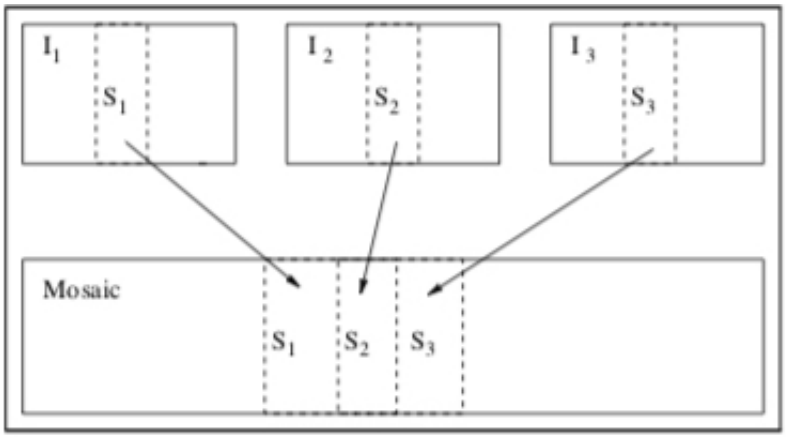

Abbildung 2: Mosaicing nach[PBE99]

Dabei werden die mittleren Streifen aller Bilder zu einem neuen Bild zusammengefügt, wobei alle Bilder von einem Viewpoint aus aufgenommen werden, allerdings mit jeweils anderer Rotation um das optische Zentrum. Für diese Aufnahmen können Videokameras verwendet werden, indem die Kamera während der Aufnahme um ihre Y-Achse gedreht wird. Die resultierenden Frames können dann als Ausgangsbilder für das Mosaicing verwendet werden. Um mithilfe von Mosaicing ein stereoskopisches Panorama-Paar zu erzeugen, muss man statt des Zusammenfügens

der mittleren Streifen jeweils einen Streifen für das rechte und linke Auge verwenden. Wie in Abbildung 3 dargestellt, wird der Streifen für das Panorama des rechten Auges von links des mittleren Streifen aus genommen und der Streifen für das Panorama des linken Auges von rechts des mittleren Streifen aus genommen. Der Abstand des linken Streifens zum rechten Streifen ist gleich des benötigten Augenabstands.

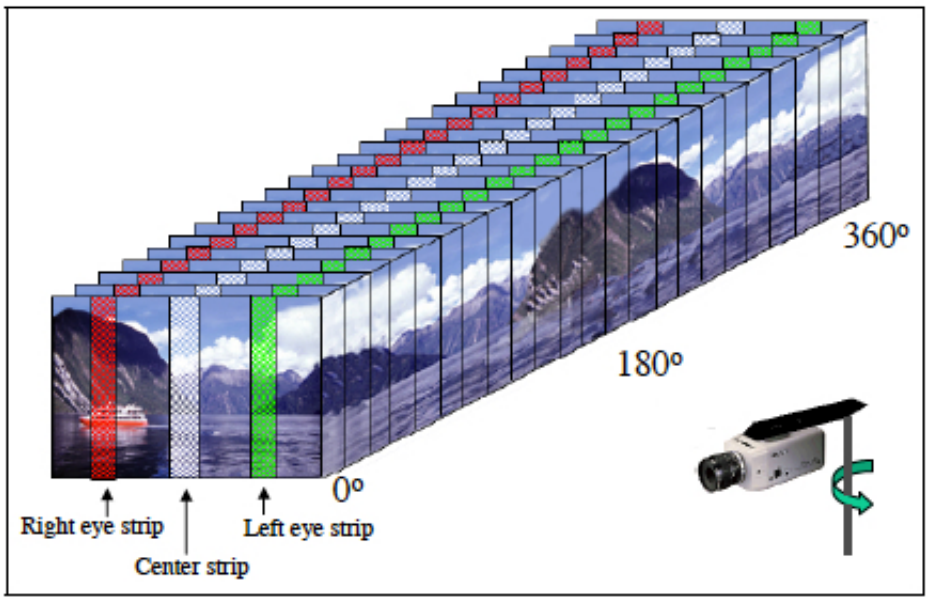

Abbildung 3: stereoskopisches Mosaicing nach[PPBE00]

Diese Art von Mosaicing kann nur bei statischen Szenen für das Erstellen von Panoramen verwendet werden. Für dynamische Szenen existiert eine Art von Mosaicing die Dynamosaicing genannt wird. Die folgende Einführung in Dynamosaicing basiert auf [RAPLP07].
Abbildung 4 stellt die Idee von Dynomosaicing dar. Hierbei wurden die Videoframes durch Rotieren der Kamera von links nach rechts aufgenommen und in einem Space-Time Volumen entlang der Zeitachse gestaffelt. Nun wird das erste Panorama-Mosaik-Frame erstellt, indem von jedem Videoframe entlang der Zeitachse Streifen von der rechten Seite zusammengefügt werden. Diese Streifen werden Appearance-Strips genannt, weil diese die Regionen zeigen, wie sie zuerst im Field of View erscheinen.
Das letzte Panorama-Mosaik-Frame besteht aus Streifen von der linken Seite, den Disappearance-Strips. Diese Streifen zeigen die Regionen bevor sie vom Field of View verschwinden. Zwischen diesen 2 Streifen im Space-Time Volumen werden weitere Streifen zu Mosaiken zusammengesetzt, die den Übergang der Regionen von ihrer Erscheinung (Appearance) zu ihrem Verschwinden (Disappearance) zeigen. Jedes dieser Mosaiken ist ein Panoramabild und zu einem Video zusammengesetzt entsteht

ein dynamisches Panorama. Durch das Zusammensetzen dieser Mosaik-Panoramen wird die chronologische Zeit der Aufnahme durch die Kamera eliminiert. Stattdessen erscheinen alle Regionen gleichzeitig gemäß der lokalen Zeit ihres Sichtbarkeitszeitraums: Von ihrem Erscheinen bis zu ihrem Verschwinden.
Allerdings kann Dynomosaicing nicht für alle dynamischen Szenen verwendet werden und die Input Frames des Videos müssen vorher bearbeitet werden. So dürfen in der Szene keine Objekte vorkommen, die sich unvorhersehbar bewegen wie z.B. ein Mensch.
Für detailliertere Darstellungen von Dynomosaicing sowie für die Vorverarbeitung der Input Frames wird auf [RAPLP07] verwiesen.

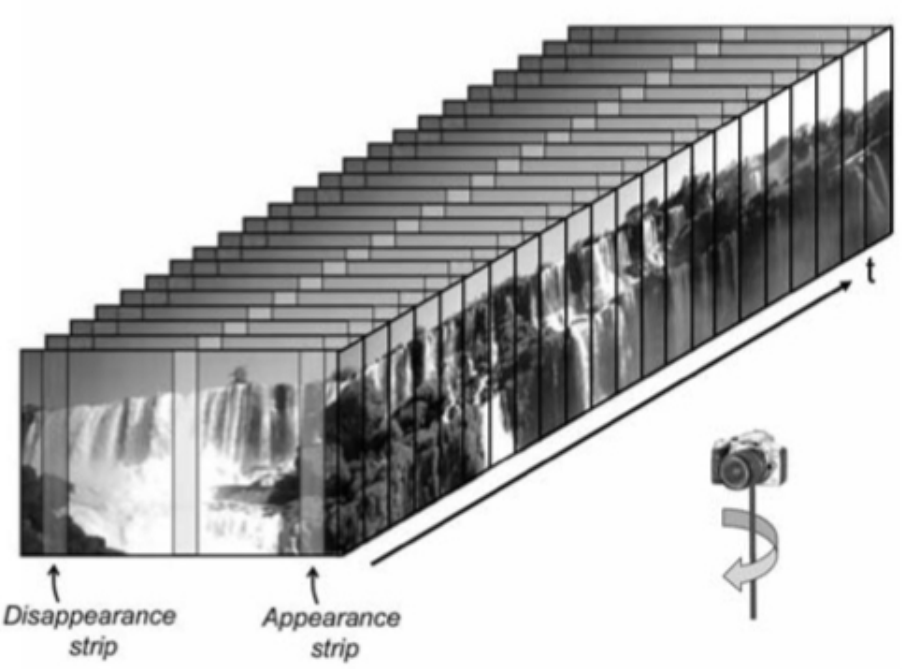

Abbildung 4: Dynamosaicing nach[RAPLP07]

Im folgenden wird eine weitere Technik vorgestellt, die nur eine Kamera benötigt, um ein Panorama zu erstellen, das Aufnehmen mithilfe eines speziellen Spiegels, dem Coffee-Filter Spiegel nach [AVN16].
Abbildung 5 zeigt ein Modell des Spiegels (a), ein Bild des Spiegels in einer Szene (b) und das resultierende Panorama, welches aus (b) entstanden ist (c). Das Ziel dieses Spiegels ist es, alle Strahlen zu erfassen, die tangential zum Sichtkreis liegen. Ein Sichtkreis (Viewing circle) entsteht, wenn ein Punkt P der Szene sowohl vom linken als auch vom rechten Auge wahrgenommen wird. Abbildung 6 (a). Um alle Strahlen zu erfassen, muss der Spiegel für jeden Viewpoint alle tangentialen Strahlen im Uhrzeigersinn

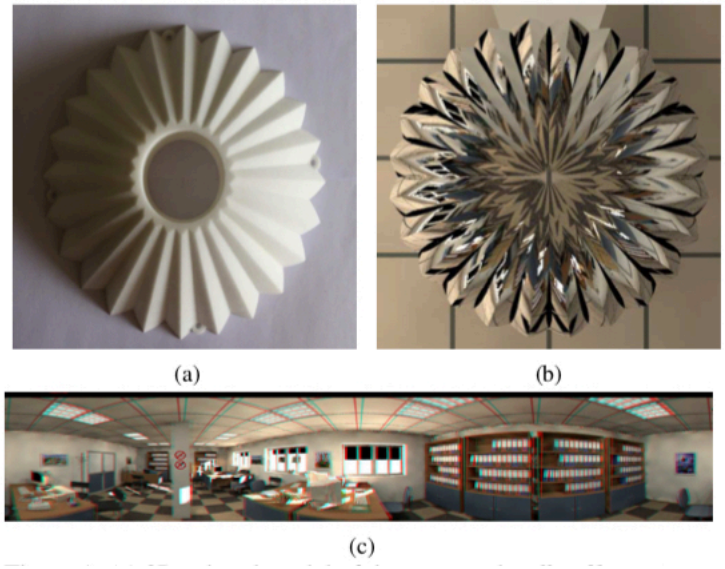

Abbildung 5: Coffee-Filter nach[AVN16]

für das linke Auge abbilden und alle Strahlen gegen den Uhrzeigersinn für das rechte Auge abbilden. Abbildung 6 (b) zeigt eine Anordnung von Spiegeln, um die tangentialen Strahlen für das jeweilige Auge im Sichtkreis zu erfassen. Der Coffee-Filter Spiegel kombiniert diese Spiegelanordnungen zu einem einzigen Spiegel Abbildung 7 . Ein Petal ist dabei die Kombination einer linken Seite (Face) und einer rechten Seite (Face). Die Seiten (Faces)

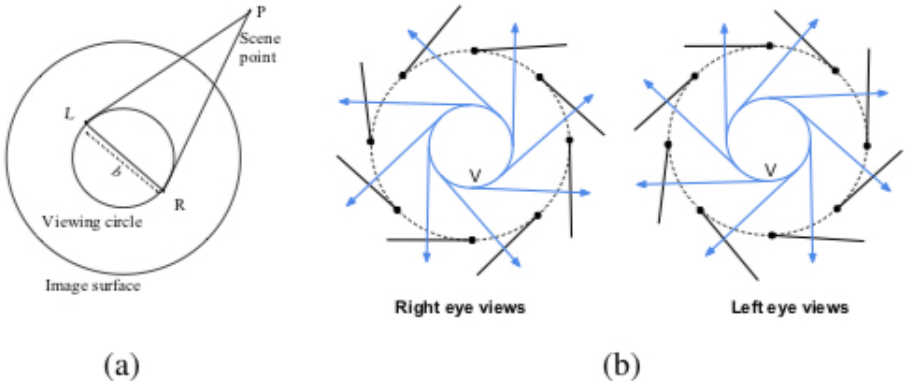

Abbildung 6: Tangentiale Strahlen nach[AVN16]

sind gekrümmt, weil sich dadurch das Sichtfeld vergrößert und tote Winkel vermieden werden, die bei flachen Spiegel auftreten. Für Berechnungen und tiefere Einblicke in den Coffee-Filter Spiegel wird an dieser Stelle auf [AVN16] verwiesen.

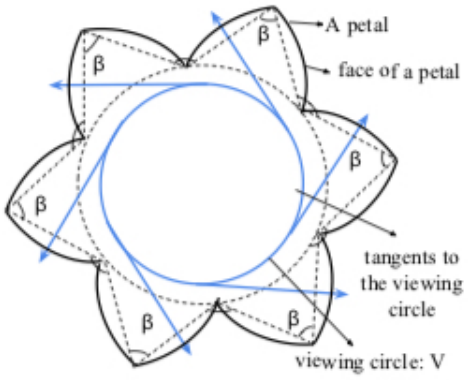

Abbildung 7: horizontale Abbildung des Coffee-Filter Spiegels nach[AVN16]

Zum Abschluss werden Vor-und Nachteile für die Aufnahme eines stereoskopisches Panorama unter Verwendung einer Kamera aufgezeigt.

Vorteile : Es wird für die Aufnahme des Panoramas nur eine Kamera benötigt. Dadurch vermeidet man die notwendige Synchronisation der Aufnahmen bei der Verwendung von mehreren Kameras sowie die Anschaffung mehrerer Kameras.

Nachteile : Mit Mosaicing können nur statische Szenen als Panorama verwendet werden und bei Dynomosaicing nur Szenen deren Bewegungsabläufe berechenbar sind. Alternativ benötigt man einen perfekt verarbeiteten Spiegel, der nicht im Handel erhältlich ist. Bei der Verwendung eines Spiegels müssen außerdem die aufgenommenen Strahlen richtig interpretiert werden.

2.2.2 Aufnahme mit mehrerer Kameras

Eine Alternative zu den Aufnahmetechniken mit einer Kamera bilden Kamerasysteme, die zur Aufnahme des Panoramas eine Anordnung mehrerer Kameras benutzen. Im folgenden wird der Ansatz von [TK02] vorgestellt.
Abbildung 8 zeigt das verwendete Kamera Setup sowie deren Anordnung. Dabei müssen die Kameras fest auf der Halterung sitzen und dürfen sich nicht bewegen. Bei der Verwendung von N Kameras werden nun zu jeder Zeit N Bilder der Umgebung in N verschiedenen Richtungen gleichzeitig vorhanden sein. Aus diesen Bildern können die Panoramen für das linke und rechte Auge konstruiert werden. Die Technik, die dazu verwendet wird, nennt sich Circular Projection.
In Abbildung 9 wird die Idee von Circular Projection dargestellt. Von jedem aufgenommenen Bild und zu jedem Zeitpunkt können die Panoramen für

das jeweilige Auge aus Streifen dieser Bilder erzeugt werden. Der verwendete Streifen wird dabei durch die Tangente zu dem Sichtkreis ermittelt. Durch die Erzeugung von Panoramen aus jedem Frame aller Kameras entsteht ein dynamisches stereoskopisches Panorama. Die besten Ergebnisse würden mit unendlich vielen Kameras erzielt, was allerdings nicht realisiert werden kann. Durch die Kosten der Kameras und den begrenzten Platz auf der Halterung wird die minimale Anzahl an benötigten Kameras bestimmt, um ein gutes Ergebnis zu erzielen.

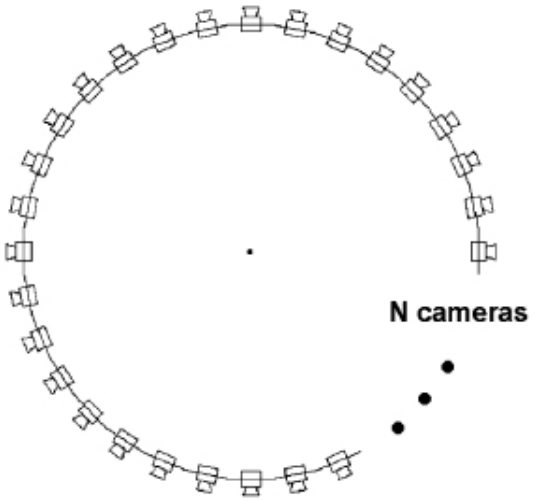

Abbildung 8: Kamera Setup nach[TK02]

Je weniger Kameras benutzt werden, desto größer sind die Streifen, die von jedem Bild für das Panorama verwendet werden und folglich entstehen sichtbare Verzerrungen, wie zum Beispiel Objekte die im Panorama in einem Streifen kontinuierlich erscheinen und in einem anderen nicht kontinuierlich verschwinden. Das liegt an den unterschiedlichen Viewpoints der verschiedenen Kameras, was zur Folge hat, dass die perspektivische Projektion der Kameras unterschiedliche Parameter aufweist.

Um dies zu vermeiden werden die Bilder der Kameras auf einen Zylinder projiziert, bevor die Streifen von den Bildern genommen und zusammengefügt werden. Dabei entspricht der Radius des Zylinders der Brennweite der Kameras und das Zentrum entspricht dem Projektionszentrum der Kamera, dargestellt in Abbildung 10.
Für Berechnungen und mehr Details zu diesem Kamerasystem wird auf [TK02] und [WLO03] verwiesen.

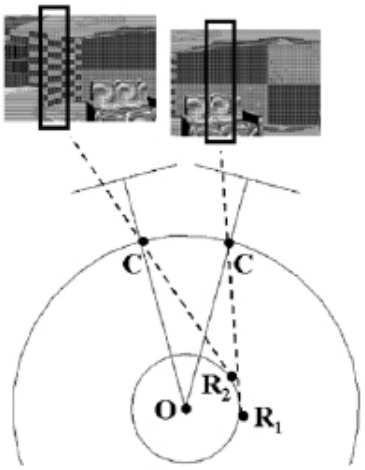

Abbildung 9: Circular Projection nach[TK02]

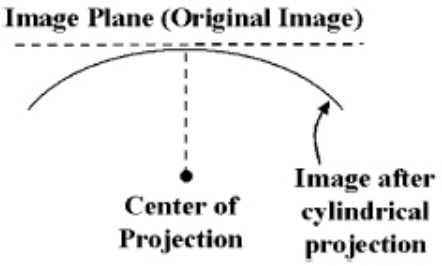

Abbildung 10: Zylindrische Projection nach[TK02]

Zum Abschluss werden Vor-und Nachteile für die Aufnahme eines stereoskopisches Panorama unter Verwendung mehrerer Kameras aufgezeigt.

Vorteile : Jede Szene kann für stereoskopische Panoramen verwendet werden und je nach Qualität der Kameras steigt die Qualität des Panoramas.

Nachteile : Die Anschaffung eines Setup mit mehreren Kameras ist teuer, vor allem wenn die Kameras eine bestimmte Auflösung / Qualität aufweisen sollen.
Bevor die Aufnahmen als stereoskopisches Panorama verwendet werden können, müssen die Bilder zu Panoramen verarbeitet werden.

3 Konzeption

3.1 Konzeption der Videoaufnahmen

Dieser Abschnitt beschäftigt sich mit der Aufnahme und Bearbeitung der stereoskopischen Panoramen bis sie als fertige Videotextur verwendet werden können. Bevor man sich Gedanken zum Aufbau und zum Equipment für die Aufnahme machen kann, müssen bestimmte Regeln berücksichtigt werden. Durch nicht Einhaltung dieser Regeln wird kein(oder ein geringer) stereoskopischer Effekt zu beobachten sein.

3.1.1 Regeln zur stereoskopischen Aufnahme

Bei der Aufnahme von Bildern, die Stereopaare bilden sollen, können viele Fehler auftreten, die erst bei der Wiedergabe wahrgenommen werden. Deshalb sollte im Vorfeld geklärt werden, wie die meisten Aufnahmefehler vermieden werden können. Der folgende Abschnitt basiert auf der Arbeit von [Wim04].
Die Szene muss für rechtes und linkes Auge von einem horizontal unterschiedlichen Standpunkt aus wahrgenommen werden. Dieser Abstand wird in dieser Arbeit Kameraabstand genannt und entspricht im Idealfall dem Augenabstand des Menschen, der die Szene betrachten wird. Da es allerdings nicht möglich ist für jeden Menschen die Szene in einem anderen Augenabstand aufzunehmen, wird sich an dem durchschnittlichen Augenabstand von 6,5cm orientiert. Die Höhe der Standpunkte muss hierbei bei beiden gleich sein, sonst tritt ein sogenannter Höhenfehler auf.
Wenn die Bilder für linkes und rechtes Auge zu unterschiedlichen Zeitpunkten aufgenommen werden, kann es zu retinaler Rivalität kommen, wobei die Bilder der beiden Augen unterschiedliche Objekte enthalten. Retinale Rivalität kann bei den Verfahren auftreten, die zur Aufnahme des

Panoramas nur eine Kamera benutzen oder indem bei synchroner Aufnahme Reflexionen vorhanden sind, die nur in einer Aufnahme zu sehen sind. Um Größenfehler zu vermeiden muss darauf geachtet werden, dass die Frames aller Aufnahmen gleich groß sind. Bei der Aufnahme mit mehreren Kameras sollte deshalb darauf geachtet werden, dass die gleichen Kameras verwendet werden. Auch wenn die gleichen Kameras verwendet werden, sollte man auf die Helligkeits-, Farbton- und Sättigungseinstellungen achten.
Die Kameras müssen parallel zueinander angeordnet sein, damit die Sehachsen beider Augen parallel zueinander verlaufen und keine Divergenz auftritt, das heißt die Sehstrahlen sich mit zunehmender Entfernung nicht voneinander entfernen, wobei eine Divergenz bis zu 1 Grad akzeptabel ist. Die Szene sollte keinen zu großen Tiefenbereich besitzen, da sonst Vorder- und Hintergrund getrennt voneinander wahrgenommen werden, was zur Folge hat, dass entweder Vorder- oder Hintergrund als versetztes Doppelbild erscheint. Dieses Phänomen wird als Bildzerfall bezeichnet.
Unter Berücksichtigung dieser Regeln kann ein Konzept für die Aufnahme der stereoskopischen Panoramen vorgestellt werden.

3.1.2 Kamera

Bevor der Aufbau des Aufnahmesystems vorgestellt werden kann, muss die verwendete Kamera vorgestellt werden, da sich je nach verwendeter Kamera beziehungsweise verwendetem Kamera-Setup der Aufbau verändert.

In dieser Arbeit werden 2 Somikon Kameras verwendet, die jeweils 2 Linsen mit Weitwinkelobjektiven besitzen, um die komplette Szene als Panorama aufzunehmen Abbildung 11. Die Kamera hat eine Videofunktion, die 360 Grad Videos mit 30 Frames pro Sekunde aufnimmt bei einer Auflösung von 1920 zu 960 als MOV File. Für eine komfortable Aufnahme und zur Steuerung kann die Kamera über WLan mit einem mobilen Endgerät verbunden werden. Dafür wird eine App angeboten, die für Android und iOS verfügbar ist. Technische Details zu der Kamera können hier [som] nachgelesen werden.

Abbildung 12 zeigt ein Bild, das mit der Somikon Kamera aufgenommen wurde. Auffällig hierbei ist, dass die Kamera das Bild rund aufnimmt. Dies liegt an dem Objektiv der Kamera, ein sogenanntes Fischaugenobjektiv.
Das Fischaugenobjektiv ist ein spezielles Weitwinkelobjektiv, was eine 180 Grad Aufnahme der Szene ermöglicht, allerdings, im Gegensatz zu einem normalen Weitwinkelobjektiv, ohne übermäßige Verzerrungen. Die vorgestellte Kamera benutzt 2 dieser Fischaugenobjektiven, die gegenüber angebracht sind, um 360 Grad Aufnahmen zu ermöglichen. Die Aufnahmen

Abbildung 11: Somikon Kamera[som]

Abbildung 12: Beispiel einer Aufnahme der Somikon Kamera

dieser Objektive werden zusammen auf einem Frame aufgenommen, wie in Abbildung 12 zu sehen ist.

3.1.3 Aufbau

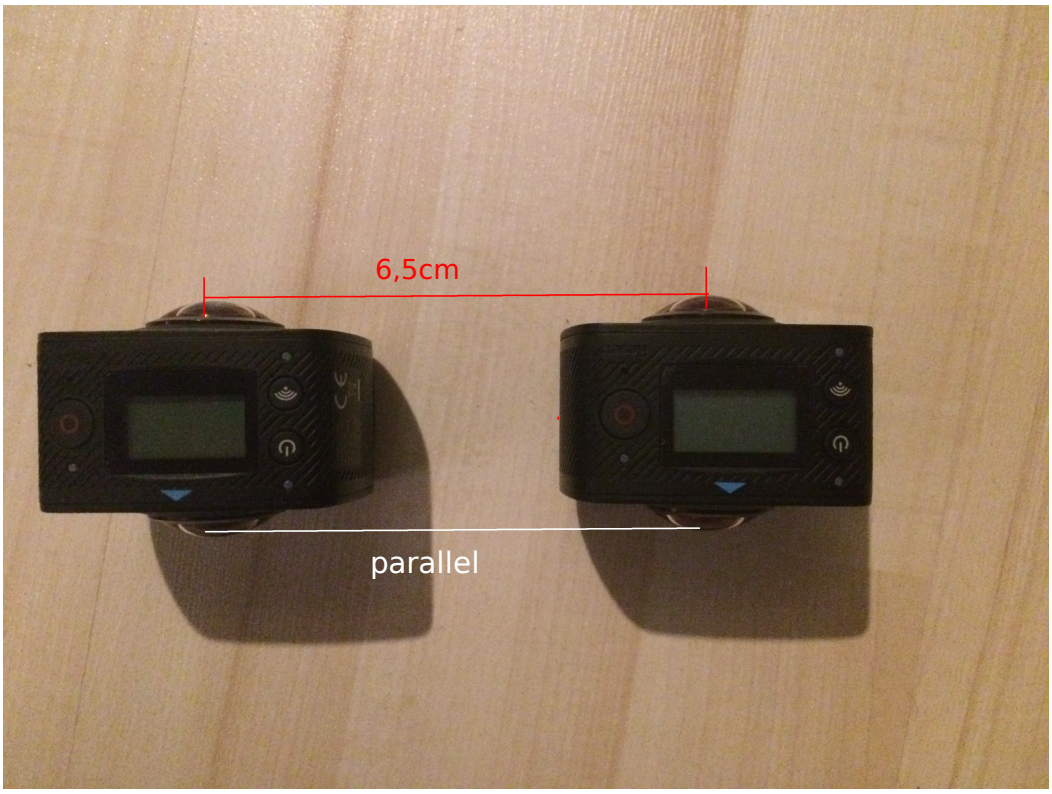

Abbildung 13: Aufbau

Nachdem die verwendete Kamera sowie die Regeln zur Stereoaufnahme vorgestellt wurden, kann der Aufbau zur stereoskopischen Panorama-Aufnahme vorgestellt werden. Abbildung 13 zeigt den groben Aufbau der Kameras. Die Linsen müssen dabei in einem Abstand von zirka 6,5cm parallel angeordnet sein. Die Ausrichtung muss gleich sein und die Kameras dürfen nicht auf einem unterschiedlichem Untergrund stehen, damit die Linsen genau parallel zueinander sind. Kleinste Abweichungen in jede Richtung können den stereoskopischen Effekt stören oder verhindern. Das Paar, das in Abbildung 14 und Abbildung 15 aufgenommen wurde, stand auf einem unebenen Untergrund. Die Linse auf der rechten Seite der linken Kamera in Abbildung 14 steht höher als die Linse auf der linken Seite. Da die Abweichung der Linsen bei der linken Kamera größer ist als die Abweichung der Linsen auf der rechten Seite, steht die linke Kamera schiefer

Abbildung 14: Kamerafehler, Aufnahme der linken Kamera

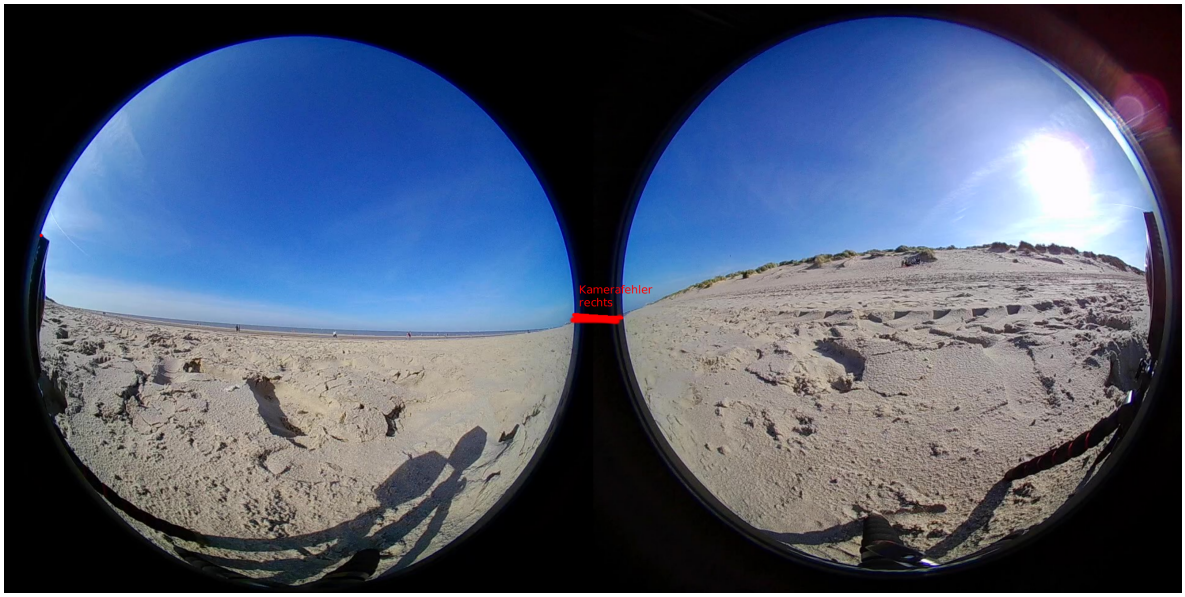

Abbildung 15: Kamerafehler, Aufnahme der rechten Kamera

als die rechte Kamera. Somit sind die Linsen nicht parallel zueinander und es entstehen Fehler.
Die Kameras können alternativ auf einem Untergrund, wie zum Beispiel einer Holzplatte, befestigt werden, um Fehler zu vermeiden und die Aufnahmen beziehungsweise den Aufbau zu beschleunigen. Der Nachteil hierbei ist, dass der Augenabstand fix ist und nicht mehr variabel geändert werden kann.
Zusätzlich sollte die Höhe der Kameras etwa der Körpergröße eines Menschen entsprechen. Als Richtwert wird hierbei 1,75 Meter gewählt. Dadurch befindet sich der Blickpunkt auf das Panorama zirka auf der Höhe eines Menschen, was für den Nutzer den Effekt hat, als würde er in der Szene aufrecht stehen. Würde der Blickpunkt weiter unten liegen, würde dem Nutzer vermittelt, dass er in der Szene sitzt oder liegt.
Generell sollte für eine optimale stereoskopische Aufnahme darauf geachtet werden, dass der Mittelpunkt des Raumes zirka zwischen den zwei Kameras liegt, denn ein gleichmäßiger Abstand von den Kameras zu den Objekten verstärkt den stereoskopischen Effekt.

3.1.4 Synchronisation

Abbildung 16: Synchronisation, Abbildung basierend auf [syn]

Im nächsten Schritt werden die Videos der beiden Kameras synchronisiert. Damit die Videos als stereoskopisches Panorama genutzt werden können, müssen die zueinander passenden Frames beider Videos zur gleichen Zeit auftreten.
Als erstes sollte überprüft werden, ob die Aufnahme gelungen ist, bevor der Aufwand, der mit einer Synchronisation verbunden ist, gestemmt wird. Abbildung 16 stellt die Idee der verwendeten Synchronisation vor. Dabei wird während der Aufnahme ein Synchronisationstartpunkt festgelegt, wie eine markante Bewegung oder ein Zeichen. Wichtig ist, dass dieses Zeichen eindeutig ist.

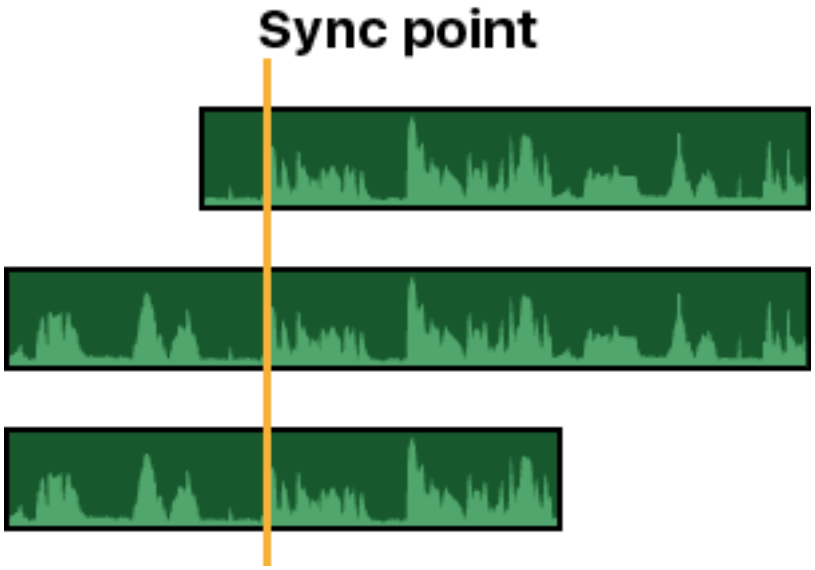

Abbildung 17: Synchronisationsmöglichkeit über die Audiospur [syn]

Abbildung 18: Synchronisationszeichen

Alternativ kann auch, wie in Abbildung 17 , ein lauter Ton als Synchronisationstartpunkt gewählt werden, da die Kameras über eine Tonaufnahme verfügen. Die verwendete Software zum Synchronisieren sollte dann allerdings auch Audiospuren unterstüzten.
Im Zweifelsfall ist eine Synchronisation über die Audiospur einfacherer, da als Startpunkt der lauteste Moment des Tons, also der Peak der Audiospur, verwendet werden kann. In dieser Arbeit wurden die Panoramavideos über ein Zeichen im Bild synchronisiert, welches in Abbildung 18 dargestellt wird. Um dieses Zeichen in beiden Videos wiederzufinden und als Startpunkt auszuwählen, können gängige Videobearbeitungsprogramme wie zum Beispiel Adobe Premiere Pro verwendet werden. [wob]

3.2 Konzeption des Virtual-Reality Renderer

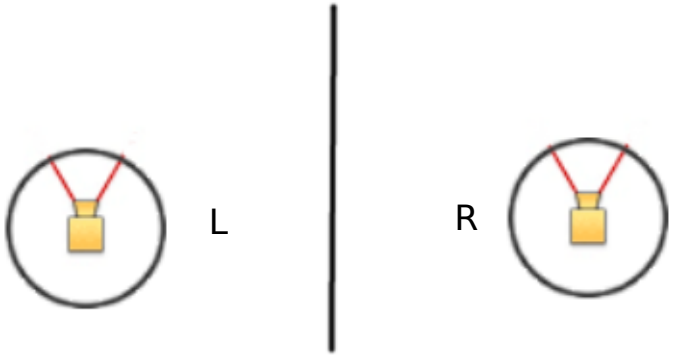

Abbildung 19: Renderkonzept

Das aufgenommene Panorama soll nun mithilfe einer Virtual-Reality Brille stereoskopisch dargestellt werden. Abbildung 19 führt das verwendete Konzept ein. Die Grundidee dabei ist, dass 2 Szenen unabhängig voneinander gerendert werden. Eine Szene für das linke Auge und eine Szene für das rechte Auge der Virtual-Reality Brille. Die Szene beinhaltet jeweils eine Kugel, auf welche die Panorama Textur gelegt wird und eine lokale Kamera, die durch die Bewegungen der globalen Kamera (durch Bewegungen der Virtual-Reality Brille) verändert wird. Beide lokalen Kameras sollen sich dabei gleich verhalten. Der Vorteil durch das Verwenden 2 separater Szenen ist die Vermeidung von Störbildern. Störbilder entstehen, wenn das linke Auge Teile der rechten Szene wahrnimmt oder umgekehrt. Damit dieses Konzept funktioniert, müssen die Panoramen richtig auf die Kugel gerendert werden. Wenn die Panoramen vertauscht angebracht werden, entstehen Pseudostereoskopische Abbildungen.
Nach [Wim04] ergeben vertauschte Stereopaare kein sinnvolles Raumbild. Um dies zu vermeiden, muss sich der Aufbau der Kameras genauer angeschaut werden: In Abbildung 20 sind die Kameraobjektive mit jeweils der

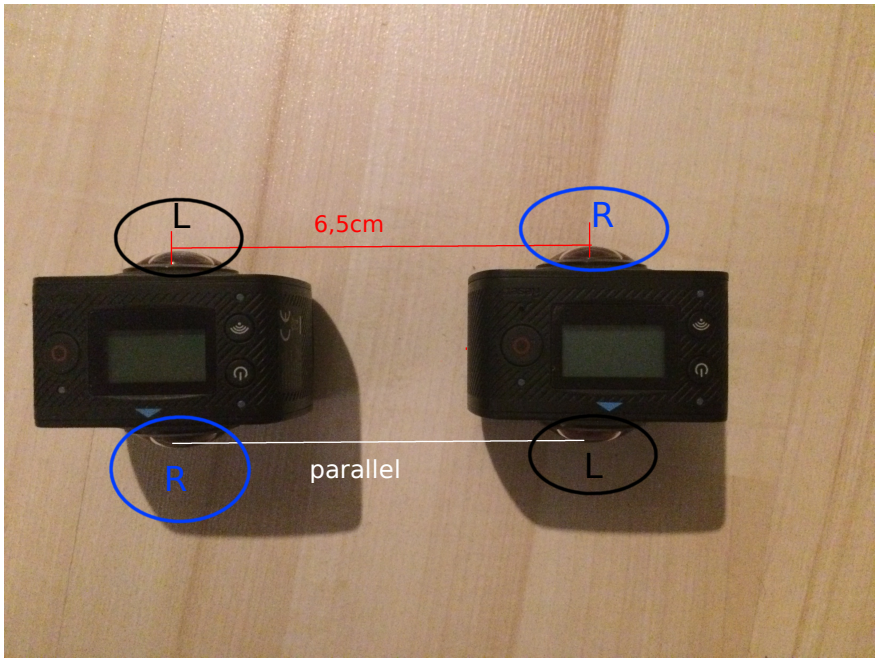

Abbildung 20: Ausrichtung der Kameras im Detail

Seite markiert, die der Seite in der Szene entspricht d.h. die linke Kamera nimmt nicht die Szene für das gesamte linke Auge auf, sondern eine Hälfte der Szene muss auf das linke Auge gerendert werden, während die andere Hälfte auf das rechte Auge gerendert werden muss. Dadurch beinhaltet ein Video einer Kamera sowohl eine linke Hälfte als auch eine rechte Hälfte des Stereopaars . Der Grund daür ist, dass sich die Augen des Menschen mitbewegen, wenn man von einer Raumhälfte in die andere schaut. Die Kameras hingegen sind stationär, ihre Blickrichtung bleibt gleich. Um die richtigen Teile des Panoramas auf die jeweilige Seite zu bringen, werden an dieser Stelle 2 Möglichkeiten vorgestellt :
In der Ersten werden die 2 Videopaare aufgeteilt. Das Video wird dabei genau in der Mitte getrennt, sodass 2 Videos entstehen. Die Aufnahme in Abbildung 12 wurde zur Verdeutlichung in Abbildung 21 und Abbildung 22 gesplittet. Dabei ist Abbildung 21 die Vorderseite, die zur linken Szene gehört und Abbildung 22 ist die Rückseite der rechten Szene. Bei der 2. Kamera sind die Hälften des Videos genau gegenteilig angeordnet, sodass die Kugeln der jeweiligen Szenen voll texturiert sind.
Die 2. Möglichkeit beinhaltet einen aktiven Wechsel der beiden Szenen, je nachdem in welche Richtung die Virtual-Reality Brille zeigt. Als Referenzpunkt wird hierbei der Mittelpunkt der Virtual-Reality Brille genommen, also der Punkt der sich zwischen rechtem und linkem Auge befindet. Wenn dieser Punkt in Richtung vorderer Raumhälfte gerichtet ist, wird die linke

Abbildung 21: Teil 1 des gesplitteten Panoramas

Abbildung 22: Teil 2 des gesplitteten Panoramas

Szene auf das linke Auge und die rechte Szene auf das rechte Auge gerendert. Sollte sich der Punkt in der hinteren Raumhälfte befinden, werden die Szenen für die Augen vertauscht. Da diese Methode in der Arbeit verwendet wird, wird sie in Unterabschnitt 4.2 detaillierter behandelt.

4 Implementation

Abbildung 23: Oculus Rift [ocu]

In diesem Kapitel wird aus dem vorgestellten Konzept ein Virtual-Reality

Renderer für das Darstellen von stereoskopischen Video-Panoramen entwickelt. Zum Testen dieses Renderers wurde eine Oculus Rift verwendet. Abbildung 23

4.1 Wahl der Programmierumgebung

Der erste Schritt zur Implementierung eines Virtual-Reality Renderer ist die Wahl einer geeigneten Programmierumgebung.Die Wahl fällt auf mehrere Bibliotheken oder Game Engines mit Virtual-Reality Unterstützung, die im folgenden gegenüber gestellt werden.

Unreal Engine 4 [unr] Die Unreal Engine 4 ist eine mächtige Game Engine von Epic Games mit Virtual-Reality Unterstützung. Der Vorteil der Unreal Engine 4 ist, dass Funktionen vorimplementiert sind wie z.B. Funktionen zur Steuerung der Kamera und man sich somit einiges an Arbeit spart.
Allerdings ist die Unreal Engine 4 ungeeignet für die Verwendung als Renderer dieser Arbeit, da sie keinen direkten Zugriff auf die einzelnen Augen der Virtual-Reality Brille gewährt, sondern die Bilder der einzelnen Augen aus den Szenenkoordinaten berechnet.

Oculus SDK [SDK] Das Oculus SDK ist eine Bibliothek, die Funktionalitäten besitzt, um in C++ Programme für Virtual Reality Geräte zu schreiben. Da es sich um eine C++ Bibliothek handelt, bleiben sämtliche Freiheiten erhalten, was aber auch mit einem höheren Arbeitsaufwand verbunden ist. Außerdem unterstützt das Oculus SDK offiziell nur die Oculus Rift, bei anderen Virtual-Reality Geräten kann es zu Fehlern kommen.

Unity [uni] Unity ist eine Game Engine ähnlich der Unreal Engine 4. Dadurch besitzt auch sie vorimplementierte Funktionen, die mit einer Zeitersparnis einher gehen. Im Gegensatz zu der Unreal Engine 4 kann allerdings das Oculus SDK als Add-On eingebunden werden, was einen Zugriff auf die einzelnen Augen der Virtual-Reality Brille erlaubt. Unity ist zwar mit mehreren Virtual-Reality Brillen kompatibel, kann allerdings durch die Verwendung des Oculus SDK bei anderen Virtual-Reality Brillen außer der Oculus Rift Fehler auftreten lassen.

OpenVR [Vala] OpenVR ist eine C++ Bibliothek ähnlich des Oculus SDK mit dem Unterschied, dass es mit den meisten Virtual-Reality Brillen kompatibel ist. Durch diese Kompatibilität und die Freiheiten beim Programmieren wird in dieser Arbeit die OpenVR Biblitohek zur Implementation des Renderer verwendet.
Dafür wurde als Ausgang ein OpenGL Beispielprogramm von OpenVR

verwendet, das texturierte Würfel rendert [Valb] .
Abbildung 24 zeigt die gerenderte Szene dieses Beispielprogramms. Da

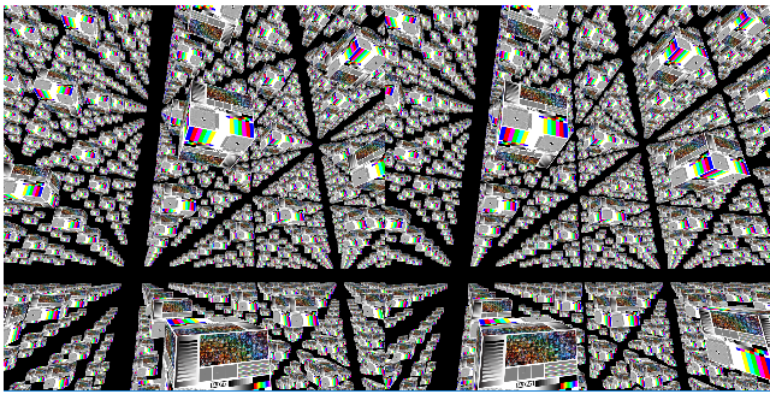

Abbildung 24: OpenVR Beispielprogramm, Ausgangsprogramm

dies als Ausgangsbasis für den Panorama Renderer dient, kann es vorkommen, dass Codebruchstücke des Beispielprogramms in dieser Arbeit zu finden sind. Diese Bruchstücke sind geistiges Eigentum von Valve. [Valb]

4.2 Implementation des Renderer

```
if (nEye == Left) {
    glUseProgram(m_unSceneProgramID);
    glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
    glBindVertexArray(m_unSceneVAOl);
    glBindTexture(GL_TEXTURE_2D, m_iTextureL);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, m_u
    glBindVertexArray(0);
}
else {
    glUseProgram(m_unSceneProgramID);
    glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
    glBindVertexArray(m_unSceneVAO);
    glBindTexture(GL_TEXTURE_2D, m_iTexture);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, m_uiVertcount);
    glBindVertexArray(0);
}
```

GLuint CMainApplication::m_iTextureL

Abbildung 25: stereoskopisches Rendering der Szenen

Der erste Schritt zur Implementierung des Renderer ist die Erstellung einer Ausgabe für linkes und rechtes Auge in 2 separaten Renderschritten. Abbildung 25 zeigt die dafür relevanten Codezeilen. Wenn das linke Auge angesprochen wird, also wenn nEye=links gilt, soll die linke Szene gerendert werden. Die relevanten Vertexdaten der linken Szene liegen in dem m_unSceneVAOl Vertex Array. Die Textur der linken Szene wird an m_iTextureL gebunden. Mit glDrawArrays wird diese Szene nun mit ei-

```
// Left Eye
glBindFramebuffer( GL_FRAMEBUFFER, leftEyeDesc.m_nRenderFramebufferId );
glViewport(0, 0, m_nRenderWidth, m_nRenderHeight );
RenderScene( vr::Eye_Left );
glBindFramebuffer( GL_FRAMEBUFFER, 0 );
```

Abbildung 26: Rendern der linken Szene auf eine Textur

```
// Right Eye
glBindFramebuffer( GL_FRAMEBUFFER, rightEyeDesc.m_nRenderFramebufferId );
glViewport(0, 0, m_nRenderWidth, m_nRenderHeight );
RenderScene( vr::Eye_Right );
glBindFramebuffer( GL_FRAMEBUFFER, 0 );
```

Abbildung 27: Rendern der rechten Szene auf eine Textur

```
glBindTexture(GL_TEXTURE_2D, leftEyeDesc.m_nResolveTextureId );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glDrawElements( GL_TRIANGLES, m_uiCompanionWindowIndexSize/2, GL_UNSIGNED_SHORT, 0 );

// render right eye (second half of index array )
glBindTexture(GL_TEXTURE_2D, rightEyeDesc.m_nResolveTextureId  );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glDrawElements( GL_TRIANGLES, m_uiCompanionWindowIndexSize/2, GL_UNSIGNED_SHORT, (const void *)(uintptr_t)(m_uiCompanionWindowIndexSize) );
```

Abbildung 28: Rendern der Texturen

nem Framebuffer auf eine Textur gerendert (Abbildung 26).
Das Gleiche wird mit dem rechten Auge gemacht, nur dass eine andere Szene und eine andere Textur verwendet werden und die rechte Szene auf eine separate Textur gerendert wird (Abbildung 27).
Diese Texturen werden dann an die Virtual-Reality Brille übertragen und für linkes und rechtes Auge gerendert.
Nachdem der Basisrenderer steht, werden als Nächstes die Szenen erstellt und somit die Vertex Arrays mit Vertexdaten gefüllt. Abbildung 29 zeigt

```
void CMainApplication::SetupSceneL()
{
    //Scene for right eye
    if (!m_pHMD)
        return;
    std::vector<float> vertdataarray;


    Matrix4 matScale;
    matScale.scale(m_fScale, m_fScale, m_fScale);
    Matrix4 matTransform;
    matTransform.translate(
        ((float)0),
        ((float)0),
        ((float)1));

     mat = matScale * matTransform - m_mat4eyePosLeft.rotateX(-90);

    createSphere(70, 0, 0, 0, vertdataarray, mat);
```

Abbildung 29: Erstellen der Szene

das Setup der Szene für das linke Auge, welches eine Kugel rendert und in den Kameraursprung verschiebt. Die Funktion createSphere erstellt Vertices und Texturkoordinaten zum Rendern einer Kugel. Beide werden innerhalb der Funktion mit der Modelmatrix mat transformiert und dann in vertdataarray geschrieben. Dabei werden die Vertices und die Texturkoordinaten für einen Punkt direkt hintereinander angegeben und mithilfe eines Offset separat in den Shader geladen (Abbildung 30).

Die ersten 4 Werte von createSphere geben an, wie viele Unterteilungen durchgeführt werden sollen und um welchen x,y,z - Wert die Kugel verschoben werden soll. Da die Kugel durch die Modelmatrix verschoben wird, werden für alle 3 Werte 0 angegeben. Um die Kugel in den Kameraursprung zu verschieben, wird die Kameraposition des linken Auges m_mat4eyePosLeft von allen Vertices subtrahiert.
Der Unterschied zu der Szene für das rechte Auge liegt in der Kamera Position. Dort wird die Kameraposition des rechten Auges m_mat4eyePosRight

```
m_uiVertcount = vertdataarray.size() / 5;

//VAO to draw the scene
glGenVertexArrays(1, &m_unSceneVAO1);
glBindVertexArray(m_unSceneVAO1);

glGenBuffers(1, &m_glSceneVertBuffer);
glBindBuffer(GL_ARRAY_BUFFER, m_glSceneVertBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertdataarray.size(), &vertdataarray[0], GL_STATIC_DRAW);

GLsizei stride = sizeof(VertexDataScene);
uintptr_t offset = 0;

//loads the vertices with index 0 and size 3
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, stride, (const void *)offset);

// add offset from vertex data (first 3 vertices then 2 uv coord)
offset += sizeof(Vector3);
//loads the uv coordinates

glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, stride, (const void *)offset);

glBindVertexArray(0);
glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
```

Abbildung 30: Binden der Vertices an den Szenenbuffer

```
glBindTexture(GL_TEXTURE_2D, m_iTextureL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

GLfloat fLargest;
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &fLargest);
glTexParameterf(GL_TEXTURE_2D, GL_LINEAR, fLargest);



//binds the actual frame to a texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, nImageWidth, nImageHeight,
    0, GL_BGR, GL_UNSIGNED_BYTE, imageRGBA.data);


glGenerateMipmap(GL_TEXTURE_2D);


glBindTexture(GL_TEXTURE_2D, 0);
```

Abbildung 31: Binden der Textur

```
// Vertex Shader
"#version 410\n"
"uniform mat4 matrix;\n"
"layout(location = 0) in vec4 position;\n"
"layout(location = 1) in vec2 v2UVcoordsIn;\n"
"layout(location = 2) in vec3 v3NormalIn;\n"
"out vec2 v2UVcoords;\n"
"void main()\n"
"{\n"
"   v2UVcoords = v2UVcoordsIn;\n"
"   gl_Position = matrix * position;\n"
"}\n",

// Fragment Shader
"#version 410 core\n"
"uniform sampler2D mytexture;\n"
"in vec2 v2UVcoords;\n"
"out vec4 outputColor;\n"
"void main()\n"
"{\n"
"   outputColor = texture(mytexture, v2UVcoords);\n"
"}\n"
```

Abbildung 32: Shader für das Rendern der Szene

von allen Vertices subtrahiert. Nachdem die Vertices und Texturkoordinaten für die Kugel berechnet wurden, muss die Textur gebunden werden. Abbildung 31 zeigt das Binden einer Textur für das linke Auge, wobei die Texturdaten in imageRGBA liegen. Mehr zu Texturen wird in Unterabschnitt 4.3 behandelt.
Eine funktionierende Kamera ist in OpenVR vorhanden und war im Beispielprogramm [Valb] schon implementiert. Sie erlaubt eine Rotation der Kamera in alle Richtungen.
Die benötigten Shader enthalten Standardoperationen, dargestellt in Abbildung 32 . Im Vertex Shader muss lediglich die Position mit der View-, Projection- und Modelmatrix transformiert werden, die hier schon alle in der Variablen matrix zusammengefasst wurden. Im Fragment Shader wird der Farbe mithilfe der UV-Koordinaten der passende Texturwert zugeordnet.

4.3 Einbindung der Videotextur

Da keine statischen Panoramen dargestellt werden sollen sondern dynamische, wird in diesem Kapitel darauf eingegangen, wie das aufgenommene Video als Textur gerendert werden kann. Um das Video in das C++ Programm zu laden, wird OpenCV [ope] verwendet. OpenCV ist eine Bibliothek mit Algorithmen und Datenstrukturen für Bildverarbeitung und

Computergrafik und wurde von Intel und Wilow Garage entwickelt. Sie wird auf neusten Forschungsergebnissen aufgebaut und ist somit von der Menge an Algorithmen als auch von der Geschwindigkeit her aktuell. In

```
VideoCapture video("../bin/1R.mov");
```

Abbildung 33: Laden des Videos in die Video Capture Datenstruktur

```
VideoCapture videoL("../bin/1L.mov");
```

Abbildung 34: Laden des 2.Videos in die Video Capture Datenstruktur

Abbildung 33 undAbbildung 34 werden die Videos für linkes und rechtes Auge in die OpenCV Datenstruktur VideoCapture geladen. Die geladenen Videoframes müssen nun während der Laufzeit als Kugeltextur verwendet werden. Dafür werden die aktuellen Videoframes an die Textur

```
while ( !bQuit )
{
    bQuit = HandleInput();

    //for each frame another texture from the video
    SetupTexturemaps(video);
    SetupTexturemapsL(videoL);
    RenderFrame();
}
```

Abbildung 35: Binden der Videotexturen

gebunden, bevor der Frame gerendert wird, dargestellt in Abbildung 35. Die Datenstruktur, die das geladene Video beinhaltet, wird hierbei an die Funktion übergeben, die das Binden der Textur des aktuellen Videoframes übernimmt.

In Abbildung 36 wird der aktuelle Videoframe aus der OpenCV Datenstruktur gelesen, wenn das Video erfolgreich geladen und geöffnet wurde. Die Funktion video.read(imageRGBA) liest den vordersten Frame und speichert diesen in imageRGBA. Dabei merkt sich die Funktion, welcher Frame als Letztes ausgelesen wurde und gibt automatisch den Nächsten aus. Über video.get(CV_CAP_PROP_FRAME_WIDTH) wird die Breite der Videoframes und über video.get(CV_CAP_PROP_FRAME_HEIGHT) die Höhe der Videoframes ermittelt.
Da die verwendete Videokamera zur Aufnahme der Panoramen die Videos mit 30 Fps aufnimmt, müssen sie im Renderer auch mit 30 Fps abgespielt werden. OpenCV stelt hierfür eine Funktion bereit, wie in Abbildung 38 zu sehen ist. Mit video.set(CV_CAP_PROP_FPS, 30) wird die maximale Fps

```
//shutdown the program if the video file cant be loaded
if (video.isOpened() == false) {
    Shutdown();
    return 0;
}


//stores the actual frame data
Mat imageRGBA;
//read the actual frame
bool bSuccess = video.read(imageRGBA);

if (!bSuccess)
{
    std::cout << "Cannot read a frame from video stream" << std::endl;

}

//load the frame width and height
unsigned nImageWidth = video.get(CV_CAP_PROP_FRAME_WIDTH), nImageHeight = video.get(CV_CAP_PROP_FRAME_HEIGHT);
```

Abbildung 36: Lesen des aktuellen Videoframes

```
  //if looking backwards, change the texture from left and right eye
  float angleY = 5;
  angleY = atan2(-m_mat4HMDPose[8], sqrt((m_mat4HMDPose[9] * m_mat4HMDPose[9] + m_mat4HMDPose[10] * m_mat4HMDPose[10])));
  dprintf("Winkel %f \n", angleY);
  if(angleY > 0){
  if (nEye == Left) {
      glUseProgram(m_unSceneProgramID);
      glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
      glBindVertexArray(m_unSceneVAOl);
      glBindTexture(GL_TEXTURE_2D, m_iTextureL);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, m_uiVertcount);
      glBindVertexArray(0);
  }
  else {
      glUseProgram(m_unSceneProgramID);
      glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
      glBindVertexArray(m_unSceneVAO);
      glBindTexture(GL_TEXTURE_2D, m_iTexture);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, m_uiVertcount);
      glBindVertexArray(0);
  }
  }
  if (angleY <= 0) {
      if (nEye == Left) {
          glUseProgram(m_unSceneProgramID);
          glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
          glBindVertexArray(m_unSceneVAOl);
          glBindTexture(GL_TEXTURE_2D, m_iTexture);
          glDrawArrays(GL_TRIANGLE_STRIP, 0, m_uiVertcount);
          glBindVertexArray(0);
      }
      else {
          glUseProgram(m_unSceneProgramID);
          glUniformMatrix4fv(m_nSceneMatrixLocation, 1, GL_FALSE, GetCurrentViewProjectionMatrix(nEye).get());
          glBindVertexArray(m_unSceneVAO);
          glBindTexture(GL_TEXTURE_2D, m_iTextureL);
          glDrawArrays(GL_TRIANGLE_STRIP, 0, m_uiVertcount);
          glBindVertexArray(0);
      }
  }
```

Abbildung 37: Tauschen der Texturen

```
//load video for right eye and set fps to 30
VideoCapture video("../bin/1R.mov");
video.set(CV_CAP_PROP_FPS, 30);
SetupTexturemaps(video);

//Video for the left eye and set fps to 30
VideoCapture videoL("../bin/1L.mov");
videoL.set(CV_CAP_PROP_FPS, 30);
SetupTexturemapsL(videoL);

while ( !bQuit )
{
    bQuit = HandleInput();

    //for each frame another texture from the video
    SetupTexturemaps(video);
    SetupTexturemapsL(videoL);
    RenderFrame();
}
```

Abbildung 38: Setzen der Fps

eingestellt, mit der gerendert wird. Nun wird das Video auf der Kugel in der Virtual-Reality Brille für rechtes und linkes Auge separat gerendert. Wie in Unterabschnitt 3.2 erwähnt wurde, sind die Videotexturen der Augen aber für eine Hälfte der Szene verdreht. Dadurch muss die Textur von rechtem und linkem Auge getauscht werden, wenn in diese Richtung geschaut wird (Abbildung 37). Die Float Variable angleY steuert hierbei die Ausrichtung der Kamera. Wenn die Kamera in die vordere Hälfte der Szene schaut, ist angleY positiv und die linke Textur wird auf das linke Auge gerendert. Wenn die Kamera in die hintere Hälfte der Szene schaut, ist angleY negativ und die linke Textur wird auf das rechte Auge gerendert. Der Wert von angleY wird über eine Funktion berechnet, die die Rotation der Virtual-Reality Brille um die Y-Achse in einem Float -Wert darstellt.
Dabei enthält m_mat4HMDPose die aktuelle Pose-Matrix der Virtual-Reality Brille.
Somit werden die Videos auf der Kugel erfolgreich mit 30 Fps und stereoskopisch dargestellt. Sobald das Ende des Videos erreicht ist, wird das Programm geschlossen.

5 Bewertung

In diesem Kapitel wird das entwickelte System zur Erstellung stereoskopischer Video-Panoramen mit Google Jump [Goo] in Bildqualität, Immersion und Benutzung verglichen, einem High-end System, was zur Erstellung solcher Panoramen im Hinblick auf Videoqualität entwickelt wurde. Goo-

Eigenschaft	Somikon	Yi Halo
optische Sensorauflösung	4 Megapixel	12 Megapixel
Auflösung	1920 x 960 Pixel bei 30 fps	bis zu 8192x8192 Pixel bei 30-60 fps
Videoformat	MOV	H.264codec, .mp4
Wifi	Ja	Ja
Audioaufnahme	Ja	Ja
Akkubetrieb	bis zu 70min Aufnahme	bis zu 100min Aufnahme
Appsteuerung	Android und IOS	Android
Größe	62 x 51 x 47 mm	287 x 287 x 192 mm
Gewicht	106g	< 3650g
Preis	ca 150 Euro für beide Kameras	ca 19 999 Euro für das System

Tabelle 1: Vergleich der verwendeten Somikon Kamera mit der Yi Halo [som], [YiR]

gle Jump funktioniert ähnlich des Systems, das in Unterunterabschnitt 2.2.2 vorgestellt wurde. Die Kameras sitzen auf einem festen Gehäuse, welches die Aufnahmen der verschiedenen Kameras selbständig zu einem Panoramavideo für rechtes und linkes Auge zusammensetzt.

Abbildung 40 verdeutlicht das Konzept für das Zusammensetzen eines Panoramas aus den Videodaten der Kameras. Die entstandenen Panorama-

Abbildung 39: Google Jump Kamera Yi Halo [Goo]

Videos können mithilfe des VR-Youtube Players abgespielt werden. Mit Google Jump aufgenommene Video-Panoramen sind in der von Google erstellten Youtube Playlist [You] zu finden.

Abbildung 40: Zusammensetzen eines Panorama-Videos von Google Jump [Goo]

5.1 Bildqualität

Als Nächstes werden die erstellten stereoskopischen Panorama-Videos hinsichtlich ihrer Bildqualität miteinander verglichen. Dazu werden zunächst die Somikon Kamera und die Yi Halo gegenübergestellt. Die Yi Halo [YiR] ist eine Google Jump Kamera, die aus 17 Kameras besteht.
In Tabelle 1 sind wichtige Eigenschaften beider Kameras aufgelistet, die auch Einfluss auf die aufgenommene Bildqualität haben. Auffallend hierbei ist, dass die technischen Eigenschaften der Yi Halo besser sind, was im Folgenden erläutert wird :
Die optische Sensorauflösung der Yi Halo ist mit 12 Megapixeln um ein 3-faches höher als die der Somikon Kamera. Dadurch können die Bilder schärfer und größer dargestellt werden, weshalb die Auflösung der Yi Halo bis zu 8192 x 8192 Pixel beträgt und die der Somikon nur 1920 x 960 Pixel. Zusätzlich dazu kann die Yi Halo Videos mit bis zu 60 Fps aufnehmen, also mit 30 Fps mehr als die Somikon, allerdings verringert sich die Auflösung dabei auf 5760 x 5760 Pixel. Eine höhere Fps sorgt für flüssigere Bewegungsablaüfe innerhalb des Videos. So können zum Beispiel ruckartige Bewegungen als störend empfunden werden.
Mit Abbildung 41 und Abbildung 42 werden 2 Aufnahmen gegenübergestellt, wobei das eine Panorama von der Yi Halo und das andere von der Somikon aufgenommen wurde. Es fällt auf, dass die Bildfläche des Panorama der Yi Halo weiter gefasst ist. Der Grund hierfür ist, dass die Somikon das Panorama mit einem Weitwinkelobjektiv aufnimmt, wodurch die Bildfläche kugelförmig ausfällt und Teile der Szene somit leicht gekrümmt sein können. Des Weiteren nehmen sich die Somikon Kameras inklusive ihrer Schatten gegenseitig auf, wenn stereoskopische Aufnahmen gemacht werden. Dies geschieht bei der Yi Halo nicht, da dort 17 Kameras mit normalen Objektiven verwendet werden, wobei die Kameras außerhalb des Sichtbereichs ihrer Nachbarn liegen.

Abbildung 41: Aufnahme eines Panorama-Video Frames von Google Jump mit der Yi Halo, dargestellt als Anaglyphe [ASG+]

Abbildung 42: Aufnahme eines Panorama-Video Frames der Somikon

5.2 Immersion

Ein weiterer wichtiger Faktor zur Bewertung der beiden Systeme ist die Immersion, auf die die Bildqualität auch indirekt Einfluss ausübt. Denn je besser die Aufnahme der Videos ist, desto besser kann die Immersion sein. Des Weiteren ist die Immersion abhängig von der Qualität des stereoskopischen Effekts.
Für den Vergleich wurde ein mit der Yi Halo aufgenommenes Video aus der Google Youtube Playlist genommen ([You]) und mithilfe einer Oculus Rift wiedergegeben. Videos, die sowohl durch das Setup mit der Somikon Kamera als auch mit der Yi Halo aufgenommen wurden, besitzen einen stereoskopischen Effekt und den damit zusammenhängenden Tiefeneindruck. Allerdings ist der stereoskopische Effekt bei beiden Systemen nicht ideal, da beide die Videos mit einem bestimmten Augenabstand erstellen müssen, der Augenabstand eines jeden Menschen aber unterschiedlich ist. Somit müsste für jeden Menschen ein eigenes Video erstellt werden mit seinem perfekten Augenabstand.
Ein Unterschied besteht trotzdem zwischen dem stereoskopischen Effekt der beiden Kameras, da die Somikon Kameras sich gegenseitig aufnehmen. Die linke Kamera nimmt die Rechte auf und umgekehrt. Somit gibt es bei den Videos der Augen jeweils ein Objekt, welches nur auf dem Video des einen Auges auftaucht. Dieses Objekt kann somit keinen Stereoeffekt darstellen. Dies tritt an rechter und linker Seite des Panoramas auf.
Das Google Jump System beziehungsweise die Yi Halo haben damit kein Problem, da sich dort, wie in Unterabschnitt 5.1 gezeigt, die Kameras nicht gegenseitig aufnehmen.
Ein weiteres Problem auf Seiten der Somikon wurde in Abbildung 20 gezeigt. Da ein Video sowohl den linken als auch den rechten Teil eines Stereopaars enthält, muss entweder der Renderer die Texturen tauschen, wenn in die andere Richtung geschaut wird oder die Videos müssen vorher bearbeitet werden. Durch das Tauschen der Texturen während der Laufzeit kann dies als störend empfunden werden, wenn man sich an einem Grenzbereich befindet und die Texturen öfters hintereinander getauscht werden. Dadurch entsteht ein Flackern der Texturen.
Wenn man die Videos vorher trennt und richtig zusammenfügt, können an dem Bereich, an dem die Videos zusammengefügt wurden, Fehler entstehen, da die zwei Teilbereiche des Videos von unterschiedlichen Standpunkten aufgenommen wurden.
Dies passiert bei den Google Jump Systemen nicht, da die Panoramen aus den Aufnahmen der Kameras direkt richtig erzeugt werden, wie in Abbildung 43 zu sehen ist.

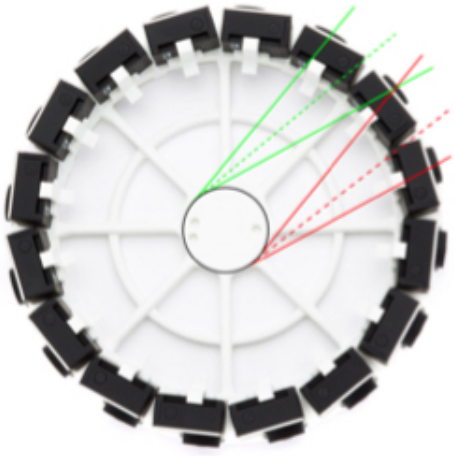

Abbildung 43: Erzeugung der Stereobilder bei Google Jump Systemen [ASG+]

5.3 Nutzung

Ein Punkt der bei der Bewertung ebenso berücksichtigt werden muss, ist die Komfortabilität der Nutzung der Systeme. Dazu wird als Erstes Tabelle 1 herangezogen. Beide Kameras verfügen über WiFi, wodurch die Aufnahmen über ein Smartphone gesteuert werden können, indem man die zugehörige App verwendet. Allerdings benötigt die App der Yi Halo ein Android System, während die App der Somikon Kamera sowohl mit Android als auch mit IOS benutzt werden kann. Ebenso ist die Somikon zirka 30x leichter als die Yi Halo und zirka 5x kleiner. Dadurch kann die Somikon wesentlich besser transportiert werden und sogar immer mitgeführt werden, da sie mit ihrer Größe in fast jede Tasche passt.
Um mit der Yi Halo eine Aufnahme zu bewerkstelligen, muss im Vorfeld geplant werden, welche Szene aufgenommen werden soll, damit die Yi Halo an diesen Ort gebracht werden kann. Generell ist sie weniger geeignet für spontane Aufnahmen.
Im Gegenzug treten bei Aufnahmen der Yi Halo keine Fehler auf, wie zum Beispiel das Verschieben einer Kamera während der Aufnahme, da alle Kameras zusammen fest auf einem Gehäuse sitzen. Die Somikon Kameras können zwar auch zusammen befestigt werden, allerdings verringert sich dabei ein Teil ihrer Transportierbarkeit, da die Kameras nichtmehr seperat transportiert werden können.
Nachdem die Aufnahmen erfolgreich gelungen sind, müssen die Videos bearbeitet werden. Die Google Jump Systeme verfügen hierfür über einen eigenen eingebauten Prozessor, mit dem das Panorama aus den aufgenommenen Videos direkt richtig berechnet und ausgegeben wird. Diese Videos

können direkt zur Vorschau an einen Renderer übergeben werden.
Die Videos der Somikon hingegen müssen über ein externes Programm bearbeitet werden, bevor sie verwendet werden können. Dies ist mit Arbeit und Zeit verbunden und es können Fehler bei der Verarbeitung gemacht werden. Dazu muss der Benutzer mit externen Programmen vertraut sein, um dieses System nutzen zu können.

6 Fazit und Ausblick

Mit dem in dieser Arbeit vorgestelltem prototypischen System können stereoskopische Video-Panoramen erstellt und mithilfe des entwickelten Renderers in Virtual-Reality Brillen dargestellt werden. Dafür werden 2 Somikon Kameras verwendet, die jeweils mit 2 Fischaugenobjektiven ausgestattet sind, um die gesamte Umgebung der Szene wahrzunehmen. Durch die Verwendung dieser preisgünstigen Kameras müssen Abstriche in Bildqualität und Immersion gemacht und die Panoramen müssen über ein externes Programm synchronisiert werden. Im Gegenzug dazu sind die Kameras klein und leicht und somit flexibel zu benutzen, weshalb dieses System für spontane Momentaufnahmen verwendet werden kann.
Für professionelle Panoramaaufnahmen gibt es alternative Systeme wie zum Beispiel Google Jump, welche einen hohen Wert auf eine gute Bildqualität legen und die Panoramen automatisch bearbeiten und synchron aufnehmen, sodass sie direkt verwendet werden können. Solche sind für Privatpersonen mit einem Preis von zirka 20.000 Euro aber nahezu unerschwinglich.
Der entwickelte Renderer hingegen kann jegliche Video-Panoramen darstellen, die in einem horizontalem Format vorliegen, auch wenn sie von einem anderen System erzeugt wurden. Außerdem ist der Renderer kompatibel mit allen derzeit für Windows verfügbaren Virtual-Reality Brillen.

Im Zuge zukünftiger Verbesserungen kann die Verbindung zwischen der Aufnahme der Kameras und dem Renderer verbessert werden. Dazu kann zum Beispiel ein Programm erstellt werden, welches die Aufnahmen der beiden Kameras automatisch synchronisiert, bearbeitet und an den Renderer weiterleitet. Dadurch würde nicht jedes aufgenommene Video mit einem externen Programm manuell bearbeitet werden müssen. Alternativ könnte zur Aufnahme eine Konstruktion entwickelt werden, die ermöglicht, dass beide Kameras die Aufnahme gleichzeitig starten zum Beispiel über einen Schalter der beide Auslöser der Kameras gleichzeitig betätigt.
Ein spannender Aspekt für die Zukunft stereoskopischer Video-Panoramen wäre es Kamerafahrten möglich zu machen, das heißt man hätte während des Betrachtens eines Panoramas die Möglichkeit sich in der Szene zu bewegen, anstatt sich nur mit einer Kopfrotation umschauen zu können. Im

Zuge dessen wäre es interessant ein Kamerasystem zu erstellen, welches die Panorama Videos über das Internet live an die Virtual-Reality Brille überträgt und gleichzeitig in seiner Position verändert werden kann.

Abbildungsverzeichnis

Literatur

[ASG+] Robert Anderson, Noah Snavely, David Gallup, Jonathan T. Barron, Carlos Hernandez, Sameer Agarwal, Janne Kontkanen, Steven M. Seitz, and Google Inc. Jump: Virtual reality video. 31, 33, 36

[AVN16] R. Aggarwal, A. Vohra, and A. M. Namboodiri. Panoramic stereo videos with a single camera. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3755–3763, 2016. 5, 6, 7, 35

[Goo] Google. Google jump. `https://vr.google.com/jump/`. [Online; accessed 14-11-2018]. 28, 29, 30, 36

[ocu] `https://www.oculus.com/rift/#oui-csl-rift-games=star-trek`. [Online; accessed 19-11-2018]. 19, 35

[ope] `https://opencv.org`. [Online; accessed 13-11-2018]. 25

[PBE99] S. Peleg and M. Ben-Ezra. Stereo panorama with a single camera. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 1, pages 395–401 Vol. 1, 1999. 3, 35

[PPBE00] S. Peleg, Y. Pritch, and M. Ben-Ezra. Cameras for stereo panoramic imaging. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, volume 1, pages 208–214 vol.1, 2000. 3, 4, 35

[RAPLP07] A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg. Dynamosaicing: Mosaicing of dynamic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10):1789–1801, 2007. 4, 5, 35

[SDK] `https://developer.oculus.com/documentation/pcsdk/latest/concepts/pcsdk-intro/`. [Online; accessed 11-11-2018]. 20

[som] `http://www.somikon.de/2Lens-360Grad-Cam-NX-14273-919.shtml`. [Online; accessed 7-11-2018]. 11, 12, 29, 35

[syn] `https://support.apple.com/kb/PH12585?locale=en_US&viewlocale=de_DE`. [Online; accessed 9-11-2018]. 15, 16, 35

[TK02] Stavros Tzavidas and Aggelos K. Katsaggelos. A multicamera setup for generating stereo panoramic video. *IEEE Transactions on Multimedia*, 7:880–890, 2002. 7, 8, 9, 35

[uni] https://unity3d.com/de. [Online; accessed 11-11-2018]. 20

[unr] https://docs.unrealengine.com/en-us/. [Online; accessed 11-11-2018]. 20

[Vala] Valve. Openvr. https://github.com/ValveSoftware/openvr. [Online; accessed 11-11-2018]. 20

[Valb] Valve. Openvr bsp. https://github.com/ValveSoftware/openvr/tree/master/samples/hellovr_opengl. [Online; accessed 11-11-2018]. 21, 25

[Wim04] Peter Wimmer. Aufnahme und wiedergabe stereoskopischer videos im anwendungsbereich der telekooperation. 05 2004. 2, 10, 17, 35

[WLO03] C. Weerasinghe, Wanqing Li, and P. Ogunbona. Stereoscopic panoramic video generation using centro-circular projection technique. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, volume 3, pages III–473, 2003. 8

[wob] https://www.adobe.com/de/products/premiere.html?gclid=Cj0KCQiA2o_fBRC8ARIsAIOyQ-kXhndJIViRWZl3WNStVQiD8S7LadU-vcMoRxNPmLzm596wcB&sdid=88X75SKP&mv=search&ef_id=W@TgGQAAAF0u1fP@:20181109011713:s&s_kwcid=AL!3085!3!272835361837!e!!g!!adobe%20premiere%20pro. [Online; accessed 9-11-2018]. 17

[YiR] https://www.yitechnology.com/yi-halo-vr-camera. [Online; accessed 14-11-2018]. 29, 30

[You] https://www.youtube.com/playlist?list=PLjhgRr1fomj17Xm7gG028Bcsk5zJ33PJm. [Online; accessed 14-11-2018]. 29, 32