Technische Universität Berlin

Fakultät I
Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Magisterarbeit

Implementierung einer netzwerkfähigen und interaktiven stereoskopischen Visualisierungsumgebung

vorgelegt von:
Ralf Baumbach
[redacted]

Erstgutachter:	Prof. Dr. Stefan Weinzierl
Zweitgutachter:	Dr. Hans-Joachim Mempel
Abgabedatum:	13. März 2009

Eidesstattliche Erklärung

Durch meine Unterschrift versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe.

Berlin, 13. März 2009 .....

Ralf Baumbach

Inhaltsverzeichnis

1 Vorwort

Besonderer Dank gebührt Alexander Lindau für die notwendigen Anpassungen der FABIAN-Steuerungssoftware, und für seine generelle Unterstützung bei Vorüberlegungen, Materialbeschaffung und Umgang mit Institutsressourcen. Außerdem danke ich Michael Horn für die Konstruktion des Kameraauslösekabels und diverse Hilfestellungen bei der Arbeit mit den institutseigenen Computern.

2 Einleitung

2.1 Motivation

Moderne Untersuchungen zu Wahrnehmung und Interaktion basieren häufig auf Experimenten, die es Versuchsteilnehmern ermöglichen, sich in virtuellen Umgebungen relativ frei zu bewegen. Die dafür verwendeten Systeme werden meist unter dem Begriff 'Virtuelle Realität' (VR) zusammengefasst. Das Ziel solcher Simulationsumgebungen ist die Erzeugung plausibler Stimuli. So lassen sich Untersuchungsbedingungen schaffen, die es erlauben unter kontrollierten Bedingungen Erkenntnisse über Wahrnehmungsphänomene zu gewinnen. Der Erkenntnisgewinn ist umso größer, je stärker sich die Versuchspersonen auf die künstliche Umgebung einlassen, in sie eintauchen können.

Der Idealfall einer vollständigen Immersion in eine künstliche Umgebung, liesse sich nur durch eine exakte Reproduktion der in einer vergleichbaren realen Situation anzutreffenden Sinneseindrücke und Interaktionsmöglichkeiten herbeiführen[1]. Ein System, welches dies ermöglicht würde zu Recht die Bezeichnung VR-System tragen. Da das visuelle Sinnessystem als wichtige Quelle menschlicher Wahrnehmung betrachtet wird, versuchen viele VR-Systeme vor allem visuelle Reize plausibel zu generieren. Dazu werden meist computergenerierte Ansichten für jedes Auge erzeugt und so eine stereoskopische Wahrnehmung ermöglicht[2].

Um die Plausibilität virtueller Umgebungen zu erhöhen, werden zunehmend andere Sinneskategorien mit visuellen Simulationen kombiniert. Bei solchen Simulationsumgebungen muss jeder Bestandteil der Simulation für sich genommen plausibel und interaktiv sein. Außerdem muss das Zusammenwirken der verschiedenen Reizkategorien ebenfalls plausibel sein.

Durch fortschreitende Entwicklungen auf dem Gebiet von 3D-Audio lassen sich bereits heute interaktive, auf Bewegungen des Hörers reagierende, akustische Umgebungen simulieren, die kaum noch von der Realität unterscheidbar sind[3]. Am Fachbereich Audiokommunikation der TU Berlin werden für solche Simulationen Daten eingesetzt, die mit dem von Lindau (2006) entwickelten Messsystem FABIAN erhoben werden. Dieses System erfasst Binaural-Daten, mit denen plausible akustische Umgebungen virtualisiert werden können. Für die

[1] Vgl. Grau 2003.
[2] *AVIE - Advanced Visualisation and Interaction Environment*; Cruz-Neira, Sandin und DeFanti 1993.
[3] Lindau, Hohn und Weinzierl 2007.

Erhöhung der Plausibilität wird eine interaktive Visualisierung der realen Aufnahmeräume benötigt. Dadurch, so die Annahme, würden die perzeptiven Unterschiede zwischen reellen und synthetischen Audio-Stimuli, aufgrund der höheren Plausibilität des Gesamteindrucks, vollends verschwinden.

2.2 Stand der Forschung

Visuelle VR-Systeme basieren bisher meist auf computergenerierten stereoskopischen Szenen[4]. Cruz-Neira, Sandin und DeFanti (1993) stellten mit CAVE eine immersive visuelle Umgebung vor, die heute in vielen Simulationen verwendet wird, in denen Blickwinkel von bis zu 360° horizontal und 180° vertikal notwendig sind. Eine Erweiterung um binaurale Reize stellte beispielsweise Assenmacher, Kuhlen und Lentz (2005) vor. CAVE-Umgebungen zeichnen sich vor allem durch den Vorteil der Mehrbenutzer-Fähigkeit aus. Einzelbenutzersysteme, die z.B. auf der Verwendung immersiver Darstellungstechnologien wie Head Mounted Displays (HMD) basieren, lassen sich theoretisch leicht in bestehende Audio-Frameworks integrieren. Allen Verfahren gemein ist die Beschränkung auf computergenerierte Darstellungen.

Echte periphere 360° x 180° -Wahrnehmung in allen Raumrichtungen wird bisher vor allem mit monoskopischen Panoramen ermöglicht[5]. Systeme wie AVIE (*AVIE - Advanced Visualisation and Interaction Environment*) können stereoskopische Panoramen in hoher Qualität visualisieren und ermöglichen auch die Kombination mit 3D-Audio. Der vertikale Sichtbereich ist jedoch aufgrund der verwendeten Zylinderprojektion begrenzt. Bourke (2006) und Peleg und Ben-Ezra (1999) beschreiben mögliche Verfahren zur Erzeugung stereoskopischer Panoramabilder mit begrenztem vertikalen Bildwinkel. Diese Verfahren sollten sich erweitern lassen und so die Erstellung von 360° x 180° -Vollpanoramen ermöglichen.

2.3 Zielsetzung

Die vorliegende Arbeit verfolgt zwei grundlegende Ziele: Zum einen soll eine Möglichkeit vorgestellt werden, stereoskopische Vollpanoramen mit einem überschaubaren Aufwand an Material und Zeit zu erzeugen. Besonderer Wert wird hierbei auf die Automatisierbarkeit der Panorama-Erzeugung gelegt. Dies ist unabdingbar, da für die Erstellung von Stereo-Panoramen ein deutlich höherer Aufwand zu betreiben ist als für monoskopische Panoramen[6]. Die Umsetzung dieses Teils der Arbeit wird in Kapitel 4 behandelt.

[4] Vgl. Assenmacher u. a. 2004; Assenmacher, Kuhlen und Lentz 2005; Cruz-Neira, Sandin und DeFanti 1993.
[5] Vgl. Chen 1995.
[6] Siehe Abschnitt 3.6.

Das zweite Ziel dieser Arbeit besteht darin die Entwicklung einer Visualisierungssoftware zu dokumentieren, die es ermöglicht, stereoskopische Vollpanoramen und manuell erstellte 3D-Szenen auf verschiedenen Geräten und in verschiedenen Stereo-Formaten darzustellen. Zu den Formaten gehören *anaglyph*, *interlaced*, *DLP3D*, *frame-sequentiell* und Dual-Ausgabe[7]. Damit ließen sich, neben vielen anderen, die bereits im Fachgebiet Audiokommunikation vorhandenen Geräte eMagin Z800 (HMD) und ein 62"-Samsung DLP-Fernseher (inkl. Shutterbrillen) verwenden. Bei der Entwicklung wurde, wie bereits im Titel der Arbeit ersichtlich, großer Wert auf Netzwerkfähigkeit und Interaktionsmöglichkeit gelegt. Unter Netzwerkfähigkeit wird hierbei die Möglichkeit verstanden, die Software weitestgehend über ein Netzwerk steuern zu können. Ein weiterer wichtiger Aspekt bei der Konzeption und Entwicklung der Software war Modularität. Damit liesse sich eine gute Erweiterbarkeit gewährleisten und die Software leicht neuen Bedürfnissen anpassen. Desweiteren soll die Software auf einer gängigen modernen PC-Architektur mit Grafikkarten aus dem Consumer-Bereich funktionieren und dabei möglichst plattformunabhängig sein. Die Umsetzung dieses Teils der Arbeit wird in Kapitel 5 behandelt.

Um die beschriebenen Zielstellungen umsetzen zu können, müssen eine Reihe von theoretischen Vorüberlegungen angestellt werden. In Kapitel 3 werden die menschliche Fähigkeit zur Wahrnehmung räumlicher Tiefe, sowie relevante Aspekte der Stereoskopie und der Panoramafotografie erörtert.

2.4 Geplante Anwendungsbereiche

Das in dieser Arbeit entwickelte Modellverfahren zur Erstellung stereoskopischer Vollpanoramen soll im Rahmen der Forschungsarbeiten am Fachbereich Audiokommunikation eingesetzt werden. Zum Zwecke der späteren Visualisierung mit der ebenfalls in dieser Arbeit dokumentierten Betrachtungssoftware, sollen Räume stereoskopisch abbildbar werden, an denen mit dem FABIAN-Messystem binaurale Audiodaten erhoben werden.

Die Visualiserungssoftware soll in das am Fachbereich Audiokommunikation entwickelte wonder-Projekt[8] integriert werden. Daher mussten bestimmte Schnittstellen und Verfahrensweisen übernommen werden, wie z.B. die Möglichkeit, die Anwendung über OpenSoundControl (OSC)[9] zu steuern.

Unabhängig von der Integration in das wonder-Projekt, soll es problemlos möglich sein, die entwickelte Software eigenständig zu verwenden. Als stand-alone-Programm zum Betrachten

[7] Siehe Abschnitt 3.2.5.

[8] Wave-Field-Synthesis Of New Dimensions of Electronic Music in Realtime - hardwareunabhängiges Wellenfeldsyntheseprogramm `http://sourceforge.net/projects/swonder`

[9] Vgl. Wright 2003.

von stereoskopischen Vollbild-Panoramen soll die Betrachtungssoftware auch in Verbindung mit anderen Versuchsframeworks verwendet werden können. Die einfache Erweiterbarkeit über Plugins, sowie die Verwendung von plattformunabhängigen Standard-Pythonmodulen für die Programmlogik und der ebenfalls plattformunabhängigen Grafik-Bibliothek OpenGL für das 3D-Rendering, soll eine leichte Integration in andere Softwarepakte gewährleisten. Insbesondere für Anwendungen im akademischen Umfeld wäre dies von Nutzen.

3 Theoretische Grundlagen

Um ein System zu entwickeln, mit dem ein realitätsnaher Raumeindruck ähnlich dem natürlichen Sehen erzeugt werden kann, müssen zunächst die Eigenschaften geklärt werden, die ein solches System erfüllen muss. Dieses Kapitel beschäftigt sich mit den dafür relevanten Aspekten der menschlichen visuellen Wahrnehmung sowie den Grundlagen und technischen Aspekten der Stereoskopie und der Stereofotografie. Außerdem werden die mathematischen und technischen Grundlagen für die Erstellung und Darstellung des benötigten Bildmaterials eingeführt.

3.1 Wahrnehmung räumlicher Tiefe

Die visuelle Wahrnehmung beim Menschen basiert auf der Verarbeitung von Lichtreizen aus der Umwelt. Über die Hornhaut, die Pupille und die Linse treten Lichtstrahlen in das Auge ein und werden von Rezeptoren der Netzhaut in bioelektrische Aktivität des Nervensystems umgesetzt[1]. Diese Aktivität wird vom visuellen System in hochspezialisierten Arealen des menschlichen Gehirns weiterverarbeitet und führt letztlich zu einer mentalen Repräsentation einer visuellen Szene. Dies stellt, grob verkürzt, den Sehvorgang dar[2].

Für die räumliche Analyse visueller Szenen stehen dem Menschen eine Reihe von Informationen zur Verfügung. Goldstein (2002)[3] unterscheidet okulomotorische[4], monokulare[5], bewegungsinduzierte und stereoskopische Informationen für die Wahrnehmung räumlicher Tiefe.

Okulomotrische Informationen Durch Auswertung der für Konvergenz und Akkommodation verantworlichen Muskelspannung kann das Gehirn auf die Entfernung des betrachteten Objektes schließen.

Monokulare Informationen Hierbei handelt es sich um strukturelle Eigenheiten der betrachteten Szene, die mit der Entfernung kovariieren. In der Literatur werden u.a. die Ver-

[1] Vgl. Goldstein 2002, S 44.
[2] Vgl. Goldstein 2002, Kap. 1 - 8.
[3] Vgl. Goldstein 2002, S. 226 - 236.
[4] Die Motorik des Auges betreffend.
[5] Einäugig, ein Auge betreffend.

deckung von Objekten, die relative Höhe und Größe von Objekten im Gesichtsfeld, atmosphärische Perspektive und Texturgradienten genannt.

Bewegungsinduzierte Informationen Dies sind Informationen, die durch einen bewegten Betrachter oder eine um diesen herum bewegte räumliche Szene entstehen, z.B. das Verdecken von Objekten durch andere Objekte, abhängig von Bewegungen des Betrachters. Ein anderes Beispiel sind Bewegungsparallaxen[6].

Stereoskopische Informationen Diese Art von Information stellt die Hauptquelle der menschlichen Tiefenwahrnehmung dar. Sie basiert auf dem Vergleich der leicht unterschiedlichen Perspektiven, die jedes der beiden Augen auf die gerade betrachtete Szene hat. Diese Differenzen in der Abbildung der Umwelt auf die Netzhäute beider Augen werden Querdisparation oder auch binokulare[7] Disparation genannt.

Die meisten dieser Informationen stehen auch bei monokularem Sehen zur Verfügung, indem das Gehirn die betreffenden szeneninhärenten Informationen auswertet. Ein besonderer Aspekt der visuellen Wahrnehmung, der auf der Auswertung von Querdisparation und Konvergenz basiert, ist die Stereopsis[8].
Die menschlichen Augen befinden sich im Gesicht in gleicher Höhe und liegen in etwa symmetrisch zur Nasenwurzel. Der horizontale Augenabstand[9] beträgt im Durchschnitt etwa 63 mm[10], so dass linkes und rechtes Auge eine lateral leicht unterschiedliche Sicht auf die betrachtete Szene haben. Dieser parallaktische Versatz ist umso größer, je näher das betrachtete Objekt liegt. Das Gehirn ist in der Lage die beiden unterschiedlichen Bilder zu fusionieren. Das Ergebnis ist eine Wahrnehmung räumlicher Tiefe, das sogenannte Raumsehen[11].

Kemner (1989, S. 10) fasst es folgendermaßen zusammen: "Voraussetzung für die Entstehung eines Raumeindrucks im Gehirn sind also zwei scharf eingestellte, parallaktisch unterschiedliche Einzelbilder (Halbbilder)".

3.1.1 Konvergenz und Akkommodation

Konvergenz[12] bezeichnet die Ausrichtung beider Augen auf einen Raumpunkt (Fixationspunkt). Dies geschieht durch eine gegensinnge Augenbewegung um die Hochachse der Augen herum. Dadurch wird der Fixationspunkt auf den hochauflösendsten Teil der Netzhaut abge-

[6] Ein bewegter Betrachter sieht nahe Objekte schneller an sich vorüberziehen, als weiter entfernte Objekte. Dies lässt sich besonders gut in schnellen Fortbewegungsmitteln wie Autos oder Zügen beobachten.
[7] Beide Augen betreffend, mit beiden Augen.
[8] Aus dem Griechischen: *stereo* = räumlich, ausgedehnt – *opsis* = Sicht.
[9] In der Stereoskopie auch Stereobasis genannt.
[10] Vgl. Dalzell und Linssen 1953, S. 8; Kemner 1989, S. 10.
[11] Vgl. Poggio und Poggio 1984, S. 2; Bruce, Georgeson und Green 2006, Kap. 7.
[12] Aus dem Lateinischen *convergere* = sich hinneigen, zusammenneigen.

bildet. Die Augen werden stärker zueinander gedreht, je näher der Betrachter dem Fixationspunkt ist (siehe Abbildung 3.1a). Bei Betrachtung eines sehr weit entfernten Fixationspunktes sind die Augen parallel ausgerichtet. Der Winkel zwischen den konvergierenden Sichtlinien wird als *Parallaxenwinkel* bezeichnet. Er ist umgekehrt proportional zur Entfernung des Fixationspunktes und lässt sich mit folgender Formel bestimmen:

$$\phi = 2 * arctan(r/f) \tag{3.1}$$

Dabei entspricht r dem halben Augenabstand und f der Länge zwischen dem Mittelpunkt beider Augen und dem Fixationspunkt.

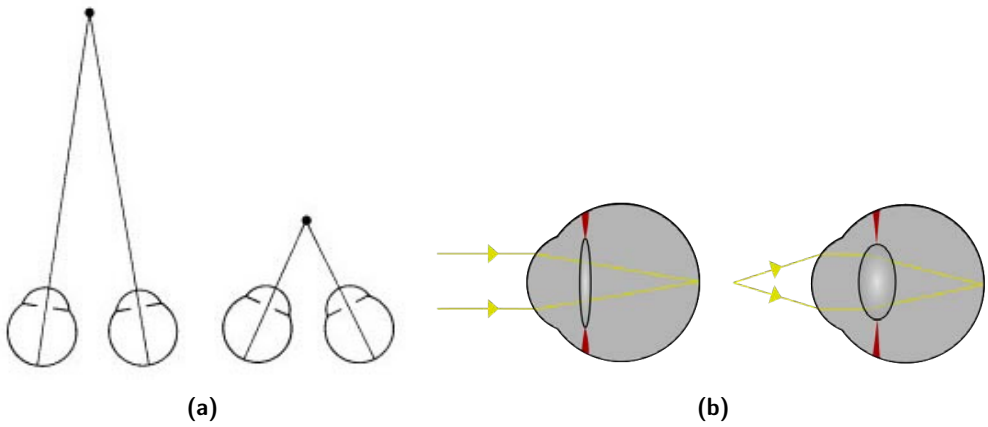


Abbildung 3.1: (a) konvergente Augenstellung bei Fern- und Nahfokussierung, (b) Linsenverformung um bei unterschiedlichem Objektabstand ein scharfes Netzhautbild zu erhalten

Akkommodation[13] bezeichnet das Verformen der Linse, wodurch die Gesamtbrechkraft des optischen Systems und damit die Schärfentiefe verändert wird. Bei der Akkommodation auf einen Fixationspunkt, wird die Linse so eingestellt, dass auf der Netzhaut ein scharfes Abbild des fokussierten Objektes abgebildet wird (siehe Abbildung 3.1b). Laut Bruce, Georgeson und Green (2006, S. 16) umfasst der natürliche, d.h. keine Akkommodation erfordernde, Schärfetiefenbereich des menschlichen Auges den gesamten Sichtbereich ab einer Entfernung von 6 m. Nähere Bereiche müssen durch Akkommodation scharf gestellt werden. Die Grenzpunkte, zwischen denen das Scharfstellen gelingt, werden als Nah- und Fernpunkt bezeichnet. Ihre Lage ist altersabängig[14].

Beim natürlichen Sehen sind Akkommodation und Konvergenz immer miteinander gekoppelt[15]. Beide Anpassungsmechanismen laufen normalerweise unbewusst ab, können aber bewußt kontrolliert werden, z.B. beim Schielen oder beim entspannt ins Unendliche schauen".

[13] Aus dem Lateinischen *accomodare* = anpassen, adaptieren.
[14] Vgl. Goldstein 2002.
[15] Vgl. Goldstein 2002; Kemner 1989.

Laut Goldstein (2002, S. 228) sind Konvergenz und Akkommodation "vor allem im unmittelbaren Greifbereich und bis zu Distanzen wirksam, die kleiner als eineinhalb bis drei Meter sind".

3.1.2 Querdisparation

Die Netzhautabbildungen aus rechtem und linkem Auge sind lateral verschoben. Dieser Unterschied wird als Querdisparation bezeichnet. Abbildung 3.2 zeigt ein Beispiel für unterschiedliche Netzhautabbildungen in beiden Augen.

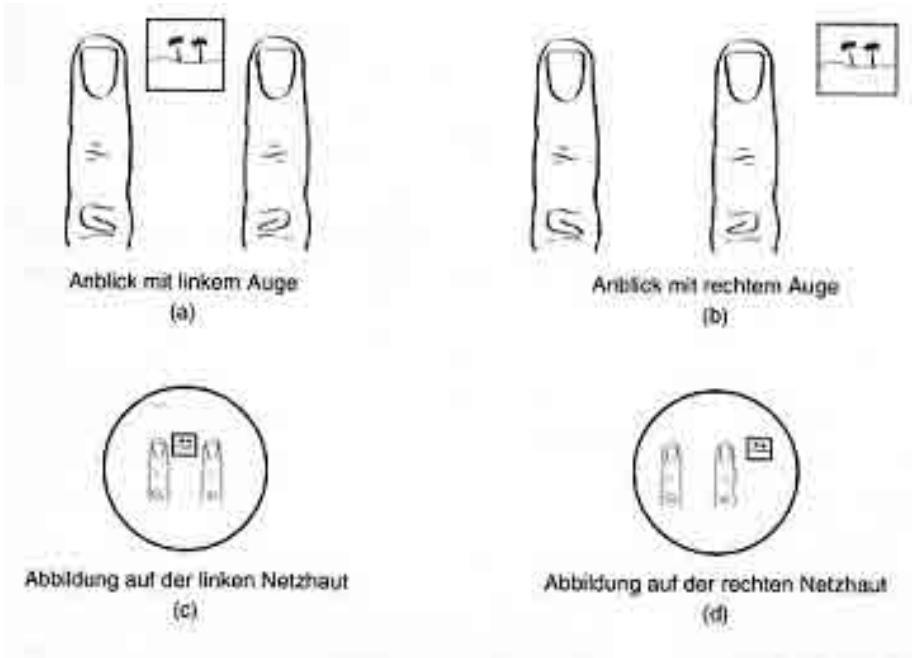


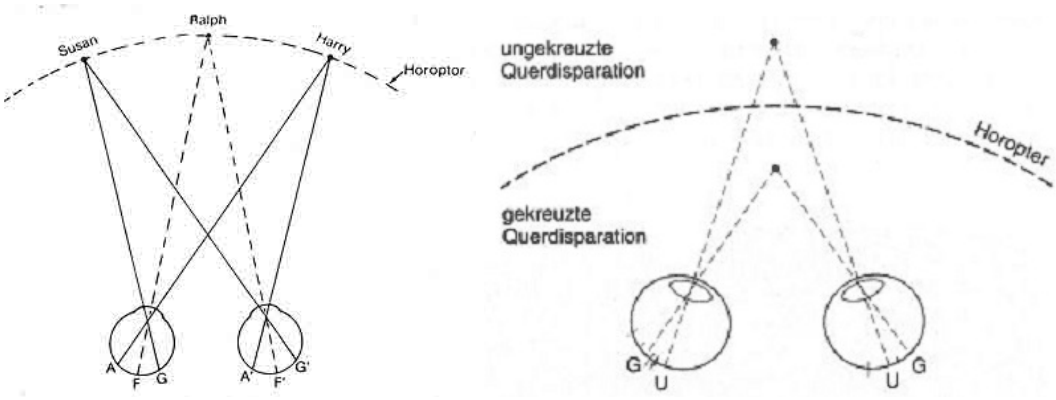
Abbildung 3.2: Beispiel für Querdisparation in rechtem und linkem Auge (Goldstein 2002, S.237)

Aufgrund von Querdisparation kommt es auch zur Wahrnehmung von Doppelbildern: Wenn ein Objekt, z.B. ein Finger, vor das Gesicht gehalten wird, ein Punkt im Hintergrund des Objektes fokussiert wird und dann abwechselnd das rechte und das linke Auge geöffnet und geschlossen werden, so scheint das Objekt im Vordergrund hin- und herzuspringen. Werden beide Augen gleichzeitig geöffnet während weiterhin der Punkt im Hintergrund fokussiert bleibt, so erzeugt das Objekt im Vordergrund ein Doppelbild. Beide Phänomene lassen sich damit erklären, dass die Netzhautabbildungen des Objektes im Vordergrund in beiden Augen auf unterschiedliche Punkte fallen.
Im Gegensatz dazu wird der fokussierte Punkt im Hintergrund in beiden Augen im Zentrum der Netzhaut abgebildet. Relativ zur Geometrie der Netzhaut liegt die Abbildung also in beiden Augen an der gleichen Stelle. Diese Punkte werden als korrespondierende Netzhaut-

punkte bezeichnet[16].

Es gibt eine gedachte Linie, den sogenannten Horopter, auf dem Objekte liegen, die auf korrespondierenden Netzhautpunkten abgebildet werden (siehe Abbildung 3.3a). Befinden sich Objekte vor oder hinter dem Horopter, werden diese nicht auf korrespondierende Netzhautpunkte abgebildet und erzeugen so Doppelbilder bei der Fusion beider Bilder. Gekreuzte Querdisparation entsteht bei Objekten, die vor dem Horopter liegen. Ungekreuzte Querdisparation entsteht bei Objekten die hinter dem Horopter liegen (siehe Abbildung 3.3b)[17].

Im Kontext der Stereoskopie wird die Querdisparation auch als parallaktische Verschiebung oder einfach Parallaxe bezeichnet.



(a) (b)

Abbildung 3.3: Querdisparation (a) und Horopter mit korrespondierenden Netzhautpunkten (b) (Goldstein 2002, S.240 - 241)

3.1.3 Schlussfolgerungen

Für die vorliegende Arbeit haben von den genannten Tiefenkriterien lediglich die Querdisparation und die okulomotorischen Informationen Konvergenz und Akkommodation eine Bedeutung.
Konvergenz und Akkommodation sind insofern von Bedeutung, als sie beim natürlichen Sehen gekoppelt sind, beim künstlichen stereoskopischen Sehen aber entkoppelt werden müssen[18]. Dies führt zu Verzögerungen beim Scharfstellen der visuellen Szene und kann von einigen Betrachtern als unangenehm empfunden werden.
Querdisparation ist für die Qualität und die Akzeptanz des Stereo-Eindrucks wichtig und lässt sich über die Stereobasis beeinflussen.

[16] Vgl. Bruce, Georgeson und Green 2006, S. 171; Goldstein 2002, S. 237 - 238.
[17] Goldstein 2002, S. 238-241.
[18] Siehe Abschnitt 3.2.2.

Die übrigen Tiefenkriterien können im Rahmen dieser Arbeit vernachlässigt werden, da sie von dem konkreten Vorgang des Binokularsehens unabhängige Bestandteile der Analyse visueller Szenen sind. An der Nachbildung bzw. Erzeugung eines realistischen Raumeindrucks sind sie nur indirekt beteiligt, z.B. in Hinblick auf die Komposition und Anordnung von Objekten in einer zu erfassenden stereoskopischen Szene.

3.2 Stereoskopie

Unter dem Begriff Stereoskopie werden alle Methoden und Verfahren zusammengefasst, welche die Erzeugung und Wiedergabe dreidimensionaler Bilder auf Basis der Prinzipien des natürlichen Binokularsehens ermöglichen.

3.2.1 Grundlagen der Stereoskopie

Seitdem Wheatstone im Jahre 1838 seine Idee von der Konstruktion eines Stereoskops vorgestelle[19], hat sich an den Prinzipien der stereoskopischen Darstellung nichts verändert: Zwei Bilder, sogenannte stereoskopische Halbbilder, die aus leicht unterschiedlichen Blickwinkeln die gleiche visuelle Szene darstellen, werden den Augen getrennt dargeboten und bieten somit fast das gleiche sensorische Material wie es auch bei einer direkten binokularen Betrachtung der Szene vorhanden wäre.

Abbildung 3.4: Beispiel für stereoskopische Halbbilder (aus Goldstein, 2002)

In Abbildung 3.4 sind zwei zusammengehörige stereoskopische Halbbilder dargestellt. Bei genauer Betrachtung wird deutlich, dass die auf den ersten Blick fast identisch wirkenden Bilder von einem unterschiedlichen Blickpunkt aus aufgenommen wurden. So ist z.B. die Person, die im Hintergrund des Bildes auf dem Spielfeld steht, dem Spieler mit der Nummer

[19] Vgl. Wheatstone 1838.

12 im linken Bild sichtbar näher als im rechten Bild. Es ist diese Parallaxenverschiebung, die dem Auge die Fusion beider Teilbilder zu einem dreidimensionalen Bild ermöglicht. Das menschliche Gehirn ist in der Lage, die aus den unterschiedlichen Netzhautabbildern resultierende Querdisparation zu bestimmen und damit ein Indiz für die räumliche Lage relativ zum Blickpunkt des Betrachters zu erhalten.

3.2.2 Grenzen der stereoskopischen Darstellung

Wie jede Simulationstechnik, ist auch die Stereoskopie mit Schwierigkeiten konfrontiert, die ein realistisches und angenehmes Sehereignis beeinträchtigen können. Die im Rahmen dieser Arbeit relevanten Punkte werden im folgenden erläutert:

Akkommodation: Ein grundlegender Unterschied zwischen realem Sehen und der Nachbildung des binokularen Sehens durch stereoskopische Techniken liegt in der Entkopplung von Konvergenz und Akkommodation. Der Fokuspunkt, auf den die Augen akkommodieren, stimmt nicht mit der wahrgenommenen Position des betrachteten Objektes überein. Stattdessen akkommodieren die Augen auf die Abbildungsebene, z.B. auf den Monitor. Die Konvergenz bleibt davon unberührt, so dass im Gegensatz zum natürlichen Sehvorgang beide Mechanismen unanbhängig voneinander gesteuert werden müssen. Dies kann von einigen Betrachtern als unangenehm empfunden werden. Vor allem aber führt es bei schnellen Szenenwechseln dazu, dass die Augen immer wieder den Akkommodationspunkt in der Nähe des Konvergenzpunktes suchen und sich dann neu anpassen müssen[20].

Dynamische Szenen: Während generell die Möglichkeit besteht, dynamische stereoskopische Szenen zu erstellen und zu betrachten, ergeben sich notwendige Einschränkungen aus der beschriebenen Unmöglichkeit der Augen, auf einen fixierten Punkt zu akkommodieren. Bei Bildwechseln die einen großen Tiefenbereich im Stereobild überspringen, können Betrachter dem Bildverlauf nur mit Anstrengung folgen. Bei der Erstellung stereoskopischen Materials mit dynamischen Szenen ist also darauf zu achten, dass Bildwechsel möglichst kleine Tiefenbereiche umfassen.

Schärfe: Stereoskopische Darstellungen müssen über eine gute Allgemeinschärfe und eine gute Schärfentiefe verfügen. Wenn es unscharfe Bildbereiche gibt, versuchen die Augen vergeblich auf den Bereich scharf zu stellen, können aber immer nur auf das pyhsische Darstellungsmedium akkommodieren. Daher sind für die Aufnahme stereoskopischer Bilder Objektive mit möglichst geringen Brennweiten zu bevorzugen.

[20] Vgl. Javidi und Okano 2002, S. 9 - 10.

3.2.3 Stereo-Fotografie

Stereoskopische Bilder können mit speziellen zwei-objektivigen Stereokameras (siehe Abbildung 3.5) aber auch mit handelsüblichen Spiegelreflex- oder Kompaktkameras aufgenommen werden. Erstere bieten den Vorteil, dass auch dynamische Szenen fotografiert[21], sowie Freihandaufnahmen gemacht werden können, weil die Kamera beide Perspektiven synchron aufnimmt. Die Aufnahme mit Ein-Objektiv-Kameras erfordert den Einsatz einer Stereoschiene und eines Stativs. Da zwei Bilder in Folge aus verschiedenen Blickwinkeln aufgenommen werden, muss sichergestellt sein, dass sich keine beweglichen Objekte im Aufnahmebereich befinden.

Abbildung 3.5: Stereokamera[22]

Außerdem gibt es die Möglichkeit, eine Ein-Objektiv-Kamera mit einem sogenannten Strahlenteiler zu versehen. Diese Vorsätze werden heute aber nicht mehr produziert und sind ebenso wie Stereokameras schwer erhältlich.

Der Abstand zwischen den beiden Objektiven wird als Stereobasis bezeichnet und beeinflusst direkt den erzeugten Stereoeffekt. Bei zu großer Stereobasis kann es zu sogenanntem Liliputismus kommen, d.h. "das Raumbild wirkt plastischer, erscheint aber verkleinert"(Wimmer 2004, S. 28). Das Gegenteil davon, der Gigantismus trifft bei zu kleiner Stereobasis auf und wird z.B. für Makroaufnahmen verwendet.

[21] Dies gilt aus nachvollziehbaren Gründen nicht für stereoskopische Panoramen, da diese die Aufnahme mehrerer Bilder hintereinander erfordern (Siehe Abschnitt 3.6.)

[22] Quelle: `http://commons.wikimedia.org/w/index.php?title=File:FED_Stereo_100_2706.jpg&oldid=10604446` (Abruf am 06.03.2009)

Als allgemeine Faustformel für die Wahl der Stereobasis b für ein Objekt in der Entfernung Y von der Kamera gibt Ray (1988, S. 476) an:

$$b/Y = 1/50 \tag{3.2}$$

Für eine umfassende Einführung in die stereoskopische Fotografie siehe Ray (1988); Dalzell und Linssen (1953).

3.2.4 Betrachtungsarten

Neben mechanischen Betrachtungsapparaten, wie den lange Zeit populären klassischen Stereoskopen nach Wheatstones Entwurf, gibt es weitere Betrachtungsarten für Stereobilder. Allen gemein ist das in Abschnitt 3.2.1 besprochene Prinzip der stereoskopischen Halbbilder. Je nach Art der gewünschten Betrachtung müssen die Bilder mehr oder weniger aufwendig vorbereitet werden. Abhängig von den verwendeten Hilfsmitteln lassen sich vier Betrachtungsarten unterscheiden[23]:

Hilfsmittelfrei: Ohne spezielle Zusatzgeräte können stereoskopische Bilder mittels Kreuzblick und Parallelblick betrachtet werden. Da die bewusste Trennung von Konvergenz und Akkommodation trainiert werden muss, eignen sie sich nur für geübte Betrachter.

Passive Systeme sind Betrachtungsarten, die keine elektronische Ansteuerung der Hilfsmittel erfodern und auf herkömmlichen Darstellungsmedien ausgegeben werden können. Beispiele sind Anaglyphen, Polarisation und KMQ. Sie stellen teils besondere Anforderungen an die dargestellten Bilder, können dafür aber auch von ungeübten Benutzern verwendet werden und eigenen sich für Bilder jeder Größe.

Aktive Systeme: Bei Betrachtung mit aktiven Hilfsmitteln, v.a. Shutterbrillen, wird eine elektronische Ansteuerung und eine Synchronisation zwischen Computer und verwendeter Peripherie benötigt.

3D-Displays: Außerdem gibt es noch speziell auf die Stereowiedergabe optimierte Systeme wie Head Mounted Displays (HMDs) oder autostereoskopische Displays.

Für eine ausführliche Vorstellung und Diskussion moderner Stereo-Technologien siehe Javidi und Okano (2002).

[23] Vgl. Wimmer 2004, Kap. 4.

3.2.5 Technische Verfahren zur Bildtrennung

Für die Darstellung stereoskopischer Bilder haben sich mehrere technische Verfahren etabliert. Welches davon eingesetzt wird, hängt vor allem vom verwendeten Ausgabemedium und -gerät ab. Während in der Anfangszeit noch mit Spiegelsystemen gearbeitet wurde, kamen bald Prismen und Linsen zum Einsatz. Seit der weiten Verbreitung von Computern stehen eine Vielzahl anderer Betrachtungsmöglichkeiten zur Verfügung.

Anaglyph: Bei Anaglyphbildern erfolgt die Trennung von linkem und rechtem Halbbild über die Trennung des Farbraumes. Dabei gibt es verschiedene Formate, von denen die Rot-Grün- bzw. Rot-Cyan-Aufteilung die häufigste ist. Für die Anaglyphbetrachtung werden die beiden, unterschiedlich eingefärbten Halbbilder überlagert. Die für die Betrachtung notwendige Trennung erfolgt über Brillen mit Farbfilter. Auf diese Weise sieht jedes Auge je ein Halbbild. Anaglyphbilder bieten im Vergleich zu den folgenden Verfahren den Vorteil, dass sie auch in Büchern und Zeitschriften abgebildet werden können[24].

Interlaced: Bei der Interlaced-Darstellung erfolgt die Trennung der stereoskopischen Halbbilder über die Bildzeilen (Zeilen-Interlaced) oder die Bildspalten (Spalten-Interlaced). Die beiden Teilbilder werden so überlagert, dass jeweils nur jede zweite Zeile bzw. Spalte übernommen wird. Für die Betrachtung bedarf es einer speziellen Shutter-Brille.

DLP3D: Ähnlich dem Interlaced-Format werden auch bei DLP3D zwei Bilder pro Frame kodiert. Die Verschachtelung der Bildinformationen erfolgt allerdings nicht zeilenweise, sondern schachbrettartig (siehe Abbildung 3.6). Auch bei diesem Format sind für die Betrachtung spezielle Shutterbrillen erforderlich.

Dual: Bei diesem Format werden die Teilbilder bei doppelter Gesamt-Bildschirmauflösung nebeneinander gerendert und in dieser Anordnung über die Grafikkarte ausgegeben. Dieses Format lässt sich u.a. in Verbindung mit Dual-VGA-Ausgängen verwenden um beispielsweise entsprechend ausgelegte HMDs anzusteuern.

Frame-Sequentiell: Bei diesem Format erfolgt die Bildausgabe an das Endgerät in der doppelten effektiven Bildwiederholrate. Dabei werden die Halbbilder in alternierender Reihenfolge erzeugt. Bei gerätespezifischen effektiven Bildwiederholraten von 60 Hz müssen also

[24] Vgl. auch Wimmer 2004, S. 37 - 40.

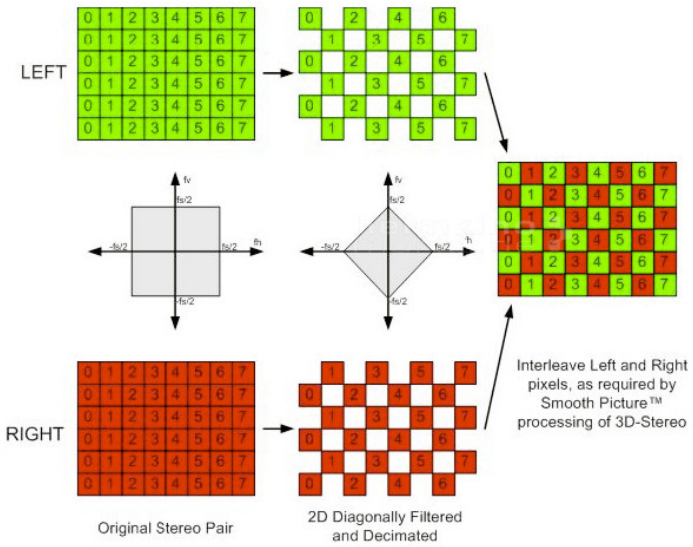


Abbildung 3.6: DLP3D HDTV Videoformat (aus einer Werbebroschüre von Texas Instruments)

120 Bilder pro Sekunde generiert werden. Der erzeugte Bilder-Strom besteht dabei aus abwechselnd je einem Bild für das linke und einem Bild für das rechte Auge. Das verwendete Ausgabegerät muss sicherstellen, das die Einzelbilder dem richtigen Auge präsentiert werden.

3.3 OpenGL

OpenGL[25] ist eine offene Grafik-Bibliothek zur Erstellung von plattformunabhängigen 2D- und 3D-Computeranwendungen. Die für die weiteren Ausführungen erforderlichen Grundlagen sollen hier verkürzt eingeführt werden. Für eine ausführlichere Einführung siehe Shreiner u. a. (2005).

3.3.1 Buffer

OpenGL verwendet für das Rendering einer 3D-Szene sogenannte Buffer. Das sind 2-dimensionale Datenstrukturen (Arrays), die pro Pixel bestimmte Daten speichern. Die Anzahl der Pixel richtet sich nach der Bildschirm-Auflösung, d.h. ein Bildschirm mit einer Auflösung von 800 x 600 Pixeln wird auch durch ein Array präsentiert, dass Platz für 800 x 600 Elemente hat. Durch Manipulationen der Buffer lässt sich das Ergebnis des Renderings direkt beeinflussen. Als Framebuffer wird die Zusammenfassung der vier im Folgenden beschriebenen Buffer bezeichnet.

[25] Open Graphics Library `http://www.opengl.org/` (Abruf am 12.03.2009)

Color Buffer: Der Color Buffer ist ein Bitmap, in dem jedem Pixel eine bestimmte Farbe zugeordnet ist. Der Inhalt des Color-Buffers entspricht dem resultierenden Bild auf dem Bildschirm. In den meisten Fällen wird ein sogenannter Double Buffer verwendet. Das sind zwei identische Color Buffer zwischen denen hin- und hergewechselt werden kann. Während der Front Buffer auf dem Bildschirm dargestellt wird, wird im Hintergrund das nächste Bild in den Back Buffer geschrieben. Sobald das Zeichnen in den Back Buffer abgeschlossen wurde, werden die beiden Buffer vertauscht und der Inhalt des Back Buffers wird am Bildschirm dargestellt. Dieses Umschalten ist wesentlich performanter als das direkte Zeichnen in den Front Buffer, v.a. wenn komplexere geometrische Berechnungen notwendig sind.

Depth Buffer: Der Depth Buffer (auch als Z-Buffer bekannt) speichert für jeden Pixel im Color Buffer, in welcher Entfernung vom Betrachter er sich befindet. Dadurch lassen sich Verdeckungsberechnungen durchführen, indem vor dem Überschreiben eines Pixels überprüft wird, ob der neue Pixel näher am Betrachter liegt.

Stencil Buffer: Der Stencil Buffer maskiert bestimmte Bereiche des Bildschirms, so dass dort keine Pixel geschrieben werden können. In dieser Arbeit wird der Stencil Buffer z.B. bei den Stereo-Modi *Interlaced* und *DLP3D* verwendet.

Accumulation Buffer: Der Accumulation Buffer kann ebenso wie der Color Buffer Farbinformationen aufnehmen. Allerdings verfügt er über eine höhere Farbtiefe von 32 Bit. Dadurch lassen sich Farb-Berechnung mit einer größeren Bittiefe durchführen, als das Grafikausgabe-Gerät unterstützt.

3.3.2 Matrizen

OpenGL verwendet drei 4x4-Matrizen zur Generierung, Manipulation und Darstellung von 3D-Szenen. Beim Arbeiten mit OpenGL wird fast immer eine dieser Matrizen verändert. Die Wahl der Matrix, die geändert werden soll, erfolgt in OpenGL mit dem Befehl `glMatrixMode()`.

Projektionsmatrix: Diese Matrix kontrolliert die Projektion einer 3D-Szene auf den Bildschirm. Sie fungiert also als Kamera, die man beliebig transformieren kann um die Bildschirmausgabe zu verändern.

Modelviewmatrix: Diese Matrix enthält das aktuelle Koordinatensystem. Sollen Objekte auf den Bildschirm gezeichnet werden, muss diese Matrix aktiviert sein. Sie kann durch die Befehle `glTranslate()`, `glRotate()` und `glScale()` verändert werden.

Texturmatrix: Die Texturmatrix wird zur Manipulation von Texturobjekten verwendet. Sie stellt ein Äquivalent zur Modelviewmatrix dar, bezieht sich aber auf einen anderen

Gültigkeitsbereich. Für die Manipulation können die gleichen OpenGL-Befehle verwendet werden.

3.3.3 Geometrie

OpenGL verwendet ein rechtshändiges kartesisches Koordinatensystem, wie in Abbildung 3.7b zu sehen. Die Kamera ist standardmässig im Koordinatenursprung positioniert und entlang der negativen Z-Achse ausgerichtet. Um die Position von Objekten im Koordinatensystems steuern zu können, gibt es drei grundlegende Funktionen:

`glTranslate(x, y, z)`: Translation um den mit x, y und z spezifizierten Vektor

`glRotate(`ϕ`, x, y, z)`: Rotation von ϕ Grad um die durch x, y und z spezifizierte Achse

`glScale(x, y, z)`: Skalierung der Achsen des Koordinatensystems um x, y und z

Lokale Koordinaten: Objekte können in ihrem eigenen lokalen Koordinatensystem definiert werden. Die Zeichenoperation erfolgt dann, in dem mit Hilfe der Transformationsbefehle in den Ursprung des lokalen Koordinatensystems transformiert wird. Dort kann das Objekt dann mit den lokalen Koordinaten konstruiert werden. Damit ist es möglich Objekte an beliebigen Positionen einer 3D-Szene einzufügen ohne die Koordinaten anzupassen.

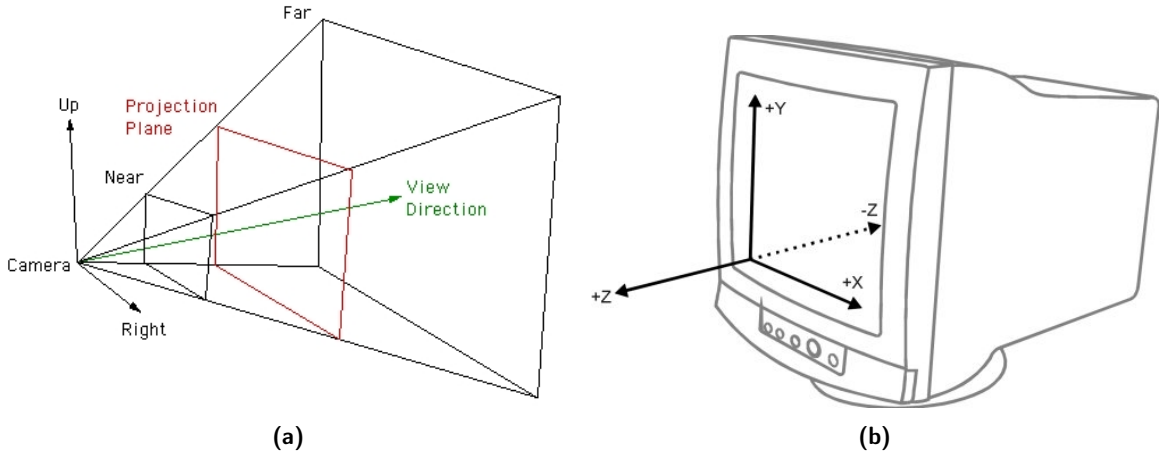


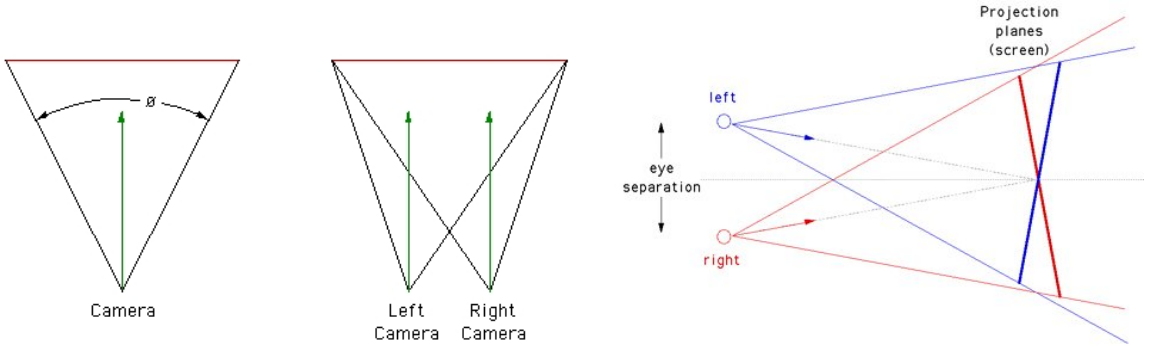
Abbildung 3.7: (a) Sichtvolumen und (b) OpenGL-Koordinatensystem[26]

[26] Quellen: (a) Bourke (1999), (b) `http://graphics.cs.uni-sb.de/Courses/ws9900/cg-seminar/Ausarbeitung/Philipp.Walter/vertexops.htm` (Abruf am 05.03.2009)

3.3.4 Sichtvolumen

Als Sichtvolumen oder *Frustum*[27] wird der darstellbare Bereich einer OpenGL-Szene bezeichnet. Objekte, die aus dem Sichtvolumen herausragen, werden abgeschnitten (engl. *Frustum Culling*). Ein Sichtvolumen ist durch sechs Parameter gekennzeichnet: Zwei Koordinaten für die linke und rechte Schnittfläche, je eine Koordinate für die untere und obere Schnittfläche und den Abstand zur Nah- und Weit-Schnittfläche. Um das Frustum zu definieren, wird in OpenGL der Befehl `glFrustum()` verwendet[28].

Bei der Darstellung einer stereoskopischen Szene sollen beide Kameras, von lateral verschobenen Standpunkten aus, den gleichen Ausschnit aus der 3D-Szene zeigen. Um das zu erreichen, gibt es prinzipiell folgende zwei Möglichkeiten.



(a) symmetrisches und asymmetrisches Sichtvolumen **(b)** Toe-In-Methode

Abbildung 3.8: (a) Sichtvolumen für monoskopisches und steresokopisches Rendering (b) Toe-in-Methode (aus Bourke (1999))

Symetrisches Sichtvolumen: Abbildung 3.8a zeigt in der linken Darstellung ein Beispiel für ein symmetrisches Sichtvolumen bei monoskopischem Rendering. Die rechten und linken Begrenzungen sind gleich weit von der Kameraachse entfernt. Um mit diesem Ansatz in beiden Kameras den gleichen Ausschnitt auf der Projektionsebene abzubilden, muss der in Abbildung 3.8b gezeigte Toe-in-Aufbau verwendet werden. Durch die zu den Rändern der Projektionsfläche hin zunehmende vertikale Parallaxe, wird die Betrachtung unangenehm. Dieser Effekt fällt bei größeren Blickwinkeln stärker aus als bei kleinen. Obwohl mit der Toe-in-Methode keine korrekten Stereobilder gerendert werden können, kommt sie aus Kostengründen oder aufgrund von Beschränkungen der Grafikhardware dennoch häufig zum Einsatz[29].

[27] Engl. für Kegelstumpf.
[28] Siehe Listing A.7.
[29] Vgl. Bourke 1999; Bourke 2001.

Asymmetrisches Sichtvolumen: Für das korrekte Rendern einer stereoskopischen 3D-Szene müssen asymmetrische Sichtvolumina mit parallelen Kameraachsen verwendet werden[30]. Die rechte Darstellung in Abbildung 3.8a zeigt den entsprechenden Aufbau. Bei diesem Ansatz treten keine Parallaxenfehler auf und die resultierenden Stereo-Bilder sind weniger anstrengend zu betrachten.

3.3.5 Kamera

Kameras werden in OpenGL in Form von Betrachtungsmatrizen realisiert. Mit diesen lässt sich die Position der virtuellen Kamera sowie deren Orientierung festlegen. Aus der Angabe von neun Parametern — x-, y- und z-Koordinate der Kamera, x-, y- und z-Koordinate des Mittelpunktes der zu betrachtenden Szene und x-, y- und z-Koordinate die einen Vektor beschreiben, der nach oben zeigt — erstellt der Befehl `gluLookAt()` eine Betrachtungsmatrix und multipliziert diese mit der Modelviewmatrix. Die Ausrichtung der Kamera bestimmt, was tatsächlich auf dem Bildschirm ausgegeben wird.

3.3.6 Rendering

Obwohl OpenGL selbst nur über Befehle für die Erstellung einfacher geometrischer Objekte verfügt, lassen sich aus diesen komplexe Objekte unproblematisch erstellen. Für eine bessere Performance empfiehlt es sich, häufig benötigte geometrische Objekte mit Hilfe von *Display-Listen* zu erstellen. Dabei handelt es sich um Gruppen von OpenGL-Kommandos, die gemeinsam gespeichert werden und an einer beliebigen Stelle im Programm aufgerufen werden können. Dadurch ist es z.B. möglich das gleiche Objekt an mehreren Stellen in einer 3D-Szene zu positionieren, ohne die dazu notwenigen Berechnungen mehrmals auszuführen.Die Speicherung von Display-Listen ist abhängig von der verwendeten Hardware. Einige Grafikkarten speichern sie in optimierter Form direkt im Grafikspeicher. Die normale Abfolge einer einfachen OpenGL-Zeichenoperation lässt sich in folgende Schritte zerlegen:

1. Definition des Sichtvolumens
2. Erstellen der Kamera
3. Zeichnen der 3D-Szene
4. Ausgabe der 3D-Szene

Stereoskopisches Rendering: Um eine 3D-Szene stereoskopisch zu rendern ist es erforderlich, die gleiche Szene von zwei unterschiedlich positionierten und ausgerichteten virtuellen

[30] Vgl. Bourke 1999.

Kameras aus zu rendern. Die konkrete Vorgehensweise hängt von der Wahl des gewünschten Stereo-Formates ab. Soll die Ausgabe beispielsweise im Frame-Sequential-Format erfolgen, könnte wie folgt vorgegangen werden:

1. Definition des Sichtvolumens für das linke Auge
2. Erstellen der Kamera für das linke Auge
3. Zeichnen der 3D-Szene
4. Ausgabe der 3D-Szene
5. Definition des Sichtvolumens für das rechte Auge
6. Erstellen der Kamera für das rechte Auge
7. Zeichnen der 3D-Szene
8. Ausgabe der 3D-Szene

Die gezeichnete 3D-Szene ist in beiden Fällen identisch. Der einzige Unterschied liegt in der Definition von Sichtvolumen und Kamera. Durch die jedem Auge angepasste Perspektive ergeben sich bei geeigneter Bildtrennung zwei fusionierbare stereoskopische Halbbilder.

3.3.7 GLUT

GLUT, das *OpenGL Utility Toolkit*[31], ist ein Programminterface für die Erstellung plattformunabhängiger OpenGL-Programme[32]. GLUT ist event-basiert, d.h. bestimmten Ereignissen, z.B. Mausbewegungen oder Tastatureingaben, können Funktionen zugeordnet werden. Diese als Callbacks bezeichneten Funktionen werden automatisch aufgerufen, sobald ein registriertes Ereigniss stattgefunden hat[33]. Außerdem verfügt GLUT über Funktionen, die die Erstellung komplexerer 3D-Objekte aus einfachen geometrischen Objekten abstrahieren.

3.4 Objektive

Eine wichtige Eigenschaft von Objektiven ist die Art der erzeugten Abbildung, welche durch die Linseneigenschaften festgelegt sind. Zwei wichtige Abbildungsarten sind gnomonische[34] und äquidistante[35].

[31] `http://www.opengl.org/resources/libraries/glut/spec3/spec3.html` (Abruf am 04.03.2009)
[32] Vgl. Kilgard 1994.
[33] Vgl. Kilgard 1994, Kap. 7.
[34] Die englische Bezeichnung lautet rectilinear und ist auch im deutschen Sprachraum verbreitet.
[35] Vgl. Ray 1988, S. 201.

Gnomonisch: Die meisten Foto-Objektive bilden gnomonisch ab. Kennzeichnend ist, dass gerade Linien auch auf dem Abbild gerade erscheinen. Aufgrund zunehmender Verzerrungen zum Bildrand hin eignet sich diese Projektion nicht für große Bildwinkel.

Äquidistant: Diese Abbildung erzeugt stark sichtbare, tonnenförmige Verzeichnungen. Gerade Linien, die nicht durch das Projektionszentrum verlaufen, erscheinen im Bild gewölbt. Dieser Effekt wird mit zunehmender Entfernung von der Bildmitte stärker. Die meisten gängigen Fischaugenobjektive bilden äquidistant ab.

Die Abbildungsfunktionen dieser Projektionen lauten:

$$\text{gnomisch: } y = f * \theta \tag{3.3}$$

$$\text{equidistant: } y = f * tan(\theta) \tag{3.4}$$

worin y dem Abstand von der Bildmitte, f der Brennweite und θ dem Winkel zur optischen Achse entspricht. Keine reale Linse folgt vollständig ihrer theoretischen Abbildungsfunktion. Es kommt immer zu mehr oder weniger starken Abweichungen.

Weitere Parameter, die ein Objektiv kennzeichnen, sind:

Brennweite: Die Brennweite (engl. focal length) ist der Abstand zwischen einer Linse und ihrem Fokuspunkt. Bestimmt wird die Brennweite durch Form und Abmessungen der Linse. Je kleiner die Brennweite ist, desto größer ist der Bildbereich, den das Objektiv abbilden kann.

Bildwinkel: Die Größe des Bildwinkels gibt den Bereich des Bildraumes an, den ein Objektiv abbilden kann. Der Bildwinkel wird u.a. von der Brennweite und der Sensor- bzw. Filmgröße beeinflusst. Es werden horizontaler, vertikaler und diagonaler Bildwinkel unterschieden.

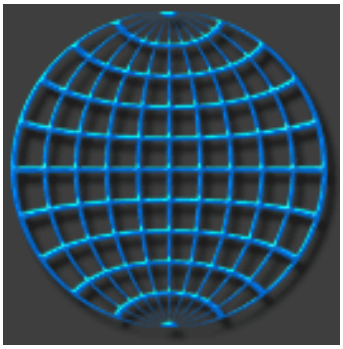
3.4.1 Fischaugenobjektive

Fischaugenobjektive sind eine spezielle Art von Foto-Objektiven, die durch eine besonders kurze Brennweite und eine äquidistante Projektion einen sehr großen Bildwinkel abbilden können. Es wird zwischen Vollformat-Fischaugen und Rundbild-Fischaugen unterschieden. Erstere füllen das gesamte Bild aus, erreichen aber dafür nur in der Diagonale ihren größten Bildwinkel. Rundbild-Fischaugen bilden sowohl den horizontalen als auch den vertikalen Bildwinkel vollständig ab, so dass sich ein rundes Bild ergibt (siehe Abbildung 3.9c). Die maximalen Bildwinkel liegen meist bei etwa 180°.

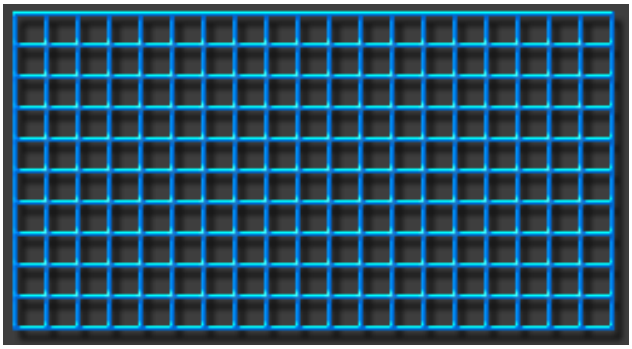
Für eine ausführliche Darstellung siehe Ray (1988).

3.5 Bild-Projektionen

Als Projektion wird die Abbildung eines dreidimensionalen Objektes bzw. seiner Fläche auf eine zweidimensionale Fläche bezeichnet. Beispielsweise stellt eine Weltkarte eine Projektion der Erdoberfläche auf eine Karte dar. Auch eine Fotografie ist eine Projektion, denn es wird der dreidimensionale Raum auf die zweidimensionale Chip- oder Filmfläche abgebildet. Bei allen Projektionen sind Verzerrungen unvermeidlich. Fehler können sich in Änderungen der Länge einer Linie, des Winkels zwischen zwei Linien oder der Größe und Form einer Strecke bemerkbar machen[36].



(a) Fischauge



(b) Equirektangular

(c) Fischauge

(d) Equirektangular

Abbildung 3.9: oben: schematische Darstellung von Fischaugen- und Equirektangularprojektion[37], unten: Beispielbilder

[36] Vgl. `http://de.wikipedia.org/w/index.php?title=Kartennetzentwurf&stableid=56190134` (Abruf am 02.03.2009)

[37] Quelle von (a) und (b): `http://www.cambridgeincolour.com/tutorials/image-projections.htm` (Abruf am 23.02.2009)

Äquidistante Fischaugenprojektion: Äquidistante Fischaugenprojektionen[38] zeichnen sich dadurch aus, dass der Bildwinkel direkt proportional zum Radius des projizierten Bildes ist. Gerade Linien, die nicht durch das Zentrum der Projektion verlaufen, werden gebogen (siehe Abbildungen 3.9a und 3.9c). Radial nach aussen laufend, erscheint das resultierende Bild zunehmend gestaucht.

Equirektangulare Projektion: Eine equirektangulare Projektion[39] ist eine spezielle Form der zylindrischen äquidistanten Projektion, bei der die horizontale Koordinate dem Breitengrad und die vertikale Koordinate dem Längengrad entspricht[40]. Die auftretenden Stauchungen nehmen in Richtung der Pole zu.

3.6 Stereo-Panoramen

3.6.1 Allgemeines zu Panoramabildern

Panoramabilder ermöglichen die Aufnahme und Darstellung visueller Szenen mit einem bis zu 360° horizontal umfassenden Sichtfeld. Der Betrachterstandort befindet sich bei traditionellen Panoramen immer im Zentrum der Projektion. Dieser Betrachterpunkt ist fix und durch den Standpunkt der Kamera während der Aufnahme festgelegt (*single viewpoint*)[41]. Die Aufnahme kann mit Panoramakameras, speziellen Spiegelsystemen[42] oder durch das Zusammensetzen (*Stitching, Mosaicing*) mehrerer Einzelbilder erfolgen[43]. Dabei reichen je nach Bildwinkel des verwendeten Objektivs schon sehr wenige Bilder.

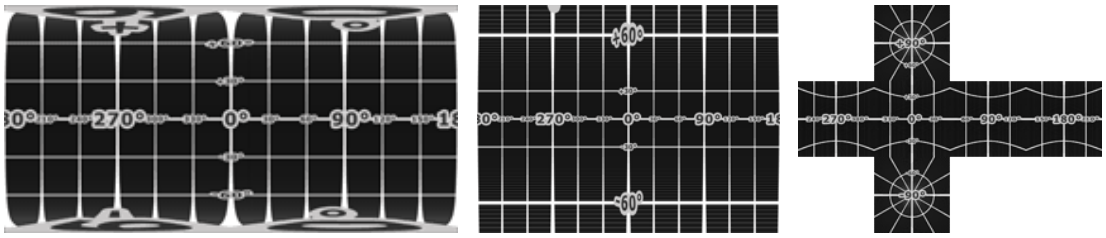
Für die Darstellung können Panoramabilder auf ausgedehnte ebene Flächen, Zylinder oder Sphären projiziert werden[44]. Je nach Art der Fläche werden grundsätzlich drei Panorama-Formate unterschieden[45].

Sphärisches Panorama: Bei sphärischen Panoramabildern, auch Kugelpanoramen genannt, kann der vertikale Sichtbereich bis zu 180° betragen. Innerhalb eines Raumes kann also sowohl der Boden als auch die Decke dargestellt werden. Die Betrachterposition befindet sich im Zentrum der Kugel und ermöglicht so ein vollständiges Betrachten der umgebenden Szene in alle Richtungen.

[38] Es gibt verschiedene Fischaugenprojektionen. Da die meisten Objektive eine äquidistante Projektion verwenden, soll es hier nur um diese gehen.
[39] Auch als Rektangularprojektion bezeichnet.
[40] `http://mathworld.wolfram.com/CylindricalEquidistantProjection.html` (Abruf am 07.03.2009).
[41] Vgl. Peleg, Ben-ezra und Pritch 2001; Bourke 2006.
[42] Vgl. z.B. Nayar 1997.
[43] Vgl. Shum u. a. 1997.
[44] Vgl. Duke Gledhilla und Clarke 2003.
[45] Vgl. Bourke 2002.

Zylindrisches Panorama: Bei einem zylindrischen Panorama ist die Darstellung von Boden und Decke prinzipiell nicht möglich. Das darstellbare vertikale Blickfeld ist auf einen Winkel deutlich kleiner als 180° begrenzt. Ein Vorteil der zylindrischen Darstellung ist die Möglichkeit der großformatigen Projektion auf Leinwände, die den Betrachter vollständig umgeben können[46].

Kubisches Panorama: In einem kubischen Panorama kann ebenso wie in einem sphärischen Boden und Decke dargestellt werden. Damit ist ein Betrachten aller Raumrichtungen möglich. Auch dieses Format eignet sich für großflächige Projektionen und wird z.B. in CAVE-Umgebungen eingesetzt[47].



(a) Sphärisch **(b)** Zylindrisch **(c)** Kubisch

Abbildung 3.10: Panoramaformate[48]

Panorama-Art	max. darstellbares Blickfeld		Boden und Decke darstellbar
	horizontal	vertikal	
Sphärisch	360	180	ja
Zylindrisch	360	<180	nein
Kubisch	360	180	ja

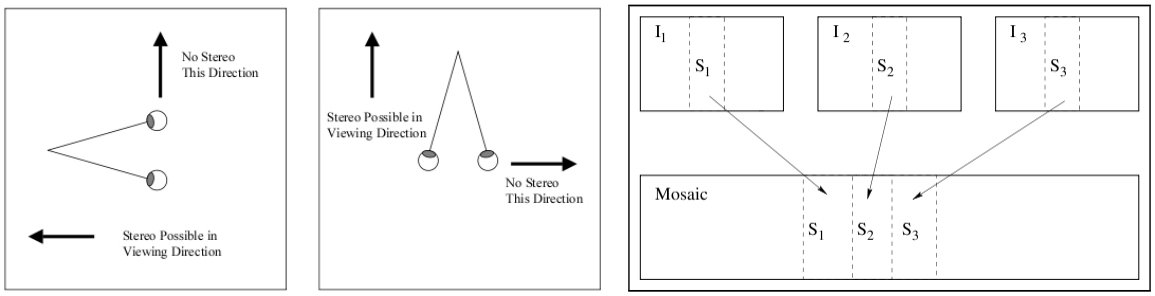
Tabelle 3.1: Maximale Blickwinkel verschiedener Panorama-Arten

3.6.2 Aufnahmemethoden

Stereoskopische Panoramen unterscheiden sich von traditionellen monoskopischen Panoramen dadurch, dass sie zwei Betrachterpunkte benötigen. Während für die Erzeugung monoskopischer Panoramen eine Kamera um ihre optische Achse rotieren muss[49], ist es für stereoskopische Panoramen nicht ausreichend zwei Kameras um ihre jeweilige optische Achse rotieren zu lassen. Die für Stereo-Bilder notwendigen Parallaxen wären nur für Blickrichtungen gegeben,

[46] *AVIE - Advanced Visualisation and Interaction Environment.*
[47] Vgl. Cruz-Neira, Sandin und DeFanti 1993, S. 136.
[48] Quelle: `http://panoramen.hohenauer.ch/theorie/panoramas2_d.php` (Abruf am 02.03.2009)
[49] Vgl. Duke Gledhilla und Clarke 2003; Kerr 2008.

die annähernd senkrecht zur Verbindungslinie beider Kameras verlaufen (siehe Abbildung 3.11a).



(a) Parallaxe in alle Raumrichtungen ist mit einer statischen Kameraanordung nicht möglich

(b) Mosaicing

Abbildung 3.11: (a) Viewpoints für Stereoskopische Panoramen und (b) Mosaicing (aus Peleg und Ben-Ezra (1999))

Um korrekte stereoskopische Panoramen aufzunehmen, müssen zwei Kameras auf einem gemeinsamen Stativ rotiert werden[50]. Jede Kamera erzeugt ein Panoramabild für die jeweilige Augenperspektive. Abbildung 3.12 zeigt den Unterschied zwischen monoskopischer und stereoskopischer Aufnahme. Bourke (2002) weist außerdem darauf hin, dass die Kameras parallel zueinander ausgerichtet sein müssen[51]. Es ist auch möglich, die Bilder nach dem gleichen Prinzip, aber mit nur einer Kamera aufzunehmen. Dazu wird eine Kamera auf mittels Schiene auf dem Stativ befestigt und zuerst links ausgerichtet um das Panorama für die linke Augenperspektive aufzunehmen. Danach wird die Kamera nach rechts verschoben um die rechte Augensperspektive aufzunehmen.

Als Schrittweite für die Rotation empfiehlt Bourke (2006) maximal 1°. Bei einem 360°-Panorama sind so also 360 Bilder pro Auge notwendig, für das finale Stereopanorama also 720 Einzelbilder. Für jede Augenperspektive werden aus den aufgenommenen Einzelbildern Streifen aus der Bildmitte genommen und sequentiell zusammengefügt (siehe Abbildung 3.11b). Die Breite der Streifen in Grad entspricht der Schrittweite der Kamerarotation. Da die horizontale Öffnung der Kamera auf diese Weise faktisch verkleinert wird, spricht Bourke auch von *finite slit approximation*. Je kleiner die Öffnung wird, desto stereotreuer wird das resultierende Stereopanorama.

Eine alternative Methode, um stereoskopische Panoramen mit nur einer Kamera aufzunehmen, wird in Peleg und Ben-Ezra (1999) vorgestellt. Dabei wird eine Kamera um eine hinter dem optischen Zentrum liegende Achse rotiert. Anstatt das Panorama aus den Mittelstrei-

[50] Vgl. Bourke 2006; Peleg und Ben-Ezra 1999; Peleg, Pritch und Ben-Ezra 2000; Peleg, Ben-ezra und Pritch 2001.

[51] Vgl. auch Bourke 2006.

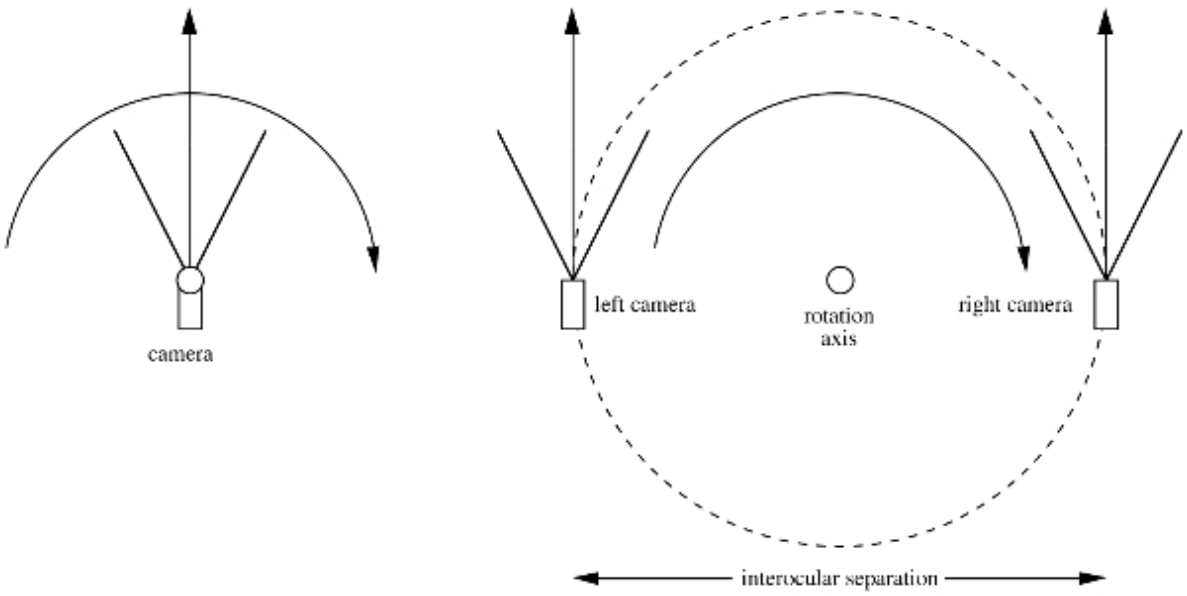


Abbildung 3.12: Aufnahmemethode für monoskopische (links) und stereoskopische (rechts) Panoramabilder (aus Bourke (2006, S. 2))

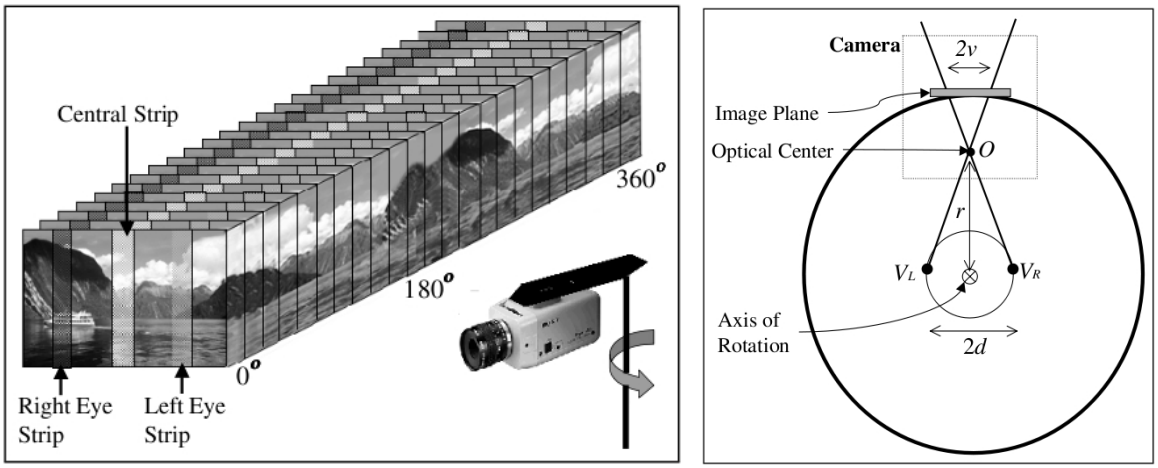
fen der Einzelbilder zusammenzufügen, werden Streifen von den Seiten der Bilder gewählt. Abbildung 3.13a zeigt das Prinzip. Diese Methode besitzt den Vorteil, dass die Anzahl der benötigten Einzelbilder halbiert wird. Außerdem umgeht sie die mögliche Fehlerquelle einer Szenenänderung während der zwei Aufnahme-Durchläufe für beide Augenperspektiven. Ein erheblicher Nachteil ist, dass diese Methode einen höheren Konstruktionsaufwand und mehr Platz am Aufnahmeort erfordert. Die Entfernung zwischen Rotationsachse und optischem Zentrum der Kamera ergibt sich entsprechend Abbildung 3.13b aus:

$$l/2v = r/2d \tag{3.5}$$

Dabei entspricht l der Entfernung zwischem dem optischen Zentrum der Kamera (Optical Center) und dem Bildsensor (Image Plane). $2v$ entspricht dem Abstand der Bildstreifen, welche für die Konstruktion von rechtem und linkem Panorama verwendet werden. r gibt die Distanz vom Rotationszentrum (Axis of Rotation) zum optischen Zentrum der Kamera an. $2d$ ist der horizontale Abstand der beiden virtuellen Kameras.

Bourke weist auf zwei wichtige Aspekte bei der Aufnahme stereoskopischer Panormabilder hin[52]. So können monoskopische Panoramen auch fehlerbehaftet erstellt werden, z.B. durch Verschiebung des Betrachterstandpunktes oder approximatives Zusammensetzen der Einzelbilder. Diese Fehler sind im Ergebnis nicht zwingend sichtbar. Aufgrund der bei stereo-

[52] Vgl. Bourke 2006, S. 3.



(a) Mosaicing mit einer Kamera **(b)** Kamera-Model

Abbildung 3.13: Methode und Kamera-Model für Ein-Kamera-Panorama-Systeme (aus Peleg und Ben-Ezra (1999))

skopischen Panoramen zusätzlichen Tiefenwahrnehmung, sind derartige Fehler hierbei meist deutlich wahrnehmbar. Außerdem ist es bei stereoskopischen Panoramabildern nicht möglich, über die Sichtachse zu rotieren, zum Beispiel durch das Schrägstellen des Kopfes. Durch die Aufnahme von zwei in gleicher Höhe gelegenen Positionen aus, entweder durch eine Kamera mit zwei Objektiven oder durch zwei nebeneinander liegende Kameras, ist die parallaktische Verschiebung auf die horizontale Ebene festgelegt, so dass es ein festgelegtes Lot gibt. Ein Schrägstellen des Kopfes führt demnach zu mehr oder weniger ausgeprägten stereoskopischen Fehlern.

3.6.3 Entfernung zur Nullparallaxe

Die Ebene in der Objekte keine parallaktische Verschiebung aufweisen, das heißt auf korrespondierende Netzhautpunkte abgebildet werden[53], wird als Nullparallaxe bezeichnet. Die Entfernung f_0 zur Nullparallaxe (*focal length*) wird bei festem Augenabstand gemäß Formel 3.1 durch den Parallaxenwinkel ϕ bestimmt.

Für die stereoskopische Darstellung ist diese Entfernung von essentieller Bedeutung. Objekte die näher als f_0 sind, scheinen vor dem Bildschirm zu liegen (negative Parallaxe), Objekte weiter als f_0 scheinen hinter der Bildschirmebene zu liegen (positive Parallaxe)[54]. Die Kontrolle von f_0 kann auf zwei Arten erfolgen, je nach Ausrichtung der Kameras zum Aufnahmezeitpunkt.

[53] Siehe Abschnitt 3.1.2.
[54] Vgl. Bourke 2006, S. 5.

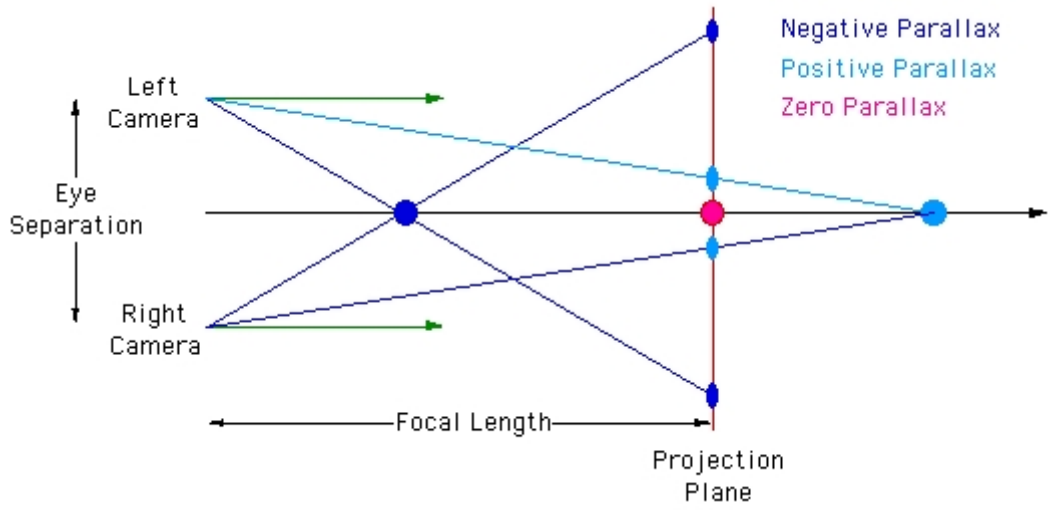


Abbildung 3.14: Nullparallaxe, negative und positive Parallaxe (aus Bourke (1999))

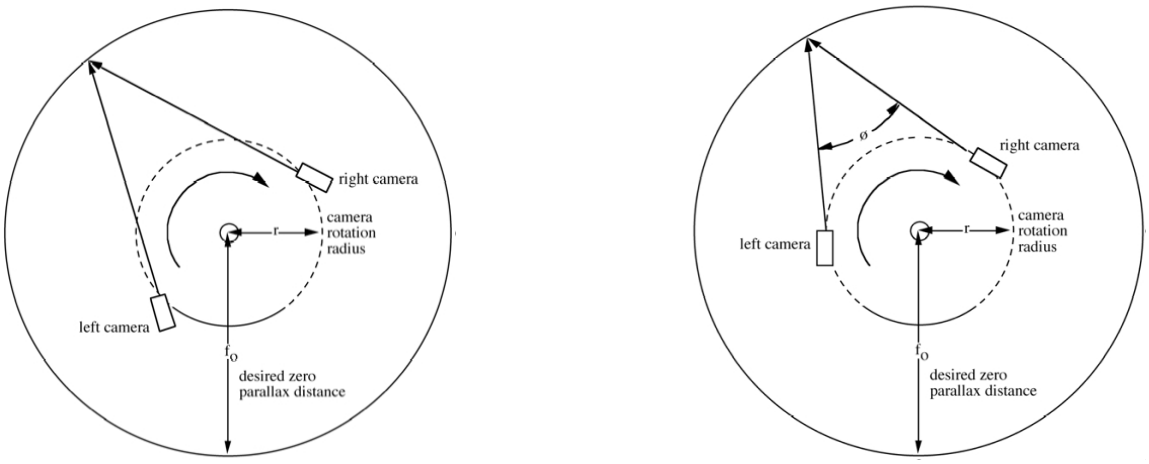
Konvergente Kameraausrichtung: Um eine feste Entfernung zur Nullparallaxe bereits zum Aufnahmezeitpunkt festzulegen, muss eine konvergente Ausrichtung der Kameras gewählt werden (siehe Abbildung 3.15a). Die Entfernung f_0 berechnet sich dann entsprechend Formel 3.1. Dieser Ansatz entspricht der in Abschnitt 3.3.4 beschriebenen inkorrekten 'Toe-in'-Methode für das Rendern einzelner Stereoszenen. Beim Erstellen von Panoramen aus schmalen Streifen vieler Einzelbilder treten die beschriebenden Probleme allerdings nicht auf, so dass der Ansatz korrekt Ergebnisse liefert[55].

Parallele Kameraausrichtung: Werden die Kameras parallel ausgerichtet, so liegt die Nullparallaxe in den resultierenden Panoramabildern im Unendlichen. Um dennoch eine bestimmte Entfernung zur Nullparallaxe zu erreichen, können die beiden Panoramabilder gegeneinander verschoben werden. Da die virtuelle Kameras dann nicht mehr punktsymmetrisch zum Mittelpunktes des Augenabstands (siehe Abbildung 3.15b) liegen, muss Formel 3.1 folgendermaßen modifiziert werden:

$$\phi = 2 * arcsin(r/f_0) \tag{3.6}$$

Dabei entspricht r dem halben Augenabstand und f_0 dem Abstand zur Nullparallaxe. Weil Menschen nur sehr kleine Stereo-Disparitäten fusionieren können gilt $sin(x) \approx tan(x)$, so dass die resultierenden Disparitäten praktisch gleich sind[56].

[55] Vgl. Bourke 2006, S. 5-6.
[56] Vgl. Peleg und Ben-Ezra 1999, S. 5.



(a) konvergente Ausrichtung **(b)** parallele Ausrichtung

Abbildung 3.15: Geometrie für Parallaxenbestimmung in Abhängigkeit der Kameraausrichtung (aus Bourke (2006, S. 5-6))

3.6.4 Darstellung stereoskopischer Panoramen

Für die Darstellung stereoskopischer Panoramen muss jedes Panorama auf eine geeignete, den Betrachter umschließende, Fläche projiziert werden. Mit einer Graphik-Bibliothek wie OpenGL kann das zum Beispiel erfolgen, indem eine Textur auf die Innenseite eines umliegenden 3D-Objektes (Sphäre, Zylinder oder Quader) projiziert wird. Wie in Abschnitt 3.6 erläutert, bieten sphärische und kubische Projektionen den Vorteil auch Boden und Decke der aufgenommenen Szene darstellen zu können.

Im Gegensatz zum Rendering von arbiträren 3D-Szenen kann im Fall der Darstellung stereoskopischer Panoramen auf eine gesonderte Berechnung von asymetrischen Sichtvolumina verzichtet werden. Die stereoskopische Information befindet sich in diesem Fall nicht in der Struktur der 3D-Szene, sondern in den für jedes Auge speziell aufgenommenen Panoramabildern. Dadurch ist es ausreichend, ein Objekt (Sphäre, Zylinder oder Quader) mit symmetrischem Sichtvolumen (siehe Abbildung 3.8a) zu rendern.

4 Erzeugung eines stereoskopischen Vollpanoramas

4.1 Vorbemerkungen

In Kapitel 3 wurde gezeigt, dass es für die Erzeugung stereoskopischer Vollbild-Panoramen notwendig ist eine große Anzahl an Einzelbildern aufzunehmen und diese sehr präzise zusammenzufügen. Im Gegensatz zu monoskopischen Panoramen, ist die Anzahl der benötigten Einzelbilder sehr hoch, so dass eine Automatisierung sowohl von Aufnahme, als auch von den in der Nachbearbeitung notwendigen Schritten unabdingbar war. Um dies zu ermöglichen wurde eine Aufnahme- und Verarbeitungsstrecke entwickelt, deren Entwicklung und Funktionsweise in den folgenden Abschnitten dokumentiert wird.

Dabei wird zuerst auf die notwendige Hardware eingegangen. Erleichternd für die Realisierung der Aufnahmen war die Möglichkeit das FABIAN-Messsystem verwenden zu können. Damit stand ein System zur Verfügung, welches viele der benötigten Funktionen, wie hochgenaue Kamerarotation und vollständig automatisierbare und parametrisierbare Aufnahmemöglichkeit, bereits integriert hat.

In Abschnitt 4.7 wird die Nachverarbeitung der aufgenommenen Einzelbilder beschrieben. Dafür werden die einzelnen Schritte anhand eines typischen Verarbeitungsdurchgangs erläutert.

4.2 Kamera und Objektiv

Als Kamera wurde eine Canon EOS 450D mit einem Sunex Fischaugen-Objektiv eingesetzt. Das Sunex-Objektiv ist ein Rundbild-Fischaugen-Objektiv mit einer Brennweite von 5,6 mm und erlaubt die Aufnahme von Bildern, die sowohl horizontal als auch vertikal, einen Bildbereich von 185°abbilden. Damit ist es möglich, den vertikalen Sichtbereich von 180° vollständig zu erfassen und ein 360°x 180°-Panorama mit einem einmaligen Umlauf der Kamera erzeugen. Die Kombination der Einzelbilder zu einem Panorama wird dadurch sehr erleichtert, da

es nicht notwendig ist, mehrere horizontale Ebenen zusammenzufügen. Ein weiterer Vorteil eines Fischaugen-Objektivs ist die große Schärfentiefe.

4.3 Verwendete Aufnahmemethode

Als Aufnahmemethode wurde die in Abschnitt 3.6.2 besprochene Variante eines Zwei-Kamera-Systems nach dem Muster von Bourke (2006) gewählt. Die alternative Methode nach Peleg und Ben-Ezra (1999) wurde trotz der Aussicht auf eine Halbierung der notwendigen Einzelbildanzahl aufgrund des erhöhten Konstruktionsaufwand verworfen. Mit der zur Verfügung stehenden Technik (siehe nächster Abschnitt) hätte sich die Notwendigkeit ergeben einen Abstand von etwa 92 cm zwischen dem optischen Zentrum der Kamera und dem Rotationszentrum einzuhalten. Bei der verwendeten Kamera-Objektiv-Kombination beträgt der Abstand zwischen Bildsensor und optischem Zentrum etwa 10,8 cm. Da das Fischaugen-Objektiv lediglich auf 2/3 des 22,2 mm breiten Bildsensors abbildet und dieser Bereich aufgrund der Verzerrungen des Objektivs nicht komplett nutzbar ist, wurde von einem maximalen Abstand der Bildstreifen von 1/3 Sensorbreite, also 0,74 cm, ausgegangen. Bei einem angenommenen Augenabstand von 6,3 cm, ergibt sich demnach der notwendige Abstand r entsprechend Formel 3.5 zu:

$$10,8cm/0,74cm = r/6,3cm \Leftrightarrow r = 92\ cm$$

4.4 Stativsystem und Kamerahalterung

Als Kamera-Stativ wird das von Lindau (2006) für die Messung binauraler Raumimpulsantworten entwickelte Messsystem FABIAN verwendet. Es beinhaltet ein hochpräzises servoelektrisches Schwenk-Neige-Gelenk vom Typ Amtec Robotics PW-070, auf dass die Kamera mit Hilfe zweier Adapterplatten aufgesetzt werden kann. Das Amtec-Gelenk erlaubt Rotationsbewegungen von -180° bis +180°, also einen vollständigen 360°-Schwenk. Die minimale Rotationsschrittweite liegt bei 0,1° mit einer Präzision von 0.004° und erfüllt damit die in Abschnitt 3.6 erörterten Bedingungen an das zu verwendende Kamera-System.

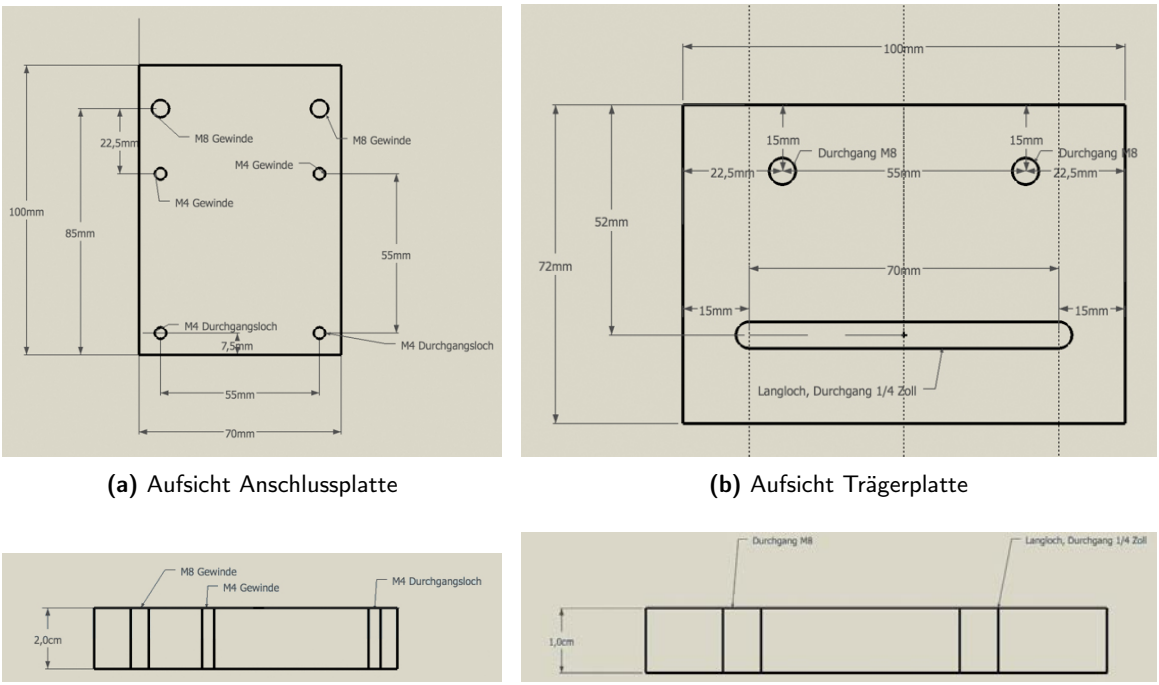
Zusätzlich zum benötigten Schwenk-Motor existiert für das FABIAN-Messsystem außerdem eine komplette MatLab-basierte Steuerungssoftware. Diese erlaubt vollautomatische Messläufe nach festgelegten Parametern. Für die Belange dieser Arbeit wurde die Steuerungssoftware um eine Schnittstelle zum Auslösen der Kamera erweitert. Somit ist die Aufnahme von 360 Einzelbildern pro Auge ohne äußeren Eingriff möglich.

Ein weiterer Vorteil der Verwendung des FABIAN-Systems liegt darin, das mit diesem System auch die Binauralmessungen durchgeführt werden, zu denen die Stereo-Panoramen als visuelle Ergänzung dienen sollen. Das System ist also am Ort der Aufnahme bereits aufgebaut. Es muss lediglich der für die Binauralmessungen verwendete Kunstkopf abmontiert und durch die Kamera ersetzt werden. Somit ist es möglich, mit einem einzigen System sowohl die Audio- als auch die Bilddaten zu generieren. Der zusätzliche Materialaufwand für die Erstellung der Stereo-Panoramen begrenzt sich auf die Kamera, das Objektiv und die Adapterplatten.

Abbildung 4.1: FABIAN mit montierter Kamera in Position für die Aufnahme der rechten Augenperspektive

Um die Kamera auf dem in FABIAN verwendeten Amtec-Gelenk zu befestigen wurden zwei Adapterplatten konstruiert. Die in Abbildung 4.2a abgebildete Anschlussplatte wird direkt auf die Kopfplatte des Amtec-Gelenks geschraubt. Die Trägerplatte in Abbildung 4.2b wird über zwei M8-Löcher an der Anschlussplatte befestigt. Die Kamera wird mit einer handelsüblichen Rändl-Schraube im Langloch der Trägerplatte fixiert. Um den Konstruktionsaufwand möglichst gering zu halten wurde auf eine separate Skaleneinteilung zur Ausrichtung der Kamera verzichtet. Für die Aufnahme von Bildern für die Perspektive des linken Auges muss die Kamera im Langloch nach links verschoben werden. Für die Bilder der rechten Augenperspektive muss die Ausrichtung rechts im Langloch erfolgen. Mit dieser Konfiguration werden Bilder der beiden verschiedenen Blickpunkte mit einer maximalen Stereobasis von 7 cm aufgenommen. Dieser Abstand wurde bewußt etwas größer gewählt als der in Abschnitt 3.1 genannte Durchschnittswert von 6,3 cm. Im Rahmen der Anwendung im Fachbereich Audiokommunikation werden vor allem größere Räume, beispielsweise Konzertsäle, aufgenommen. Durch die vergrößerte Stereobasis lässt sich der räumliche Eindruck in größeren Entfernungen vom Aufnahmeort etwas erhöhen.

Die Adapterplatten wurden so konstruiert, dass der Objektivmittelpunkt des verwendeten



(a) Aufsicht Anschlussplatte

(b) Aufsicht Trägerplatte

(c) Profil Anschlussplatte

(d) Profil Trägerplatte

Abbildung 4.2: Anschlussplatte und Trägerplatte für die Befestigung der Kamera auf dem FABIAN-Messsystem

Kamera-Objektiv-Systems in der gleichen Horizontalebene liegt, wie die Augen des für die Binauralmessung verwendeten Kunstkopfes. Die Gesamthöhe der Adapterplatten beträgt 3 cm (siehe Abbildungen 4.2c und 4.2d). Der Objektivdurchmesser beträgt 7 cm, so dass der Linsenmittelpunkt bei 3,5 cm anzunehmen ist. Relativ zum Kameragehäuse hat das Objektiv einen vertikalen Versatz von 0,3 cm. Damit ergibt sich relativ zur Oberkante der Amtec-Kopfplatte ein vertikaler Abstand zum Objektivmittelpunkt von 6,8 cm.
Außerdem wurden die Platten so konzipiert, dass der Brennpunkt der Fischaugenlinse bei zentrierter Ausrichtung der Kamera genau über dem Rotationszentrum des Aufnahmesystems liegt. Die Kamerahalterung ist also genau auf die verwendete Kamera-Objektiv-Kombination ausgelegt. Sollte sich die Notwendigkeit ergeben, eine andere Kamera oder ein anderes Objektiv zu verwenden, müssen die Adapterplatten entsprechend angepasst bzw. neu konstruiert werden.

Die Canon EOS 450D verfügt über eine Fernbedienungsbuchse an die ein RS-60E3-Auslösekabel angeschlossen werden kann. Um die Kamera mit der FABIAN-Steuerungssoftware fernauslösen zu können, wurde ein Adapter angefertigt, der den seriellen Port eines PC mit der Fernbedienungsbuchse verbindetAbbildung 4.3. Die Steuerungssoftware des FABIAN-Messsystems wurde so erweitert, dass die Kamera durch das Senden einer Spannung auf den seriellen Port ausgelöst werden kann (siehe Abbildung 4.3).

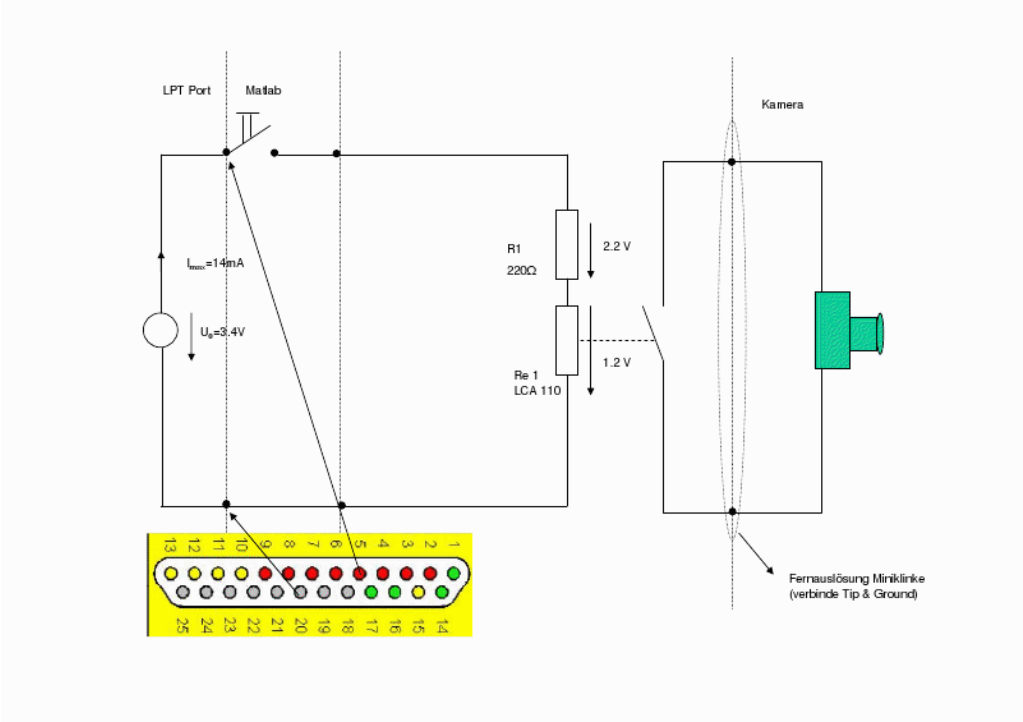


Abbildung 4.3: Schaltplan für seriellen Auslöseadapter

4.5 Aufnahme

4.5.1 Vorbereitungen

Kameraeinstellungen

Um alle Einzelbilder unter den gleichen Bedingungen aufzunehmen, muss die Kamera im manuellen Modus betrieben werden. Die ISO-Einstellung ist auf den niedrigst-möglichen Wert zu stellen, um unnötiges Rauschen zu vermeiden. Der Weißabgleich muss auf einen festen Wert eingestellt werden, der am besten vor dem Beginn der Aufnahme-Session einmalig festgelegt wird. Dies kann durch die Aufnahme eines weißen Blatts erfolgen. Der Weißabgleich dieser Aufnahme lässt sich in der Canon EOS 450D speichern und auf alle folgenden Bilder anwenden.

Der Spiegel der Kamera sollte vor jedem Auslösen hochgeklappt sein, um eventuell entstehende Verwacklungen zu verhindern. Solch ein Verhalten lässt sich im Kamera-Menu unter 'Individualfunktionen' einstellen. Als Farbbraum ist der im Vergleich zu sRGB größere Adobe RGB zu wählen. Außerdem wird empfohlen die automatische Erstellung von JPG-komprimierten Bildern zu deaktivieren, da dies mehr Speicherplatz verbrauchen würde und die komprimierten Bilder für die Nachverarbeitung nicht benötigt werden.

Konfiguration von FABIAN

Für die MatLab-basierte FABIAN-Steuerungssoftware gibt es eine Konfigurationsdatei, welche für die Verwendung von FABIAN als Kamera-Stativ angepasst werden muss. Unter anderem muss das Rotationsintervall (-180° bis + 180°), die Rotationsschrittweite (1°) und die Pausendauer zwischen zwei Auslösesignalen eingestellt werden. Letzteres muss so abgestimmt sein, dass die Pause ausreichend lang ist, um die Kamera in die nächste Position zu drehen.

Aufnahmeraum

Eventuell ist es notwendig, vor Beginn der Aufnahme am Aufnahmeort selbst Änderungen vorzunehmen. So muss sichergestellt werden, dass sich keine Objekte zu nah an der Linse befinden. Diese würden im Panoramabild als nicht zu fusionierende Doppelbilder erscheinen bzw. dem Betrachter eine stark konvergierende Augenstellung beim Betrachten abverlangen. Beides kann den resultierenden Stereoeffekt stark beeinträchtigen.

4.5.2 Aufnahmesession

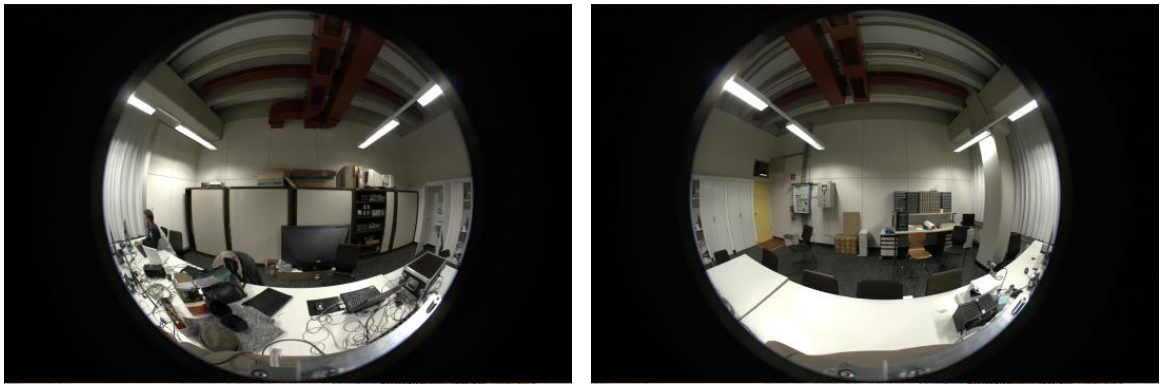
Sobald die Kamera montiert ist und die korrekten Einstellungen an der Kamera und in den FABIAN-Konfigurationsdateien vorgenommen wurden, kann die automatische Datenaquise beginnen. Dazu wird auf dem FABIAN-Messrechner MatLab gestartet und dort die FABIAN-Steuerungssoftware aufgerufen.

Das Ergebnis eines ersten fehlerfreien Durchlaufs sollten 360 Einzelbilder[1] einer Augenperspektive sein (siehe Abbildung 4.4). Nachdem die Bilder von der Speicherkarte der Kamera auf ein geeignetes Speichermedium übertragen wurden, kann die notwendige zweite Aufnahmereihe für die andere Augenperspektive gestartet werden.

4.5.3 Organisation des Dateisystems

Um mit der großen anfallenden Datenmenge effizient umzugehen, ist es unabdingbar die Verzeichnise gut zu strukturieren. Das erleichtert einerseites den Überblick, andererseits lässt sich nur so eine automatische Weiterverarbeitung der Bilder ermöglichen. Das Python-Script *panorama.py* erfordert eine feste Verzeichnisstruktur wie in Abschnitt A.2.2.
Für jedes zu erstellende stereoskopische Panorama muss ein Verzeichnis erstellt werden, dessen Name die Session eindeutig kennzeichnet. Unterhalb dieses Verzeichnises sollten zwei Unterverzeichnise `left` und `right` erstellt werden.

[1] Bei einer angenommenen Rotationsschrittweite von 1°.



(a) 0° **(b)** 90°

Abbildung 4.4: 2 von 360 resultierenden Fischaugenbildern einer Serie der linken Augenperspektive

Neu aufgenommene Bilder müssen in das der Augenperspektive entsprechende Verzeichnis in dem Unterverzeichnis `raw` abgelegt werden[2].

4.6 Kalibrierung

Um die Einzelbilder im Folgenden weiterverarbeiten zu können, ist es notwendig die Kamera-Objektiv-Kombination einmal zu kalibrieren. Die Kalibrierung ist erforderlich, weil jedes Objektiv Verzerrungen verursacht, die für eine korrekte Nachbearbeitung korrigiert werden müssen. Bei den Verzerrungen handelt es sich um:

1. den von der Kamera aufgenommene reale Bildwinkel (Field of view)
2. drei Parameter für die verursachte Tonnen-, Kissen- und wellenförmige Verzeichnung
3. die Linsen-Offset-Parameter (x und y), die zur Korrektur von Abweichungen der Linsenmitte zur Bildmitte verwendet werden

Die Kalibrierung kann mit dem Programm *Hugin* vorgenommen werden. Dieses Programm ist im Grunde eine grafische Benutzeroberfläche für die von Prof. Helmut Dertsch entwickelte Open Source Software-Sammlung Panorama Tools[3], bietet aber noch einige Zusatzprogramme die für die vorliegende Arbeit nützlich sind (siehe folgender Abschnitt 4.7).

Eine detaillierte Anleitung zum Kalibrieren des fotografischen Systems ist in Abschnitt A.4 zu finden.

[2]Siehe Abschnitt A.2.2.
[3]http://panotools.sourceforge.net

4.7 Panorama-Erstellung

Eines der Ziele dieser Arbeit ist es eine preiswerte und so weit wie möglich automatisierbare Methode zu entwickeln, um Stereo-Panoramen zu erstellen. Daher fiel die Wahl der Hilfsmittel in den meisten Fällen auf frei verfügbare Open Source Software und vor allem auf Programme, die sich auf einem Linux-System von der Kommandozeile aus aufrufen und steuern lassen. Dadurch ist es möglich Arbeitsabläufe einmal festzulegen und anschließend vollautomatisch ablaufen zu lassen. Wie im Folgenden detailliert beschreiben, erfolgt die Extraktion der aufgenommenen Bilder aus der Kamera mit dem Kommandozeilen-Programm *dcraw*, die Umwandlung der Fischaugen-Bilder mit dem zur *Hugin*-Suite gehörenden Programm *nona*, sowie das Zusammenfügen (emphStitchen) der Einzelbilder zu einem 360°-Panorama mit einem selbstgeschriebenen Python-Programm. Der komplette Arbeitsablauf zur Erstellung der Stereo-Panoramen kann damit voll automatisiert erfolgen. Der Zeitaufwand für die Erstellung eines Stereo-Panorama-Paars, ausgehend von den Kamera-RAW-Bildern, liegt bei unter einer Stunde[4].

Die vollständige Nachbearbeitung, ausgehend von den einzelnen Bildern im kameraspezifischen RAW-Format bis hin zum fertigen Panorama erfolgt mit dem Python-Skript *panorama.py*. Es erfüllt drei Aufgaben, die im folgenden ausführlicher erläutert werden.

Formatumwandlung: Die Ausgangsbilder liegen in einem kameraspezifischen Daten-Format vor, welches kaum verarbeitete Rohdaten des Bildsensors enthält. Die Bilder können mit Grafik-Software nicht bearbeitet werden, sondern müssen zunächst mit einem sogenannten RAW-Converter in ein kompatibles Datenformat konvertiert werden.

Bildtransformation: Transformation der Bilder aus der Fischaugenprojektion in eine Rektangularprojektion. Die Fischaugen-Projektion eignet sich aufgrund der mit dem Abstand vom Bildzentrum stark zunehmenden Verzerrungen nicht dafür, Bildstreifen zu extrahieren und zu einem Panoramabild zusammenzufügen. Im Original eigentlich senkrechte Linien sind zu den Rändern des Bildes hin zunehmend gekrümmt. Mit der Überführung in eine Equirektangular-Projektion können die Bilder so entzerrt werden, das senkrechte gerade Linien auch im Bild senkrecht dargestellt werden. Da das finale Panoramabild ebenfalls in einer Equirektangular-Projektion vorliegen muss, um es mit einem sphärischen Panorama darzustellen, ist dieser Transformationsschritt der kürzeste Weg, die Einzelbilder in ein verwertbares Format zu überführen.

Mosaicing / Stitching: Zusammenfügen der Einzelbilder zu einem Panoramabild. Die nun equirektangularen Einzelbildern können mit dem in Abschnitt 3.6.2 beschriebenen Mosaicing zusammengefügt werden. Das Ergebnis sind zwei stereoskopische Halbbilder,

[4] Siehe Abschnitt 4.7.5.

```
$ cd panorama_creator/scripts
$ python panorama.py NAME left all 1        % komplette Nachbearbeitung
$ python panorama.py NAME left raw2fish     % Extraktion der RAW-Dateien
$ python panorama.py NAME left fish2rect    % Umwandlung nach Rektangular
$ python panorama.py NAME left rect2pano    % Stitchen des Panoramas
```

Listing 4.1: mögliche Aufrufe von panorama.py

```
$ cd panorama_creator/scripts
$ create-all NAME           % komplette Nachbearbeitung
$ extract-raw NAME          % Extraktion der RAW-Dateien
$ create-rect NAME          % Umwandlung nach Rektangular
$ create-panorama NAME      % Stitchen des Panoramas
```

Listing 4.2: Aufruf der Shell-Skripte, bearbeitet werden jeweils die Bilder beider Augenperspektiven

die in 360° x 180° -Equirektangular-Projektion die aufgenommene Umgebung darstellen. Das so entstandene stereoskopische Panorama-Bild kann sofort im StereoViewer betrachtet werden.

4.7.1 Programmausführung

Listing 4.1 zeigt exemplarische Aufrufe von *panorama.py*. Mit dem Befehl der ersten Zeile würde aus den Bildern im Verzeichnis `panorama_creator/images/NAME/left` vollautomatisch ein 360°-Panorama aus den Einzelbildern der linken Augenperspektive erstellt und im Verzeichnis `panorama_creator/images/NAME` gespeichert werden. Die danach folgenden Programmaufrufe würden jeweils nur eine Teilaufgabe ausführen. Voraussetzung für das Funktionieren der Beispiele ist eine Verzeichnisstruktur wie in A.2.2.

Einfacher ist die Panorama-Erstellung mit dem Shell-Skript `create-all` im Verzeichnis `panorama_creator`. Es erwartet als Argument lediglich den Namen der Aufnahme-Session und erstellt automatisch je ein Vollpanorma für beide Augenperspektiven (siehe Listing 4.2).

4.7.2 Formatumwandlung

Für diesen Schritt wird das Linux-Programm *dcraw* verwendet. Dieses Programm ist in den meisten Linux-Distributionen enthalten. Es nimmt verschiedene Parameter[5] entgegen, welche die Konvertierung der RAW-Dateien beeinflussen (siehe Tabelle 4.1). Im Folgenden ein

[5] Für eine detaillierte Beschreibung siehe `http://www.cybercom.net/~dcoffin/dcraw/` (Abruf am 14.02.2009).

Beispiel für den Aufruf von *dcraw* zur Umwandlung einer Canon-RAW-Datei in das TIFF-Bildformat:

```
$ dcraw -c -w -k 1024 -H 5 -o 0 -T BILDNAME.CR2 > BILDNAME.tiff
```

Listing 4.3: Verwendung von dcraw

Parameter	Bedeutung
-c	Ausgabe des umgewandelten Bildes auf der Standardausgabe des Systems
-w	Verwendung des Kamera-Weißabgleiches wie er in den Meta-Daten der RAW-Datei gespeichert ist
-k	Dunkelheitsschwellenwert
-H	Behandlung von *Spitzlichtern*, 5 stellt einen Kompromiss dar, zwischen Weißüber- und unterbewertung
-o	Auswahl des Farbraums
-T	Erzeugung von Bildern im TIFF-Format

Tabelle 4.1: Kommandozeilenparameter für die RAW-Bild-Extraktion

4.7.3 Bildtransformation

Im zweiten Schritt werden die Bilder in eine equirektangulare Projektion überführt. Dazu wird das Programm *nona* verwendet. *Nona* ist Teil des *Hugin*-Programmpaketes und eigentlich ein Stitchingprogramm. Im Rahmen der normalen Benutzung kann es aber auch die für das Stitching umgerechneten Einzelbilder ausgeben. Durch diese Eigenschaft bietet es sich für die vorliegende Arbeit an, um die Projektions-Transformationen der einzelnen Fischaugen-Bilder vorzunehmen. Hinzu kommt, dass *nona* von der Kommandozeile bedienbar ist. Ein gängiger Aufruf von *nona* über die Kommandozeile sieht wie folgt aus.

```
$ nona -z NONE -r ldr -m TIFF_m -o ZIELDATEI -i 0 HUGIN_PROJEKTDATEI
```

Listing 4.4: Verwendung von nona

Die im Programmaufruf angegebene *Hugin*-Projektdatei wird zur Laufzeit des Skripts *panorma*.py aus den in Abschnitt A.4 erzeugten Templatedateien erstellt. *Nona* nimmt die Bildtransformation vor und speichert die resultierenden Bilder im Unterverzeichnis `rect` der aktuellen Panoramasession. Im Gegensatz zu den Original-Fischaugenbildern, deren Abbildung einen Bildwinkel von sowohl horizontal als auch vertikal 185° aufweisen, beträgt der abgebildete Bildwinkel bei den von *nona* generierten equirektangularen Bildern 180°. Das ist eine

Parameter	Bedeutung
-z	Kompressionsart
-r	Dynamikbereich, *ldr* behält den Originaldynamikbereich der Ausgangsdatei
-m	Format der Zieldatei
-o	Name der Zieldatei
-i	Index der in der angegebenen *Hugin*-Projektdatei aufgelisteten Bilddatei

Tabelle 4.2: Kommandozeilenparameter für die Bildtransformation

bewusste Vorgabe, die aus der in Abschnitt 4.6 beschriebenen Kalibrierung resultiert. Damit ist sichergestellt, dass die im nächsten Schritt erzeugten Panoramabilder den vertikalen Sichtbereich vollständig abbilden.

(a) **(b)**

Abbildung 4.5: Equirektangular transformierte Bilder

4.7.4 Mosaicing

Im letzten Schritt der Bildverarbeitung werden alle transformierten Bild-Dateien neu eingelesen. Aus jedem Bild wird ein Streifen von 1°-Breite extrahiert und dem zu erstellenden Panoramabild hinzugefügt. Die Breite des Bildstreifens in Pixeln ergibt sich dabei aus der Bildbreite des Einzelbildes, welches 180° Bildwinkel entspricht:

$$\text{Breite des Bildstreifens} = \text{Bildbreite}/180 \tag{4.1}$$

Bei der Extraktion der Streifen ist zu beachten, dass die Bildbreite ein ganzzahliges Vielfaches von 180 sein muss. Da Pixel ganzzahlig sind, wäre sonst eine Rundung notwendig, die zu Artefakten im resultierenden Panorama führen könnte. Um diese Bedingung zu gewährleisten wird die Bildgröße überprüft und bei Bedarf entsprechend angepasst.

Abbildung 4.6: equirectangulares Panoramabild für das linke Auge

4.7.5 Dauer

Die Gesamtdauer der Nachbearbeitung von den RAW-Dateien zu zwei fertigen Stereo-Panoramen hängt von der Rechnerausstattung und -auslastung ab. Bisher liegen nur Erfahrungswerte für den Betrieb auf einem Server mit 8 Intel Xeon Prozessoren (je 3 GHz) und 32 GB Arbeitsspeicher vor. Als Betriebssystem ist Ubuntu 8.10 installiert.

Bei der Nachbearbeitung der Bilder ist die Transformation von der Fischaugen- in eine Equirektangular-Abbildung der zeitaufwändigste Vorgang (siehe Tabelle 4.3). Die in der Tabelle angegebenen Zeiten beziehen sich auf die Abarbeitung des entsprechenden Schritts für beide Augenperspektiven.

Transformation	**Skript**	**Dauer**
Extraktion der RAW-Bilder	`extract-raw`	$\sim$ 16 min
Fisheye nach Equirektangular	`create-rect`	$\sim$ 24 min
Stitching	`create-panorama`	$\sim$ 4 min
Gesamtdauer		$\sim$ 44 min

Tabelle 4.3: Zeitaufwand für die Generierung eines Panoramabildes

4.7.6 Probleme

Das größte Problem, welches bei der Erstellung der Panoramabilder aufgefallen ist, ist die relativ schlechte Bildqualität. Die Bilder rauschen sehr stark. Die Ursache dafür ist der kleine Abbildungsmaßstab des Fischaugenobjektivs. Zudem verwendet es nur zwei Drittel, nämlich 15,5 mm, des 22,2 mm breiten Kamerachips (69,8%).

Da die genaue Funktionsweise der von *nona* durchgeführten Umrechnungen nicht genau geklärt werden konnte, ist nicht auszuschließen, dass den Bildern bei der Transformation von der Fischaugen- in die Equirektangular-Projektion ebenfalls Interpolationsfehler hinzugefügt werden.

Nach der Umrechnung durch *nona* haben die Bilder eine Größe von 3368 x 3368 Pixeln. Dies entspricht in Breite und Höhe 180°der vom Fischaugenobjektiv abgebildeten 185°(97,3%). Die genutzte Breite des Sensors fällt damit auf

$$15,5mm * 0,973 = 15,08mm$$

womit ein Grad also einer Breite von

$$15,08mm/180 = 0,84mm$$

auf dem Kamerachip entspricht. Multipliziert mit der Höhe des Chips von 14,8 mm ergibt sich eine effektiv genutzte Sensorfläche von

$$0,84mm * 14,8mm = 12,4mm^2.$$

Da die Gesamtfläche des Sensors

$$22,2mm * 14,8mm = 328,56mm^2$$

beträgt wird also ein 1° breiter Bildstreifen auf einer Fläche von

$$100/328,56mm^2 * 12,4mm^2 = 3,77\%$$

der Originalfläche abgebildet. Bei einer 12,5 Megapixel Auflösung der Canon EOS 450D entspricht diese Fläche einer Auflösung von

$$12,5\text{Megapixel} * 0,0377\% = 0,47\text{Megapixel}$$

bei einem Sichtbereich von 1° x 180°. Bezogen auf eine Kleinbildkamera mit geschätzten Bildwinkeln von etwa 60° horizontal und 40° vertikal ergibt sich eine resultierende Auflösung von

$$60 * 40/180 * 0,47 \text{ Megapixel} = 6,2\overline{6} \text{ Megapixel.}$$

Der Auflösungsverlust, der auf die Erstellungsmethode der Panoramabilder zurückzuführen ist, kann also auf etwa 50 % geschätzt werden.

Als zusätzliche mögliche Fehlerquelle käme die für die Bildstreifenextraktion notwendige Größenanpassung der equirektangularen Einzelbilder in Frage. Aufgrund des schon bei bloßer Betrachtung der Fischaugen-Ausgangsbilder zu sehenden Grundrauschens wird dieser Fehler aber nicht als ausschlaggebend angesehen.

5 StereoViewer

Dieses Kapitel widmet sich der Beschreibung der im Rahmen dieser Arbeit entwickelten Betrachtungssoftware. Mit dieser lassen sich stereoskopische equirektangulare 360° x 180°-Panoramabilder auf verschiedenen Ausgabegeräten darstellen. Die Konzeption und Realisierung eines Software-Projektes ist ein komplexer Prozess, in dessen Verlauf viele verschiedene Aspekte betrachtet, abgewogen, getestet, verworfen und festgehalten werden. Obwohl auf keinen verzichtet werden kann, können aus Platzgründen dennoch nicht alle in der ihnen gebührenden Tiefe besprochen werden. Die Ausführungen dieses Kapitels konzentrieren sich vor allem auf die grundlegenden Bestandteile der Software: *Rendering*, Perspektivenberechnung, Stereo-Formate, Netzwerkschnittstellen. Auf andere Teilbereiche, die hier keinen Platz mehr gefunden haben, wird an geeigneter Stelle hingewiesen. Der vollständige Quelltext der Software ist im SVN-Repository des Fachgebietes Audiokommunikation zu finden[1]. Alle Pfadangaben die in den folgenden Abschnitten verwendet werden, sind relativ zu dieser URL zu verstehen.

5.1 Vorüberlegungen

Die für diese Arbeit entwickelte Software *StereoViewer* ermöglicht vor allem die Betrachtung von stereoskopischen Vollbild-Panoramen. Zusätzlich gibt es auch die Möglichkeit, selbst erstellte 3D-Szenen stereoskopisch korrekt zu rendern.

Wie in Abschnitt 2.3 bereits besprochen, sollte der StereoViewer verschiedene Formate und Ausgabegeräte unterstützen. Das Programm sollte möglichst plattformunabhängig sein. Aus Effizienzgründen wurde die Entwicklung des StereoViewers vor allem unter Ubuntu Linux vorangetrieben. Regelmässige Tests der Software unter anderen Betriebssystemen sichern die generelle Portierbarkeit. Die Software sollte sich über die Kommandozeile starten lassen und Parameter entgegen nehmen, sowie während des Betriebs auf Anweisungen über ein Netzwerk reagieren.

Als Programmiersprache wurde die Python-Skriptsprache in der Version 2.5 verwendet. Diese ist plattformübergreifend einsetzbar und bei den meisten Betriebssystemdistributionen be-

[1] `https://srv2.ak.tu-berlin.de/demos/stereo_viewer`

Rendermode	Vollpanoramen, 3D-Szenenbeschreibungen (3ds-Format[8])
Stereo-Formate	Anaglyph, Zeilen-Interlaced, DLP3D, Frame-Sequentiell, Dual
Geräte	eMagin Z800 3D Visor[9], DLP-3D-Ready-Fernseher[10]
Schnittstellen	Kommandozeile, OSC
Programmiersprache	Python 2.5
Betriebssysteme	Ubuntu Linux (Windows XP, Mac OS X)
Grafik-Bibliothek	OpenGL

Tabelle 5.1: Zusammenfassung der Vorüberlegungen

reits enthalten. Python gehört zu den sogenannten interpretierten Sprache[2]. Die Sprache steht in dem Ruf leicht erlernbar zu sein und über eine "pseudocode-artige Syntax"(Sanner 1999, S. 2) zu verfügen, die sich sowohl intuitiv lesen als auch schreiben lässt[3]. Sie ist aufgrund ihrer Struktur sehr gut für langfristige und hochgradig modularisierte Software-Projekte geeignet. Außerdem lässt sich Python leicht durch eigene C oder C++ Module erweiterne[4]. Dadurch gibt es mittlerweile eine große Anzahl an zusätzlichen Modulen, die eigentlich Wrapper um hardwarenahe C-Bibliotheken sind und eine gute Performance aufweisen.

Als 3D-Grafikumgebung wurde *OpenGL*[5] gewählt. Es überzeugte für die vorliegende Arbeit durch eine hohe Portabilität, seinen Status als Defacto-Standard für die plattformübergreifende Entwicklung von 3D-Anwendungen, die große Verfügbarkeit von Dokumentationsmaterialien und die einfache Einbindung in verschiedenste Programmiersprachen, u.a. auch Python[6]. Mit *GLUT*[7] steht außerdem ein einfach zu verwendendes Toolkit zur Verfügung, dass z.B. über einen plattformunabhängigen Fenster-Manager verfügt.

In Tabelle 5.1 sind die resultierenden Rahmenbedingungen noch einmal zusammengefasst.

5.2 Teilprobleme

Die Aufgaben des StereoViewers lassen sich aufgrund des Anforderungsprofils folgendermaßen gliedern:

1. Entgegennahme von Kommandozeilenparametern oder XML-basierten Konfigurationseinstellungen, z.B. Szenen- oder Panorama-Dateien, Stereo-Modus, Bildschirm-Auflösung

[2] Im Gegensatz z.B. zu C++ oder Java, vgl. auch Henning und Vogelsang (2007).
[3] Diese Aussage ist dennoch stark von Größe und Komplexität des umgesetzten Systems abhängig und bedeutet nicht, dass ein Laie die Syntax problemlos versteht.
[4] Rossum 1993.
[5] `http://www.opengl.org`
[6] Vgl. PyOpenGL `http://pyopengl.sourceforge.net/`.
[7] OpenGL Utility Toolkit: `http://www.opengl.org/resources/libraries/glut/`

2. Einbindung externer Geräte, z.B. des eMagin Z800 HMD, der gesondert angesteuert werden muss um den geräteeigenen Stereo-Modus zu aktivieren
3. Erzeugen einer darstellbaren 3D-Szene
4. Zeichnen der Szene auf den Bildschirm in Abhängigkeit vom benötigten Stereoformat
5. Kommunikation mit einer Netzwerkressource, um den Programmablauf zu steuern, eine neue 3D Szene nachzuladen, oder den Stereo-Modus zu wechseln

Aus diesen Teilproblemen lässt sich bereits eine Struktur für die Software ableiten. So erscheint es einleuchtend, dass die Erzeugung einer 3D-Szene unabhängig von ihrer Darstellung erfolgen kann. Durch die Verwendung von Display-Listen[11] kann die zu rendernde OpenGL-Szene einmal konstruiert und im Speicher abgelegt werden, um sie dann wenn nötig auf den Bildschirm zu zeichnen. Die Konstruktion der benötigten Display-List kann einer eigenen *Display-List*-Klasse übertragen werden.

Das Rendern der Display-Listen kann mit einer weiteren spezialisierten Klasse von Objekten erfolgen, die im Rahmen dieses Projektes als *Display-Handler* bezeichnet werden. Jedem *Display-Handler* ist es dabei selbst überlassen, wie die vorberechnete Szene gerendert wird. Mit diesem Ansatz lassen sich die verschiedenen Stereo-Formate unabhängig von der zugrunde liegenden Szene realisieren.

Um gerätespezifische Erweiterungen oder die Kommunikation über ein Netzwerk zu ermöglichen, könnten die dafür benötigten Funktionalitäten direkt in der Software verankert werden. Dies würde auf lange Sicht wahrscheinlich Nachteile hinsichtlich der Wartungsmöglichkeiten und der Erweiterungsfähigkeit mit sich bringen. Gerade letzteres aber ist eine der grundlegenden Anforderungen an die StereoViewer-Software. Als Alternative bietet sich ein Plugin-System an[12].

5.3 Aufbau

Dieser Abschnitt widmet sich der Beschreibung der wichtigsten Komponenten des Stereo-Viewers. Die Trennung der einzelnen Komponenten ist bereits an der Verzeichnisstruktur klar erkennbar[13]. Alle Komponenten sind als separate Python-Pakete realisiert und liegen in ihrem eigenen Verzeichnis, in dem sich auch eine Datei mit dem Namen `__init__.py` befinden muss. Bei der Einbindung des jeweiligen Pakets in die Haupt-Applikation, in Python-Terminologie wird von `import` gesprochen, wird diese Datei eingelesen und initialisiert das

[11] Siehe Abschnitt 3.3.6.
[12] Vgl. Voelter 2003, S. 4.
[13] Siehe Abschnitt A.2.1.

```
self.displayHandler = DisplayHandler.DefaultHandler(self)
self.displayList    = DisplayLists.CustomList(self, self.file)
self.pluginManager  = Plugins.PluginManager(self, HOOKS)
```

Listing 5.1: Beispiele für Komponenten-Instanziierung in der Applikationsklasse

```
self.displayList = DisplayLists.PanoramaList(self, 90, 180, 180, self.file)
```

Listing 5.2: Instanziierung von PanoramaList

Paket. Je nach Paket verläuft die Initialisierung unterschiedlich. Meist werden aber nur benötigte Python-Module geladen.

Da der StereoViewer durchgehend objekt-orientiert konzipiert wurde, handelt es sich bei allen Komponenten um Klassen. Die Einbindung erfolgt immer über eine Instanziierung in eine Klassen-Variable der Hauptapplikation (siehe Listing 5.1). Über diese Klassenvariablen erfolgt die Steuerung der Komponenten.

5.3.1 Display-Listen

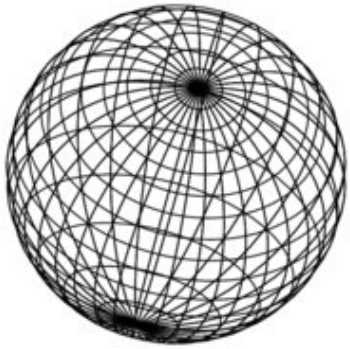
Dieses Paket ist das am einfachsten strukturierte. Beim `import` werden lediglich notwendige Python-System-Module, sowie die zum StereoViewer gehörenden DisplayList-Module `CustomList.py` und `PanoramaList.py` importiert.

Beide Klassen werden von der Basis-Klasse `BaseList`[14] abgeleitet und erben die Klassenmethode `getList`. Der Rückgabewert dieser Methode identifiziert die generierte Display-Liste und wird von der Haupt-Applikation an den gerade zuständigen Display-Handler weitergereicht.

PanoramaList Aufgabe dieser Komponente ist das Laden der stereoskopischen Vollbild-Panoramen, das Erstellen einer Sphäre (siehe Abbildung 5.1a) und die abwechselnde Projektion je eines von zwei zusammengehörenden Vollbild-Panoramen auf die Innenseite der Sphäre. In Listing 5.2 ist ersichtlich, wie ein Instanziierungsaufruf aussieht. Die numerischen Parameter geben in der Reihenfolge des Auftretens den Durchmesser und die Anzahl an Längen- und Höhengraden an[15]. Der letzte Parameter ist der Name der zu verwendenden Textur. Eine Beispieltextur ist in Abbildung 4.6 am Ende von Kapitel 4 zu sehen.
Eine weitere Funktion der PanoramaList-Klasse ist das Zeichnen eines Begrenzungsrahmens in das verwendete Panoramabild (siehe Abbildung 5.1b). Ob ein Rahmen gezeichnet werden

[14] Definiert in `DisplayList/__init__.py`.
[15] Je mehr Längen- und Höhengrade die generierte Sphäre hat, desto kugelähnlicher wird ihre Gestalt.

soll, und wenn ja, in welcher Breite, Höhe, Rahmendicke und Transparenz, kann in der in Abschnitt A.9 besprochenen XML-Datei definiert werden.



(a) Sphäre

(b) Panoramabild mit eingezeichneter Sichtbegrenzung

Abbildung 5.1: Sphäre und Panoramabild mit Sichtbegrenzung

CustomList Diese Komponente erstellt eine OpenGL-Szene auf Grundlage einer 3D-Szenenbeschreibung aus einer externen Datei. In der zum Zeitpunkt des Schreibens dieser Arbeit aktuellen Version des StereoViewer werden nur 3ds-Dateien unterstützt, und auch diese nur eingeschränkt. Es ist nicht möglich beliebig komplexe 3D-Szenen zu laden. Da das Hauptaugenmerk der Entwicklung auf den stereoskopischen Panoramabildern lag, wurde bisher nur sichergestellt, dass es prinzipiell möglich ist externe Formate zu laden und zu rendern.

Für den Import von Dateien im 3ds-Format wird das Python-Paket `Dice3ds`[16] benötigt, welches frei erhältlich, aber nicht Teil der Python-Standardbibliothek ist. Sobald CustomList wie in Zeile 2 von Listing 5.1 instanziiert wurde, wird die im Aufruf übergebene Szenen-Beschreibungsdatei geladen und mit `Dice3ds` in eine Display-Liste übersetzt (siehe Abbildung A.8).

5.3.2 Display-Handler

Aufgabe dieses Paketes ist das Rendern einer zuvor erstellten OpenGL-Szene auf den Bildschirm. Beim `import` dieses Paketes werden zunächst die benötigten Python-Module geladen. Dazu zählen die OpenGL-Abhängigkeiten und einige Python Standard-Module.

Während der Initialisierung des StereoViewer in `stereo_viewer/Application.py` wird die von `DisplayHandler` bereitgestellte Funktion `DisplayHandler.findDisplayHandlers()` aufgerufen. Diese sucht im Verzeichnis `stereo_viewer/DisplayHandler` nach verfügbaren

[16] `http://www.aerojockey.com/software/dice3ds/index.html` (Abruf am 09.03.2009)

Display-Handlern und importiert diese. So lassen sich mit wenigen Änderungen an zentralen Stellen im Quellcode neue Display-Handler hinzufügen.

In der paketeigenen Datei `__init__.py` wird neben der in Abschnitt A.6 beschriebenen einfachsten monoskopischen Display-Handler-Klasse auch die Klasse `BaseHandler` definiert. Dies ist die Display-Handler Basis-Klasse von der alle Display-Handler abgeleitet sein müssen. Da die hier definierten Methoden von zentraler Bedeutung für die Funktionsweise aller Display-Handler sind, sollen die wichtigsten hier etwas detaillierter erläutert werden.

start / leave: Diese beiden Methoden werden bei Aktivierung bzw. Deaktivierung eines Display-Handlers aufgerufen. Sie dienen zur Initiliasierung eines benötigten Rendering-Kontextes, bzw. zum Wiederherstellen des Kontextes vor der Initialisierung. Display-Handler können diese Methoden, die in der Basis-Klasse leer sind, überschreiben um benötigte Änderungen vorzunehmen. In der aktuellen Version wird die Methode `start` nur vom Display-Handler Dual verwendet, da dieser einen im Vergleich zu den anderen Display-Handler-Klassen doppelt so breiten Anzeigebereich benötigt.

setFrustum: In dieser Methode werden die notwendigen Parameter für ein perspektivisch korrektes Sichtvolumen berechnet. Wenn der Viewer eine 3D-Szenenbeschreibung rendert und eine der stereoskopischen Display-Handler-Klassen aktiviert ist (also nicht die Default-Display-Handler-Klasse) ist diese Methode auch sensibel für die jeweilige Augenperspektive. Dadurch wird im genannten Fall ein asymmetrisches Sichtvolumen konstruiert[17] statt eines symmetrischen, wie es für monoskopische Darstellungen und für die Panoramabetrachtung der Fall ist.

setLookAt: Diese Methode berechnet den korrekten Betrachtungspunkt. Wie `setFrustum` ist diese Methode unter den genannten Bedingungen sensibel für die jeweilige Augenperspektive.

renderList: Mit dem Aufruf dieser Methode wird das Rendern der gespeicherten Display-Liste gestartet. Sichtvolumen und Betrachtungspunkt werden durch Aufrufe der Methoden `setFrustum` und `setLookAt` festegelegt. Die 3D-Szene wird entsprechend den aktuellen Rotations- und Neigeparametern so gedreht bzw. geneigt, dass der Betrachter genau den Ausschnitt zu sehen bekommt, den er sehen soll. Abschließend wird die im Speicher vorrätige aktuelle Display-Liste durch den Aufruf der OpenGL-Funktion `glCallList(displayList)` auf den Bildschirm gezeichnet. Jede von BaseHandler abgeleitete Klasse sollte diese Methode während der Ausführung ihrer eigenen `render`-Methode aufrufen (siehe Listing A.6 als Beispiel).

draw: Bei dieser Methode handelt es sich um einen sogenannten Dispatcher. Um ein Frame zu rendern, ruft die Haupt-Applikation diese Methode auf. Abhängig vom aktuellen

[17] Siehe Abschnitt 3.6.4.

`drawMode` (Panorama oder Custom) wird dann eine der beiden Methoden `drawPanorama` oder `drawScene` aufgerufen. Diese Methode braucht von einer Display-Handler-Klasse nicht überschrieben zu werden.

render2DText: Diese Methode stellt 2D-Text im aktuellen OpenGL-Fenster dar. Sie wird in der aktuellen Version des StereoViewers von den Plugins *Statstik* und *KeyControl* verwendet, um die Bildwiederholfrequenz und aktive Tastenbelegungen für die Tastatursteuerung auszugeben.

In Abschnitt A.6 ist exemplarisch eine ebenfalls zum StereoViewer gehörende monoskopische Display-Handler-Klasse inklusive Quelltext aufgeführt. In der aktuellen Version gibt es fünf stereoskopische Display-Handler, welche die fünf in Abschnitt 3.2.5 besprochenen Bildtrennungsverfahren umsetzen. Auf eine detaillierte Beschreibung der einzelnen Klassen wird an dieser Stelle verzichtet und auf den dokumentierten Quelltext verwiesen.

5.3.3 Plugins

Die Hauptfunktion des StereoViewers ist die Darstellung stereoskopischer Panoramabilder oder 3D-Szenen. Alle Aufgaben, die nicht unmittelbar zu dieser Kernfunktionalität gehören, wurden in Plugins ausgelagert. Damit wird zum einen der Code übersichtlich strukturiert, zum anderen ist es so möglich zusätzliche Funktionen einzubauen, ohne grundlegende Funktionalitäten zu beschädigen. Alle Plugins sind als eigene Klassen in Python realisiert.

Das Plugins-Paket definiert in der paketeigenen `__init.py__`-Datei zwei Klassen: `BasePlugin` und `PluginManager`. Soll eine Plugin-Klasse von der Anwendung erkannt werden, so muss sie von der Klasse `BasePlugin` abgeleitet sein. Die Klasse `PluginManager` bildet die Verbindung zwischen StereoViewer und Plugin.

Sobald der Plugin-Manager in `stereo_viewer/Application.py` entsprechend Zeile 3 in Listing 5.1 instanziiert wird, werden im Verzeichnis `stereo_viewer/Plugins` vorhandene Plugin-Dateien gesucht. Diese werden eingebunden und in einem internen Register vermerkt.

Die Applikation selbst definiert sogenannte *Hooks*. Das sind Stellen im Programm, an denen sich Plugins, die über den Plugin-Manager registriert wurden, in den Programmablauf 'einhängen' können. Welche Hooks definiert sind, kann in der Datei `stereo_viewer/Data/Definitions.py` eingesehen und geändert werden. Tabelle 5.2 gibt eine Übersicht über alle derzeit verfügbaren Plugins, ihre Funktion, sowie die von ihnen verwendeten Hooks.

Plugin-Einstellungen können über die StereoViewer-Konfigurationsdatei vorgenommen werden. Es liegt in der Verantwortung des Plugins, korrekt auf die global verfügbare Konfigurationsvariable zuzugreifen. Plugin-Optionen sollten in der XML-Datei in einem eigenen Knoten

Plugin	Funktion	verwendete Hooks
KeyControl	erweiterte Tastatursteuerung zum Ändern des Stereomodus und anderer Parameter	display, keyboard
OSCControl	OSC-Schnittstelle, z.B. für Trackerdaten	options, init, close
Z800	Ansteuerung des eMagin Z800 HMD um diesen im Stereo-Modus verwenden zu können	init, setDisplayHandler, idle, keyboard, close
Statistic	Ausgabe der Bildwiederholfrequenz	display, keyboard, close
Capture	Bildschirmfotos speichern	options, keyboard

Tabelle 5.2: Plugins

abgelegt werden[18].

OSCControl Dieses Plugin implementiert die OSC-Schnittstelle, welche z.B. für die Kommunikation im Rahmen des wonder-Projekts benötigt wird. Es basiert auf dem Python-Modul *pyOSC*. Die verfügbaren Schnittstellen ermöglichen beispielsweise das Empfangen von Head-Traker-Daten, das Laden von Bildern oder Szenen, sowie die Änderung einiger Rendering-Optionen (siehe Tabelle A.2). In der XML-Konfigurationsdatei können die für die OSC-Kommunikation zu verwendende Netzwerk-Adresse und -Port angegeben werden.

1. den Wechsel des Stereo-Modus (Display-Handler)
2. das Wechseln zwischen stereoskopischem und monoskopischem Rendern
3. Laden

KeyControl Als zusätzliche Interaktionsmöglichkeit stehen während der Ausführung des StereoViewer auch eine Reihe von Tastaturkommandos zur Verfügung. Die genaue Übersicht findet sich in Tabelle A.3. Die Tastatursteuerung ermöglicht:

1. den Wechsel des Stereo-Modus (Display-Handler)
2. das Wechseln zwischen stereoskopischem und monoskopischem Rendern
3. die Veränderung der Entfernung zur Fokusebene, und damit den Konvergenzwinkel
4. die Veränderung der Stereobasis (nur im Custom-Modus)
5. die Veränderung des dargestellten Bildwinkels

[18] Siehe Abschnitt A.13.

Z800 Dieses Plugin prüft während der Initialisierung des StereoViewer, ob das eMagin Z800 HMD an das Computersystem angeschlossen ist. Falls ja, kümmert es sich um die USB-Kommunikation und schaltet das Gerät bei Aktivierung des Frame-Sequential-Modus automatisch auf den Stereo-Modus. Dieses Plugin wurde benötigt, weil für das HMD keine Linux-Treiber verfügbar sind. Für die Entwicklung konnte auf die USB-Speczfikation des Herstellers zurückgegriffen werden, welche frei zur Verfügung steht[19].

Capture Mit dem Capture-Plugin lassen sich aus dem laufenden Programm heraus Bildschirmfotos des OpenGL-Fensters anfertigen. Dies wurde als sinnvoll betrachtet, da der Fenstermodus in dem die das Programm ausgeführt wird betriebsystemeigene Steuerelemente deaktiviert und somit nicht auf die entsprechenden Funktionalitäten des Betriebssystems zurückgegriffen werden kann.

Statistic Um einen ersten approximativen Eindruck der Leistungsfähigkeit des StereoViewers zu erhalten wurde eine einfache Kalkulation der Bildwiederholrate (*Frames per second*) implementiert.

5.4 Einstellungen zur Laufzeit

Zur Laufzeit sind einige Anpassungen an der stereoskopische Darstellung möglich. Diese sind über Tastaturkommandos[20] zugänglich und werden hier kurz erläutert. Soweit nicht anders angegeben, stehen die Optionen sowohl im Panorama-Modus als auch im Custom-Modus (3ds-Szene) zur Verfügung.

5.4.1 Fokus

Um die Entfernung zur Nullparallaxe einzustellen, können linkes und rechtes Panoramabild entsprechend Abschnitt 3.6.3 gegeneinander verschoben werden. Damit ändert sich der Abstand zur Fokusebene, was eine notwendige Änderung der Augenkonvergenz erfordert. Es sollte darauf geachtet werden, dass eine Einstellung gewählt wird, die den Betrachter nicht anstrengt.

[19] `http://www.3dvisor.com/downloads/software/EMA_SDK_2_2.zip` (Abruf am 07.03.2009)
[20] Siehe Abschnitt A.3.

5.4.2 Stereobasis

Da die Stereobasis des Aufnahmesystems, mit dem die Panoramabilder erstellt wurden, nachträglich nicht mehr veränderbar ist, steht diese Option nur im Custom-Modus zur Verfügung. Sie erlaubt die Vergrößerung bzw. Verkleinerung der Stereobasis, wodurch sich der Tiefeneindruck verändert[21].

5.4.3 Bildwinkel

Die Größe des dargestellten Ausschnitts aus dem Panoramabild kann mit dem Bildwinkel kontrolliert werden. Größere Bildwinkel führen zu einer stärkeren Wahrnehmung der durch die sphärische Projektion verursachten Verzerrungen. Besonders deutlich wird dies bei Bewegungen.

5.5 Abhängigkeiten

Der StereoViewer ist von einigen Bibliotheken und Software-Paketen abhängig, die auf den meisten Systemen standardmässig nicht installiert sind. Alle sind frei im Internet erhältlich und sind unter Open-Source-Lizenzen veröffentlicht. Tabelle 5.3 gibt eine Übersicht der Pakete und ihrer Bezugsquellen.

Software	**Beschreibung**	**Bezugsquelle**
PIL	Python Image Library, Bibliothek für Bildverarbeitung	`http://www.pythonware.com/products/pil/`
PyOpenGL	Python-Bindung für OpenGL	`http://pyopengl.sourceforge.net/`
Dice3DS	Python-Paket für die Verarbeitung von 3ds-Dateien	`http://www.aerojockey.com/software/dice3ds/index.html`
NumPy	Python-Paket für wissenschaftliche Berechnungen	`http://numpy.scipy.org/`
PyOSC	Python Implementation des OSC-Protokolls	`https://trac.v2.nl/wiki/pyOSC`

Tabelle 5.3: Abhängigkeiten

Die benötigten Pakete wurden außerdem heruntergeladen und stehen im SVN-Repository im Verzeichnis `Dependencies` als nachweislich funktionierende Versionen dauerhaft zur Verfügung.

[21] Siehe Abschnitt 3.2.3.

5.6 Systemvoraussetzungen

Die entwickelte Software stellt moderate Anforderungen an die verwendete Hardware. Zwingend notwendig ist in jedem Fall eine Graphikkarte mit OpenGL-Unterstützung. Wenn dies gegeben ist, läuft der StereoViewer plattformübergreifend unter der Voraussetzung, dass die notwendigen Abhängigkeiten installiert sind. Tabelle 5.4 gibt einen Überblick über die Testsysteme die während der Entwicklung zur Verfügung standen. Testsystem 3 ist aus Gründen der Vollständigkeit aufgeführt. Aufgrund der schlechten Rechenleistung und der begrenzten Ressourcen eignet sich dieses System für den praktischen Gebrauch nicht.

	Testsystem 1	**Testsystem 2**	**Testsystem 3**
Art	Desktop PC	Desktop PC	Laptop
Betriebsystem	Ubuntu 8.10	Ubuntu 8.10	Mac OS 10.4
Prozessor	AMD Athlon 64 3500+	Intel Quad-Core i7 920	Power PC G4
Prozessortakt	2,2 GHz	2,67 GHz	1 GHz
Arbeitsspeicher	1 GB	3 GB	769 MB
Grafikkarte	GeForce 7600 GS	GeForce 9800 GTX+	ATI Mobility Radeon 9200
Grafikspeicher	512 MB	512 MB	32 MB

Tabelle 5.4: Testsysteme

5.7 Performance

Um ein ungefähres Maß für die Performance abschätzen zu können, wurde für mehrere Bildschirmauflösungen die Bildwiederholfrequenz in allen Stereo-Modi und, als Referenzwert, im monoskopischen Modus ermittelt. Die Daten wurden mit den Testsystemen 1 und 2 aus Tabelle 5.4 erhoben. Bei allen Messungen wurde über einen Zeitraum von 20 Sekunden gemittelt. Die Berechnung der Bildwiederholfrequenz beruht auf den Time-Funktionen von GLUT. Die Genauigkeit ist nicht garantiert, weshalb stark gerundet wurde. Während der Messung wurden willkürlich schnelle und langsame Kameraschwenks durchgeführt.

Die angegebene Bildwiederholfrequenz spiegelt nicht die reale Geschwindigkeit des Bildaufbaus wieder, sondern gibt ein Indiz dafür, wie oft in einem bestimmten Zeitintervall (hier 1 s) das OpenGL-System bzw. GLUT den Display-Callback aufgerufen hat und in der Lage ist Bilder zu rendern. Ob diese wirklich dargestellt werden, bleibt der Grafikkarten überlassen. Der genaue Ablauf der Synchronisierung von GLUT mit der Grafikkarte ist nicht bekannt. Es kann vermutete werden, dass GLUT Bilder rendert und in den Buffer der Grafikkarte

Stereo-Modus	Auflösung des OpenGL-Fensters			
	800x600	1280x1024	1600x1024	1920x1080
Anaglyph	120	40	35	33
Interlaced	240	95	70	85
DLP3D	220	85	65	75
Frame-Sequentiell	375	137	95	120
Dual	165	60	44	63
Default	375	135	93	120
Bildschirmauflösung	1280x1024	1920x1080		

Tabelle 5.5: Testsystem 1: maximale Bildwiederholfrequenz in Abhängigkeit von Stereo-Modus und Bildschirmauflösung, Angabe in Frames pro Sekunde (FPS)

Stereo-Modus	Auflösung des OpenGL-Fensters			
	800x600	1280x1024	1600x1024	1920x1080
Anaglyph	500	320	290	260
Interlaced	800	500	480	600
DLP3D	900	480	470	480
Frame-Sequentiell	1100	800	630	950
Dual	750	210	14	12
Default	1500	1150	1000	980
Bildschirmauflösung	1280x1024	1920x1080		

Tabelle 5.6: Testsystem 2: maximale Bildwiederholfrequenz in Abhängigkeit von Stereo-Modus und Bildschirmauflösung, Angabe in Frames pro Sekunde (FPS)

schreibt, diese Daten aber nur zu festen Taktzeiten von der Grafikkarte an den Monitor gesendet werden. Dennoch lässt sich mit den erhobenen Daten die Systemleistung schätzen.

Wie erwartet nimmt die maximale Bildwiederholfrequenz mit zunehmender Größe des OpenGL-Fensters ab. Außerdem ist die höhere Bildwiederholfrequenz in der Auflösung 1920x1080 ein Indiz dafür, dass die Leistung besser ist, wenn die Größe des OpenGL-Fensters der Bildschirmauflösung entspricht. Unbekannt ist, aus welchem Grund die Bildwiederholfrequenz im Dual-Modus auf Testsystem 2 für die höheren Auflösungen so stark sinkt.

5.8 Latenzabschätzung

Bei der Verwendung eines Trackers für die Bestimmung der Kopforientierung kann es zu Latenzen beim Bildaufbau kommen. Dies resultiert einerseits daraus, dass bereits der Tra-

cker selbst eine Latenz aufweist. Darüber hinaus erfolgt die Datenübermittlung über das OSC-Protokoll, welches mit einer zusätzlichen netzwerkabhängigen Latenz behaftet ist. Diese muss im Bedarfsfall jedes mal neu ermittelt werden. Eine Möglichkeit der Messung bestünde beispielsweise darin, auf verschiedenen Computern im Netz je einen OSC-Client und einen OSC-Server zu implementieren[22]. Wenn der Client als Datenpaket den Zeitstempel (eng. Timestamp) des Abschickens versendet, sollte der Server in der Lage sein, bei Erhalt des Datenpaketes die Differenz zur aktuellen Zeit zu bilden und daraus eine Abschätzung der Netzwerklatenz zu ermitteln.

	maximal erwartbare Latenz
Tracker	8,33 ms
Stereoviewer	16,67 ms
Gesamt	35 ms

Tabelle 5.7: Latenzabschätzung

Die Latenz des Trackers lässt sich aus der Updaterate des Trackers schätzen. Diese beträgt 120 Hz, also 1/120-Sekunden, was einen maximalen zeitlichen Versatz zwischen Positionsänderung und Meldung derselben von

$$1/120 * 1000 = 8,3\overline{3}ms$$

ergibt. Dieser Wert wurde durch eigene Messungen bestätigt. Da der Tracker über einen Seriell-zu-USB-Adapter an den Computer angeschlossen wird, könnte hier eine weitere Latenzquelle liegen. Diese ist in der Messung bereits enthalten.
Die mögliche maximale Latenz des StereoViewer ergibt sich aus der Bildwiederholrate von 60 Hz im Stereobetrieb[23]. Eingehende OSC-Anweisungen, beispielsweise eine Positionsänderung des Head-Trackers, werden sofort im nächsten Renderingzyklus umgesetzt. Damit ergibt sich eine maximale Latenz von

$$1/60 * 1000 = 16,6\overline{6}ms$$

Es bleibt zu prüfen ob dieser Wert während eines Belastungstest wirklich eingehalten werden kann, da die allgemeine Systemauslastung ebenfalls einen Einfluss hat.

[22] Entgegen der Erwartungen ist es bei OSC der Client, der Daten sendet während der Server diese empfängt.
[23] Siehe Abschnitt A.10.

5.9 Probleme

Eines der größten Probleme für die praktische Anwendung ergibt sich in Bezug auf das eMagin Z800 HMD. Dieses hat nur einen VGA-Eingang an dem es Bilder mit einer Taktfrequenz von 60 Hz im Frame-Sequentiell-Format erwartet. Alternierend wird je ein Frame auf eines der beiden OLED-Displays[24] abgebildet. Das erste Frame wird auf dem Display für das linke Auge dargestellt. Bei Verwendung mit der StereoViewer-Software kann nicht garantiert werden, dass das erste Frame, welches beim HMD ankommt, auch tatsächlich das Frame für das linke Auge ist. Daher muss vor der Verwendung immer überprüft werden, ob die Bilder eventuell vertauscht sind. Falls dies der Fall ist, können rechtes und linkes Bild per Tastatursteuerung oder OSC-Befehl korrigiert werden[25].

Als problematisch zu bewerten ist auch die Tatsache, dass nicht klar ist, was mit den gerenderten Bildern geschieht, sobald sie von der Grafikkarte an die Ausgabegeräte gesendet werden. So waren auf dem Samsung DLP-Fernseher bei einigen nicht nativen Bildschirmauflösungen Artefakte in Form von weißflächigen Doppelbildern zu sehen. Dies lässt sich eventuell auf eine durch Kompression fehlerbehaftete Kanaltrennung zurückführen.

Ein grundsätzliches Problem besteht darin, dass die Entfernung zur Fokusebene, und damit der Konvergenzwinkel der Augen, während der Programmausführung statisch ist. Wenn ein Betrachter den Blick auf Objekte richtet, die näher an der virtuellen Kameraposition liegen als die Fokusebene, erfordert deren Betrachtung eine höhere Konvergenz. Wünschenswert für eine lange und angenehme Betrachtung einer Szene wäre eine automatische Einstellung der Entfernung zur Fokusebene in Abhängigkeit von der Blickrichtung des Betrachters (siehe Kapitel 6).

Gerätespezifisch, aber mit deutlichen Folgen für das resultierte Stereo-Erlebnis, zeigt sich der beschränkte Bildbereich als Problem. Das eMagin HMD hat nur sehr kleine Displays (800x600) und ist an den Seiten nicht geschlossen. Dadurch fällt es subjektiv empfunden schwer, sich wirklich auf die dargestellte Szene einzulassen. Der DLP-Fernseher hat einen größeren Bildbereich und eine höhere Auflösung. Doch auch dieses Gerät kann nicht annähernd das von Bourke (2006, S.1) empfohlene, bis zu 160° (horizontal) umfassende periphere Sichtfeld, nachbilden.

Ein weiteres Problem ergibt sich aufgrund der recht umständlichen Konfiguration der Grafikkarte. Für feste Versuchsaufbauten hat das weniger Relevanz, da die Einstellungen lediglich einmal gemacht werden müssen. Für den breiten Einsatz als stereoskopische Betrachtungssoftware zur Verwendung mit verschiedenen Systemen wäre es jedoch wünschenswert, wenn

[24] OLED = Organic Light Emitting Display
[25] Siehe Abschnitt A.3. und Abschnitt A.2

die Software selbst in der Lage wäre, die notwendigen Einstellungen während der Initialisierung vorzunehmen.

6 Zusammenfassung und Ausblick

Es wurde gezeigt, wie sich mit einem Minimum an Material und Kosten ein System erstellen lässt, mit dem es möglich ist, 360° x 180° Stereo-Panoramen zu erstellen. Die Aufnahmestrecke kommt dabei mit weitestgehend gebräuchlicher Technik aus. Die Aufbereitung der großen Anzahl von Einzelbildern erfolgt vollautomatisch ohne menschliches Eingreifen in einem überschaubaren Zeitrahmen. Die erzeugten Stereo-Panoramen können mit dem ebenfalls im Rahmen dieser Arbeit erstellten StereoViewer betrachtet werden. Dabei werden viele gängige Formate zur Darstellung von Stereo-Inhalten unterstützt, so dass das hier beschriebene System in einer Vielzahl verschiedener Situtationen eingesetzt werden kann. Zugleich ist es in einer Weise modularisiert, die Erweiterungen jederzeit möglich macht.

Es bieten sich viele Richtungen an, in denen die Methodik und Durchführung der Panorama-Erstellung und der StereoViewer weiterentwickelt werden können.

Als Erstes wäre eine Evaluation des aktuellen Systems notwendig, da eine Weiterentwicklung nur lohnenswert scheint, sofern die zusätzliche Visualisierung die Plausibilität der binauralen Simulation wirklich erhöht.

Bezüglich der Bildqualität wäre es sinnvoll ein anderes Kamera-Objektiv-System auszuprobieren. Eine Erhöhung der effektiven Auflösung bei Beibehaltung der durch die Fischaugenlinse möglichen hohen Schärfentiefe und Bildwinkel würde den wahrgenommenen Raumeindruck und damit die Plausibilität der visuellen Szene wahrscheinlich deutlich steigern können.

Eine interessante Ergänzung des bestehenden Darstellungssystems wäre eine automatische Echtzeit-Disparitätskontrolle. Pritch, Ben-Ezra und Peleg (2000) stellen ein System vor, mit dem sich während des Mosaicing-Vorgangs die Disparitäten einzelner Bildabschnitte bestimmen lassen. Auf den Grundlagen dieser Werte verändern sie während des Zusammensetzens der Bildstreifen die disparitätssteuernden Parameter. Dadurch sind sie in der Lage, zu nah am Aufnahmestandort befindliche Objekte im Bild derart zu modifizieren, dass es nicht zu Störungen des Raumeindrucks durch ungewollte Doppelbilder kommt. Für Details siehe Pritch, Ben-Ezra und Peleg (2000, S. 4).
Als erster Ansatz einer Echtzeit-Disparitätskontrolle in Abhängigkeit der Blickrichtung des Betrachters[1], könnte durch eine Disparitätsanalyse der stereoskopischen Panorama-Bildpaare

[1] Dies ist nur möglich, wenn die Kopfbewegungen des Betrachters mit einem Head-Tracker erfasst werden.

eine Art Höhenkarte erstellt werden. Diese bildet für alle Bildbereiche die erfasste Querdisparation ab. Unter der Annahme, das der Betrachter genau in die Mitte des dargestellten Bildes blickt, könnte man die Entfernung zur Fokusebene während der Laufzeit auf den betrachteten Bereich anpassen. Das hätte unter idealen Bedingungen eine Wiederangleichung von Akkommodation und Konvergenz zur Folge. Durch die Anpassung des Konvergenzwinkels (über die Entfernung zur Fokusebene) würde diese in die Bildschirmebene verlagert. Da die Akkommodation immer nur auf die Bildschirmebene geschehen kann, wären diese beiden wichtigen Informationen für Tiefenwahrnehmung wieder stärker gekoppelt.
Eine technisch aufwendigere mögliche Verbesserung wäre der Einsatz eines Iris-Trackers für die genaue Bestimmung der Blickrichtung des Betrachters.

Eine Erhöhung der Plausibilität im Rahmen von binauralen Raumsimulationen liesse sich wohl auch erzielen, wenn es möglich wäre, die bisher statische visuelle Szene mit dynamischen Elementen zu bereichern. Denkbar wäre dies beispielsweise mittels einer Blue-Screen-Video-Aufnahme von Musikern (für das Beispiel des UDK-Hörsaals, siehe Abbildung A.7), die in die Darstellung des statischen Panoramas hineingerendert werden.

Die Erweiterung des StereoViewers um weitere Formate für den Import von künstlichen 3D-Szenen würde ebenfalls die Bandbreite möglicher Anwendungen vergößern. Denkbar wäre beispielsweise ein Import der im VRML-Format vorliegenden 3D-Szene des VEP-Projektes[2].

[2] Lombardo u. a. 2005.

7 Literatur

Assenmacher, I., T. Kuhlen und T. Lentz (2005). „Binaural acoustics for CAVE-like environments without headphones“. In: *Eurographics Symposium on Virtual Environments*.

Assenmacher, I. u. a. (2004). „Integrating real-time binaural acoustics into VR applications“. In: *Procs EGVE* 4, S. 129–136.

AVIE. *AVIE - Advanced Visualisation and Interaction Environment*. URL: `http://www.icinema.unsw.edu.au/projects/infra_avie.html` (besucht am 04. 03. 2009).

Bourke, Paul (1999). *Calculating Stereo Pairs*. URL: `http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereorender/` (besucht am 04. 03. 2009).

– (2001). *Creating correct stereo pairs from any raytracer*. URL: `http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereorender/` (besucht am 04. 03. 2009).

– (2002). *Stereoscopic 3D Panoramic Images*. URL: `http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereopanoramic/` (besucht am 06. 03. 2009).

– (2006). „Synthetic Stereoscopic Panoramic Images.“ In: *VSMM*. Hg. von Hongbin Zha u. a. Bd. 4270. Lecture Notes in Computer Science. Springer, S. 147–155. ISBN: 3-540-46304-6. URL: `http://dblp.uni-trier.de/db/conf/vsmm/vsmm2006.html#Bourke06`.

– (2007). *Offaxis frustums - OpenGL*. URL: `http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereorender/` (besucht am 06. 03. 2009).

Bruce, Vicki., Mark A. Georgeson und Patrick R. Green (2006). *Visual perception : physiology, psychology, & ecology*. Hove [u.a.]: Psychology Press.

Chen, Shenchang Eric (1995). „QuickTime VR — An Image-Based Approach to Virtual Environment Navigation“. In: *Computer Graphics* 29.Annual Conference Series, S. 29–38. URL: `citeseer.ist.psu.edu/chen95quicktime.html`.

Cruz-Neira, Carolina, Daniel J. Sandin und Thomas A. DeFanti (1993). „Surround-screen projection-based virtual reality: the design and implementation of the CAVE“. In: *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, S. 135–142. ISBN: 0-89791-601-8. DOI: `http://doi.acm.org/10.1145/166117.166134`.

Dalzell, John Moir. und Eugene F. Linssen (1953). *Practical stereoscopic photography*. London: The Technical Press Ltd.

Duke Gledhilla Gui Yun Tian, Dave Taylora und David Clarke (2003). „Panoramic imaging — a review“. In: *Computers & Graphics* 27 (3 2003), S. 435–445. DOI: http://dx.doi.org/10.1016/S0097-8493(03)00038-4.

d'Angelo, Pablo (2007). „Radiometric alignment and vignetting calibration“. In: *The 5th International Conference on Computer Vision Systems, 2007*. DOI: http://dx.doi.org/10.2390/biecoll-icvs2007-179.

Goldstein, E. Bruce (2002). *Wahrnehmungspsychologie*. Heidelberg: Spektrum Akademischer Verlag.

Grau, Oliver (2003). *Virtual art : from illusion to immersion*. Cambridge, Mass. [u.a.]: MIT.

Henning, Peter A. und Holger Vogelsang (2007). *Taschenbuch Programmiersprachen*. 2. Aufl. Hanser Verlag.

Javidi, Bahram und Fumio Okano (2002). *Three-dimensional television, video and display technology*. Berlin: Springer.

Kemner, Gerhard (1989). *Stereoskopie*. Berlin: Museum für Verkehr und Technik.

Kerr, Douglas A. (2008). *The Proper Pivot Point for Panoramic Photography*. URL: http://doug.kerr.home.att.net/pumpkin/index.htm#PanoramicPivotPoint (besucht am 08. 01. 2009).

Kilgard, Mark J. (1994). *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3*. URL: http://www.opengl.org/resources/libraries/glut/spec3/spec3.html (besucht am 07. 03. 2009).

Lindau, A., T. Hohn und S. Weinzierl (2007). „Binaural resynthesis for comparative studies of acoustical environments“. In: *Audio Engineering Society, 122nd Convention*.

Lindau, Alexander (2006). „Ein Instrument zur softwaregestützten Messung binauraler Raumimpulsantworten“.

Lombardo, V. u. a. (2005). „The Virtual Electronic Poem (VEP) Project“. In: *ICMC Proceedings*. Barcelona.

Nayar, Shree K. (1997). „Catadioptric Omnidirectional Camera“. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 0, S. 482. ISSN: 1063-6919. DOI: http://doi.ieeecomputersociety.org/10.1109/CVPR.1997.609369.

Peleg, Shmuel und Moshe Ben-Ezra (1999). „Stereo Panorama with a Single Camera“. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 1, S. 1395. ISSN: 1063-6919. DOI: http://doi.ieeecomputersociety.org/10.1109/CVPR.1999.786969.

Peleg, Shmuel, Moshe Ben-ezra und Yael Pritch (2001). „Omnistereo: Panoramic stereo imaging“. In: *IEEE Trans. Pattern Analysis and Machine Intelligence* 23, S. 279–290.

Peleg, Shmuel, Yael Pritch und Moshe Ben-Ezra (2000). „Cameras for Stereo Panoramic Imaging“. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference*

on 1, S. 1208. ISSN: 1063-6919. DOI: http://doi.ieeecomputersociety.org/10.1109/CVPR.2000.855821.

Poggio, Gian F. und Tomaso Poggio (1984). „The Analysis of Stereopsis“. In: *Ann. Rev. Neurosci.* 7, S. 379–412.

Pritch, Y., M. Ben-Ezra und S. Peleg (2000). „Automatic disparity control in stereo panoramas (OmniStereo)“. In: *Omnidirectional Vision, 2000. Proceedings. IEEE Workshop on*, S. 54–61. DOI: 10.1109/OMNVIS.2000.853805.

Ray, Sidney F. (1988). *Applied photographic optics : imaging systems for photography, film, and video.* London; Boston: Focal Press.

Rossum, Guido Van (1993). „An introduction to Python for UNIX/C programmers“. In: *Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group.*

Sanner, M. F. (1999). „Python: a programming language for software integration and development.“ In: *J Mol Graph Model* 17.1, S. 57–61. ISSN: 1093-3263. URL: http://view.ncbi.nlm.nih.gov/pubmed/10660911.

Shreiner, Dave u. a. (2005). *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition).* Addison-Wesley Professional. ISBN: 0321335732.

Shum, Heung yeung u. a. (1997). *Panoramic Image Mosaics.* Techn. Ber. Microsoft Research.

Voelter, Markus (2003). „Plug-Ins – Applikationsspezifische Komponenten“. In: *JavaSpektrum* 2.

Weisstein, Eric W. *Cylindrical Equidistant Projection.* URL: http://mathworld.wolfram.com/CylindricalEquidistantProjection.html (besucht am 07. 03. 2009).

Wheatstone, C. (1838). „On some remarkable, and hitherto unobserved, Phenomena of Binocular Vision“. In: *Philosophical Transactions of the Royal Society, London* (128 1838), S. 371–394.

Wikipedia (2009). *Kartennetzentwurf.* URL: http://de.wikipedia.org/w/index.php?title=Kartennetzentwurf&stableid=56190134 (besucht am 09. 03. 2009).

Wimmer, P (2004). „Aufnahme und Wiedergabe stereoskopischer Videos im Anwendungsbereich der Telekooperation (Diplomarbeit)“. Magisterarb. Johannes Kepler Universität Linz, Institut für Telekooperation.

Wright, M.; A. Freed; A. Momeni (2003). „Open Sound Control: State of the Art 2003“. In: *International Conference on New Interfaces for Musical Expression.* OpenSound Control. Montreal, S. 153–159. URL: http://cnmat.berkeley.edu/publications/open_sound_control_state_art_2003.

A Anhang

A.1 Kommandozeilenparameter

A.1.1 StereoViewer

```
Usage: StereoViewer.py [options]

Options:
  -h, --help            show this help message and exit
  -c CONFIGFILE, --configfile=CONFIGFILE
                        a XML file to use as configuration
  -m MODE, --mode=MODE  the stereo mode, must be anaglyph, interlaced, dlp3d,
                        dual or framesequential
  -s SCENE, --scene=SCENE
                        an external scene description file in 3ds format
  -r RESOLUTION, --resolution=RESOLUTION
                        the window size, e.g. 640x480
  -o LISTENINGPORT, --listeningport=LISTENINGPORT
                        the port for OSC communication
  -v, --verbose         Print verbose messages
```

A.1.2 PanoramaCreator

```
usage: panorama.py session side transformation [step=float] [smoothing=boolean]
  session:        name of the stereo session to process
  side:           one of "left" or "right"
  transformation: one of "raw2tiff", "fish2rect", "rect2pano" or "all"
  step:           rotation degrees between adjacent images (optional)
  smoothing:      1 or 0 (optional)
```

A.2 Verzeichnisstrukturen

A.2.1 StereoViewer

```
Data/                   # Datenverzeichnis
    captures/           # Standard Screenshots-Verzeichnis
    config/             # Konfigurationsdateien
        default.xml     # Standard-Konfigurationsdatei
    scenes/             # Szenenbeschreibungen (im 3ds-Format)
    stencil/            # Stencil-Dateien
    textures/           # Texturen
    Definitions.py      # Applikationsweite Einstellungen (Fehlermeldungen,
                        # definierte Hooks, Displaymodi, Hilfe-Texte)
DisplayHandler/         # DisplayHandler (Renderer)
    __init__.py         # Moduldatei mit Basisklasse
    Anaglyph.py         # Anglyph-Renderer
    FrameSequential.py  # Frame-Sequentiell-Renderer
    Interlaced.py       # Interlaced-Renderer
    DLP3d.py            # DLP3D-Renderer
    Dual.py             # Dual-Renderer
DisplayLists/           # DisplayLists (3D-Modelle)
    __init__.py         # Moduldatei mit Basisklasse
    CustomList.py       # 3D-Model für 3ds-Szenen
    PanoramaList.py     # 3D-Model für Panoramaansichten
Plugins/                # Plugins
    __init__.py         # Moduldatei mit Basisklasse und PluginManager
    Capture.py          # Screenshots
    KeyControl.py       # Tastatursteuerung
    OSCControl.py       # OSC-Kommunikations-Server
    Statistic.py        # Erweiterbares Statistikplugin (im Moment nur FPS)
    z800.py             # Unterstützung für eMagin Z800
Util/                   # Tools
    Config.py           # Handler für XML und Kommandozeilenparameter
    Vector.py           # Vektor-Objekt
    xmlobject.py        # XML-Parser
Application.py          # Anwendungs-Klasse
StereoViewer.py         # Startdatei
```

A.2.2 PanoramaCreator

```
images/                 # Einzel- und Panoramabilder
    session name/       # eindeutiger Name der Aufnahmesession
        left/
            raw/        # kameraspezifische RAW-Dateien
            fish/       # Bilder in Fischaugen-Projektion
            rect/       # Rectilinear transformierte Bilder
        right/
            raw/
            fish/
            rect/
scripts/                # zusätzliche Skripte
    templates/          # Templates für die Panoramaerstellung
        pto.tpl         # Haupt-Template für Hugin-Projektdatei
        pto_image.tpl   # Zeilen-Template für Hugin-Projektdatei
    panorama.py         # Python-Skript zur Panoramaerstellung
    extract_raw         # Extraktion der RAW-Bilder (ruft panorama.py auf)
    create_rect         # Rectilineare Projektion (ruft panorama.py auf)
    create_panorama     # Panorama-Stitching (ruft panorama.py auf)
    create_all          # Zusammenfassung der anderen Shell-Skripte
```

A.2.3 Platzbedarf

Der angegebene Platzbedarf wird temporär benötigt, um die Einzelbilder zu einem, aus zwei Panoramabildern bestehenden, stereoskopischen Panoramabild zusammenzufügen. Nach dem das Stereo-Panorama erzeugt wurde, können alle Bild-Dateien außer den RAW-Dateien und den erzeugten Panoramas wieder gelöscht werden, da sie bei Bedarf jederzeit wieder erzeugt könnten.

Datei-Typ	**Größe**	**Anzahl**	**Gesamtgröße**
RAW-Bild	12,9 MB	720	9,3 GB
Extrahiert (Fischauge)	35,5 MB	720	25,6 GB
Rektangulare	43,3 MB	720	31,2 GB
Panorama	80,1 MB	2	160,2 MB
Gesamtbedarf			66,2 GB

Tabelle A.1: Platzbedarf für die Erstellung eines Stereo-Panoramas bei einer angenommenen Rotationsschrittweite von 1°

A.3 Bedienung

A.3.1 OSC

Die Steuerung der Software über OSC setzt einen OSC-Client voraus, der an einen konfigurierbaren Port OSC-Datenpakete schickt. Die aktuell definierten OSC-Pfade und ihre Funktion, sowie die erwarteten Parameter sind der folgenden Tabelle zu entnehmen.

OSC-Pfad	**erwartete Parameter**	**Funktion**
tracker/move/pan	Winkel (floating point)	Rotation um die Körperachse
tracker/move/tilt	Winkel (floating point)	Rotation um die Augenachse
tracker/reset	keine	Blickrichtung auf Nulllage zurücksetzen
stereo_viewer/render/on	keine	Rendering aktivieren
stereo_viewer/render/off	keine	Rendering deaktivieren
stereo_viewer/render/mode	neuer Display-Modus (string)	Ändern des aktuellen Display Modus
stereo_viewer/render/stereoon	keine	Stereo aktivieren
stereo_viewer/render/stereooff	keine	Stereo deaktivieren
stereo_viewer/image/toggle	keine	Rechtes und linkes Bild vertauschen
stereo_viewer/image/load	Dateipfad (string)	Neue Datei laden

Tabelle A.2: OSC-Kommandos

A.3.2 Tastatur

Das Tastaturmenu ist in Ebenen gegliedert. Mit STRG-C kann in den Steuerungsmodus gewechselt werden, in dem weitere Tastaturbefehle zur Verfügung stehen. Um den Steuerungsmodus zu verlassen muss ESC gedrückt werden. Außerhalb des Steuerungsmodus wird das Programm mit ESC beendet. Bei aktiviertem Steuerungsmodus werden die gerade gültigen Tastaturbefehle in der rechten oberen Bildschirmecke angezeigt.

Shortcut	**Menu-Ebene**	**Funktion**
STRG-C	global	Steuerungsmodus aktivieren
ESC	global	Programm beenden
s	global	Stereo-Modus An/Aus
l	global	rechtes und linkes Bild tauschen
ESC	Steuerungsmodus	Steuerungsmodus beenden
STRG-M	Steuerungsmodus	Anzeige-Kontrolle
STRG-A	Steuerungsmodus	Bildwinkel-Kontrolle
STRG-F	Steuerungsmodus	Fokus-Kontrolle
STRG-E	Steuerungsmodus	Stereobasis-Kontrolle
d	Anzeige-Kontrolle	Default-Modus (monsokopisch)
a	Anzeige-Kontrolle	Anaglyph-Modus
i	Anzeige-Kontrolle	Interlaced-Modus
3	Anzeige-Kontrolle	DLP3D-Modus
2	Anzeige-Kontrolle	Dual-Modus
f	Anzeige-Kontrolle	Frame-Sequentiell-Modus
+	Bildwinkel, Fokus, Stereobasis	Wert erhöhen
SHIFT +	Bildwinkel, Fokus, Stereobasis	Wert schnell erhöhen
-	Bildwinkel, Fokus, Stereobasis	Wert vermindern
SHIFT -	Bildwinkel, Fokus, Stereobasis	Wert schnell vermindern

Tabelle A.3: Übersicht der Tastaturkürzel und ihrer Gültigkeitsbereiche

A.4 Kalibrierung des fotografischen Aufnahmesystems

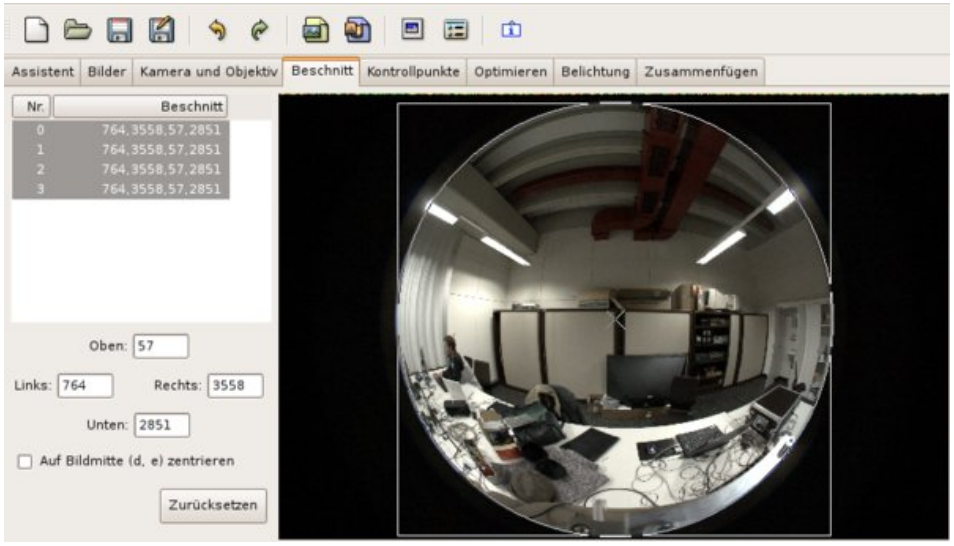


Abbildung A.1: Hugin-Programmfenster: Beschnitt

Für die Kalibrierung ist, von einigen Abweichenungen abgesehen, genauso vorzugehen, als sollte ein normales 360°-Panorama erstellt werden. Auf der *Hugin*-Projekt-Seite[1] im Abschnitt *Tutorials* gibt es detaillierte Kalibrierungsanweisungen[2].

Für die vorliegende Arbeit werden vier Einzelbilder verwendet, die mit dem zu kalibrierenden Aufnahmesystem erstellt wurden. Eine gute Voraussetzung ist die Verwendung von Bildern, die in aufeinanderfolgenden 90°-Schritten aufgenommen wurden, da so der gesamte 360°-Bereich in gleichmässig verteilte Unterabschnitte aufgeteilt wird. Die Bilder müssen beschnitten und mit Kontrollpunkten verbunden werden um anschließend den Stitchingprozess zu starten. Die genaue Verarbeitung für die Zwecke der Linsen-Kalibrierung mit *Hugin* wird im folgenden beschrieben.

A.4.1 Bildbeschnitt

Da das Objektiv nicht die gesamte Fläche des in der Kamera verwendeten Chips benutzt, müssen die Bilder beschnitten werden. Dazu wird der Reiter *Beschnitt* gewählt. In der linken Liste werden alle vier Bilder ausgewählt. In der großen Bildansicht rechts, wird der Kreis so gut wie möglich mit der Bildbegrenzung in Übereinstimmung gebracht (siehe Abbildung A.1). Mit der in dieser Arbeit verwendeten Kamera-Objektiv-Kombination muss die Option *auf*

[1] http://hugin.sourceforge.net
[2] http://hugin.sourceforge.net/tutorials/calibration/en.shtml

Bildmitte zentrieren deaktiviert werden, da das Objektiv nicht mittig auf dem Kamerachip abbildet.

A.4.2 Kontrollpunkte

Als Kontrollpunkten werden solche Punkte bezeichnet, die beiden Bildern eines Bildpaares gemein sind. Kontrollpunkte lassen sich über den Reiter *Kontrollpunkte* im *Hugin*-Hauptfenster erstellen. Für jedes benachbarte Bildpaar werden etwa 20 korrespondierende Punkte (siehe Abbildung A.2) markiert. Je besser diese über den vertikalen Bereich des Bildes verteilt sind, umso besser werden die zu extrahierenden Parameter das verwendete fotografische System beschreiben.

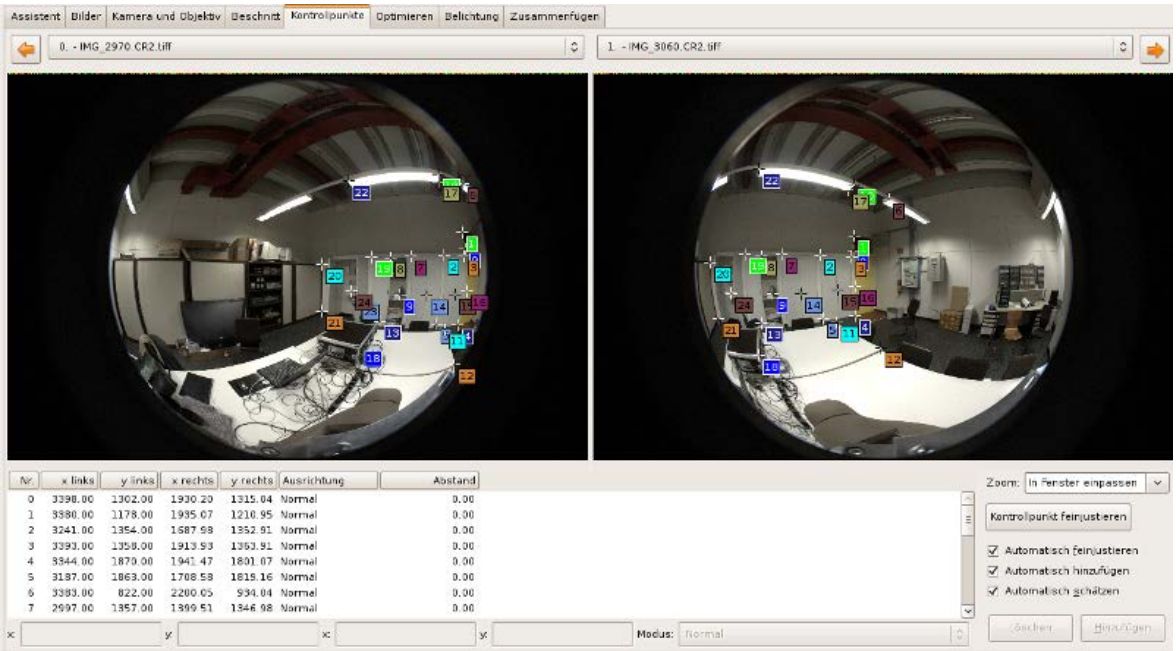


Abbildung A.2: Hugin-Programmfenster: Kontrollpunkte

A.4.3 Optimierung

Sobald ausreichend Kontrollpunkte hinzugefügt wurden, werden die Einzelbilder mit *Hugin* ausgerichtet und optimiert[3]. Das Ausrichten geschieht durch den Wechsel zum Reiter *Assistent* und Anklicken der Schaltfläche *Ausrichten*. Zum Optimieren muss zum Reiter *Optimieren* gewechselt werden. Dort können verschiedene Ziel-Parametergruppen gewählt werden. Für die Zwecke dieser Arbeit sollen die Ausrichtung, der Bildwinkel und die Verzeichnung

[3]Zum Verfahren vgl. d'Angelo 2007.

optimiert werden. Dabei versucht *Hugin* die in Linsen-Parameter so zu optimieren, dass die markierten Kontrollpunkte optimal zusammenpassen.

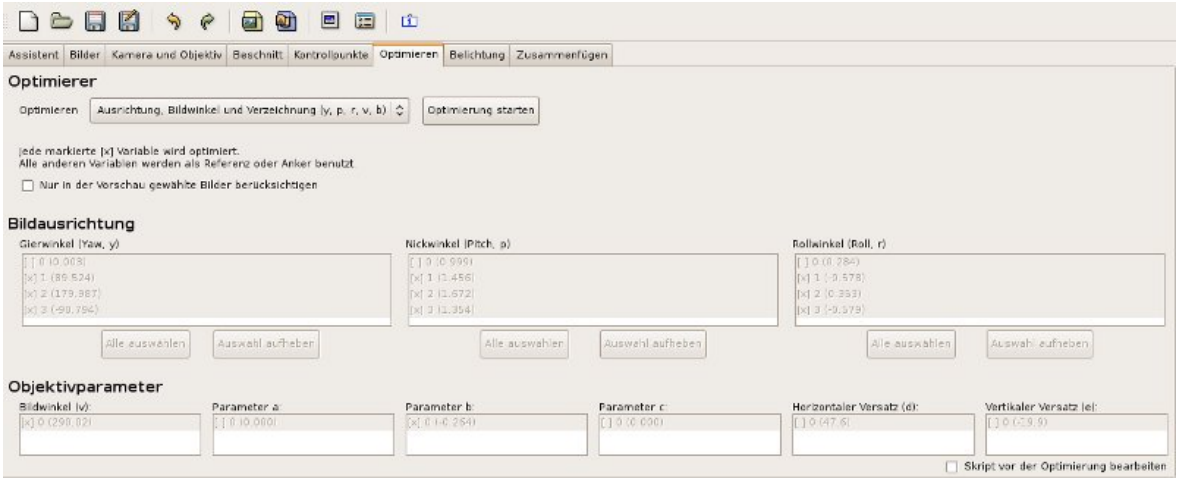


Abbildung A.3: Hugin-Programmfenster: Optimierung

A.4.4 Parameter

Sobald die Optimierung abgeschlossen wurde kann das Stitchen beginnen. Dazu wird zum Reiter *Zusammenfügen* gewechselt. Unter anderem werden dafür mit *nona* auch die einzelnen Fischaugenbilder in rectilineare Bilder transformiert. Die bei diesem Schritt gewonnenen Parameter können der temporär erzeugten *Hugin*-Projektdatei entnommen und in Template-Dateien im Verzeichnis `panorama_creator/scripts/templates` eingefügt werden. Diese Templates werden später für die automatische Bildverarbeitung verwendet.

In Abbildung A.4 ist das *Hugin*-Programmfenster *Zusammenfügen* zu sehen. Dort können Bildwinkel, Größe des zu erzeugenden Panoramabildes, Beschnittsparameter und diverse Ausgabeparameter angegeben werden. Für die Kalibrierung ist es notwendig sicherzustellen, dass der Bildwinkel 180°x180°beträgt, da das der Abbildungsbereich eines Einzelbildes ist[4]. Außerdem sollte mit einem Klick auf *Optimale Größe berechnen* die Größe des zu erzeugenden Panoramabildes der Größe der Einzelbilder angepasst werden. Die Beschnittsparameter geben an, ob und wieviel des resultierenden Panoramas abgeschnitten werden sollen. Dies kann vernachlässigt werden. Bei den Ausgabeparametern muss die Ausgabe der umgerechneten Einzelbilder aktiviert sein. Damit kann die Qualität der Parameter im Hinblick auf die benötigte Projektionstransformation überprüft werden, indem die umgerechneten Bilder nach Ablauf der Bearbeitung von Hand überprüft werden. In ihnen sollte senkrechte Linien wirk-

[4]Es sollte nicht vergessen werden, dass das Ziel dieser Operation darin besteht die Parameter zu erhalten, mit denen die Einzelbilder umgerechnet werden können. Es soll kein Panorama erstellt werden.

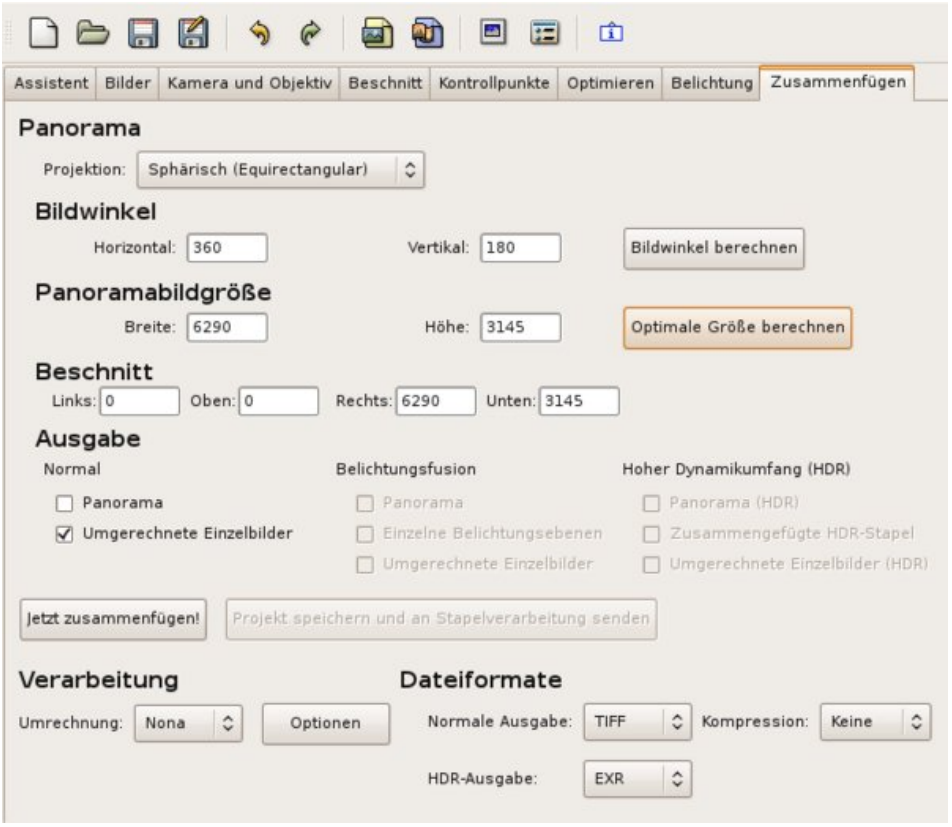


Abbildung A.4: Hugin-Programmfenster: Zusammenfügen

lich senkrecht und gerade sein, horizontale Linien aber gewölbt[5].

Nachdem die Einstellungen vorgenommen wurden, kann mit einem Klick auf *Jetzt Zusammenfügen* der Stitchingprozess beginnen. Dabei öffnet sich ein neues *Hugin*-Fenster in dem man die Kommandozeilenbefehle und -ausgaben sehen kann, mit denen die Bearbeitung vorgenommen wird. *Hugin* erzeugt eine temporäre Projektdatei, die als Argument für *nona* verwendet wird. Während die Bearbeitung läuft, muss diese Datei, die sofort nach Ende der Bearbeitung wieder gelöscht würde, manuell an einen sicheren Ort kopiert werden. Wenn die transformierten Bilder den Erwartungen entsprechen (siehe Abbildung 4.5), dient diese Datei als Ausgangspunkt für die Erstellung der Templatedateien.

Falls die transformierten Bilder Fehler aufweisen, die Projektion z.B. nicht rectilinear ist, sollten mehr Kontrollpunkte hinzugefügt werden, die Optimierung wiederholt und anschließend das Stitchen neu gestartet werden.

[5]Siehe Abschnitt 3.5.

```
1 # hugin project file
2 #hugin_ptoversion 2
3 p f2 w3144 h3144 v180  E24.764 R0 n"TIFF_m c:NONE r:CROP"
4 m g1 i0 f0 m2 p0.00784314
```

Listing A.1: Header einer Hugin-Projektdatei

```
1 # hugin project file
2 #hugin_ptoversion 2
3 p f2 w3144 h3144 v180  E24.764 R0 n"TIFF_m c:NONE r:CROP"
4 m g1 i0 f0 m2 p0.00784314
5
6 # image lines
7 {image_lines}
```

Listing A.2: Template für die Hugin-Projektdatei

A.4.5 Templates

Die im Folgenden zu erstellenden Templatedateien werden später zur automatischen Erzeugung der stereoskopischen Vollpanoramen dienen. Das entwickelte System verwendet zwei Templatedateien, welche zur Laufzeit des in Abschnitt 4.7 beschriebenen Programms *panaorama.py* zu einer temporären *Hugin*-Projektdatei zusammengestellt werden. Diese temporäre Datei fungiert dann als Kommandozeilenparameter für das Stitchingprogramm *nona*[6].

Eine *Hugin*-Projektdatei[7] besteht grundsätzlich aus einem Header und einer Dateiliste. Der Header der im letzten Abschnitt erhaltenen *Hugin*-Projektdatei sollte dem in Listing A.1 entsprechen. Er bestimmt u.a. Format, Größe und Bildwinkel der zu erzeugenden Dateien.

Dieser Header wird kopiert und in eine neue Datei mit dem Namen `pto.tpl` eingefügt. Diese Datei liegt standarmässig im Verzeichnis `panorama_creator/scripts/templates`. Sie bildet das äußere Gerüst der späteren *Hugin*-Projektdatei. Um später die Einträge der zu verarbeitenden Bilder in dieses Template einfügen zu können, wird noch der Platzhalter `{image_list}` an das Ende der Datei angehängt (siehe Listing A.2).

Der dem Header folgende Bereich listet alle geladenen und mit Kontrollpunkten verbundene Einzelbilder auf und dient als Vorlage für die zweite Templatedatei. Jeder Eintrag der Dateiliste führt eine spezifische Grafikdatei mit ihren für die Umrechnung relevanten Eigenschaften auf. Zu den Eigenschaften gehören Breite, Höhe, Projektionsformat und diverse Verzeichnungsparameter[8]. In der Bildliste entspricht jede Zeile genau einem Einzelbild. In

[6] Siehe Abschnitt 4.7.3.
[7] Vgl. `http://archive.bigben.id.au/tutorials/360/readme/`.
[8] Siehe Abschnitt A.4.

```
1 #-hugin  cropFactor=1.6
2 i w4312 h2876 f2 Eb1 Eev24.7618712968535 Er1 Ra2.23700332641602 Rb4
    .23208475112915 Rc-2.94551563262939 Rd0.730718314647675 Re
    -0.345469862222672 Va1 Vb0.116392014318528 Vc5.37190666977497 Vd
    -28.9757618933515 Vx0 Vy0 a1.03605231460659 b-1.7584301009951 c0
    .698874443989413 d66.978171955671 e12.909877797027 g0 p0
    .537471106371066 r-0.255229304174506 t0 v252.795167454931 y
    -0.00451627222693674  Vm5 u10 n"fish_0000.tiff"
```

Listing A.3: Zeile der Bildliste, die Parameter beschreiben die Transformation der Datei fish_0000.tiff

```
1 #-hugin  cropFactor=1.6
2 i w4312 h2876 f2 Eb1 Eev24.7618712968535 Er1 Ra2.23700332641602 Rb4
    .23208475112915 Rc-2.94551563262939 Rd0.730718314647675 Re
    -0.345469862222672 Va1 Vb0.116392014318528 Vc5.37190666977497 Vd
    -28.9757618933515 Vx0 Vy0 a1.03605231460659 b-1.7584301009951 c0
    .698874443989413 d66.978171955671 e12.909877797027 g0 p0
    .537471106371066 r-0.255229304174506 t0 v252.795167454931 y
    -0.00451627222693674  Vm5 u10 n"{image_path}"
```

Listing A.4: Template für einen einzelnen Eintrag in der Bildliste

Listing A.3 ist exemplarisch ein Eintrag dieser Liste dargestellt.

Jede Zeile beschreibt die notwendigen Parameter für die Transformation einer Bilddatei (im Beispiel: `fish_0000.tiff`). Um daraus ein Template zu erstellen, muss lediglich der Dateiname durch einen Platzhalter ersetzt werden, so wie es in Listing A.4 dargestellt ist. Alle anderen Angaben können unberührt bleiben.

Die Templatedatei wird unter dem Namen `pto_image.tpl` ebenfalls im Verzeichnis `panorama_creator/scripts/templates` gespeichert.

Alle restlichen Zeilen der *Hugin*-Projektdatei können ignoriert werden, da es sich um Anweisungen handelt, die nur für die Verwendung von *Hugin* als grafischer Benutzeroberfläche von Bedeutung sind.

A.5 Hugin: vollständige Projektdatei

```
1  # hugin project file
2  #hugin_ptoversion 2
3  p f2 w3144 h3144 v180  E24.764 R0 n"TIFF_m c:NONE r:CROP"
4  m g1 i0 f0 m2 p0.00784314
5
6  # image lines
7  #-hugin  cropFactor=1.6
8  i w4312 h2876 f2 Eb1 Eev24.7618712968535 Er1 Ra2.23700332641602 Rb4
       .23208475112915 Rc-2.94551563262939 Rd0.730718314647675 Re
       -0.345469862222672 Va1 Vb0.116392014318528 Vc5.37190666977497 Vd
       -28.9757618933515 Vx0 Vy0 a1.03605231460659 b-1.7584301009951 c0
       .698874443989413 d66.978171955671 e12.909877797027 g0 p0
       .537471106371066 r-0.255229304174506 t0 v252.795167454931 y
       -0.00451627222693674  S764,3558,57,2851 Vm5 u10 n"fish_0000.tiff"
9  #-hugin  cropFactor=1.6
10 i w4312 h2876 f2 Eb1 Eev24.7681477413479 Er1 Ra=0 Rb=0 Rc=0 Rd=0 Re=0 Va
       =0 Vb=0 Vc=0 Vd=0 Vx=0 Vy=0 a=0 b=0 c=0 d=0 e=0 g=0 p0.339983626967463
        r-0.0881518140869662 t=0 v=0 y89.5454243437925  S764,3558,57,2851 Vm5
        u10 n"fish_0089.tiff"
11 #-hugin  cropFactor=1.6
12 i w4312 h2876 f2 Eb1 Eev24.7748034507244 Er1 Ra=0 Rb=0 Rc=0 Rd=0 Re=0 Va
       =0 Vb=0 Vc=0 Vd=0 Vx=0 Vy=0 a=0 b=0 c=0 d=0 e=0 g=0 p0.50556618087212
       r-0.255328308101355 t=0 v=0 y179.896362361476  S764,3558,57,2851 Vm5
       u10 n"fish_0179.tiff"
13 #-hugin  cropFactor=1.6
14 i w4312 h2876 f2 Eb1 Eev24.748144365866 Er1 Ra=0 Rb=0 Rc=0 Rd=0 Re=0 Va=0
        Vb=0 Vc=0 Vd=0 Vx=0 Vy=0 a=0 b=0 c=0 d=0 e=0 g=0 p0.254809948116886 r
       -0.0885927215427348 t=0 v=0 y-90.3875786336626  S764,3558,57,2851 Vm5
       u10 n"fish_0269.tiff"
15
16
17 # specify variables that should be optimized
18 v b0 v0
19 v p1 r1 y1
20 v p2 r2 y2
21 v p3 r3 y3
22 v
23
24 # control points
25 c n0 N1 x2626.77215189873 y1678.40506329114 X1098.0322506021 Y1693
       .6078032763 t0
26 c n0 N1 x2701.82278481013 y1091.64556962025 X1210.70364538739 Y1053
       .12882932281 t0
27 c n1 N2 x2592 y1361 X1041.12458020043 Y1338.94144373786 t0
```

```
28  c n1 N2 x2892.86075949367 y2067.30379746835 X1491.8969808904 Y2070
        .98362411022 t0
29  c n2 N3 x2943 y1045 X1429.8909672506 Y1034.09243074637 t0
30  c n2 N3 x2920 y1936 X1442.89531457727 Y1931.89026147571 t0
31  c n3 N0 x2523 y892 X1152.26371541253 Y802.259400813895 t0
32  c n3 N0 x2946 y1839 X1406.17366774737 Y1830.47191384581 t0
33
34  #hugin_optimizeReferenceImage 0
35  #hugin_blender enblend
36  #hugin_remapper nona
37  #hugin_enblendOptions
38  #hugin_enfuseOptions
39  #hugin_hdrmergeOptions
40  #hugin_outputLDRBlended false
41  #hugin_outputLDRLayers false
42  #hugin_outputLDRExposureRemapped false
43  #hugin_outputLDRExposureLayers false
44  #hugin_outputLDRExposureBlended true
45  #hugin_outputHDRBlended false
46  #hugin_outputHDRLayers false
47  #hugin_outputHDRStacks false
48  #hugin_outputLayersCompression PACKBITS
49  #hugin_outputImageType tif
50  #hugin_outputImageTypeCompression NONE
51  #hugin_outputJPEGQuality 100
52  #hugin_outputImageTypeHDR exr
53  #hugin_outputImageTypeHDRCompression
```

Listing A.5: vollständiges Beispiel einer Hugin-Projektdatei

A.6 DefaultHandler

Um einen Einblick in die Funktionsweise der Display-Handler-Klassen zu geben, wird an dieser Stelle der vollständige Quelltext der DefaultHandler-Klasse wiedergegeben. Die Default-Handler-Klasse ist als monoskopische Standard-Display-Handler-Klasse angelegt. Im Panorama-Modus wird lediglich das linke Bild gerendert. Sie kann als Beispiel-Implementation für die Erstellung neuer Display-Handler-Klassen dienen. Alle Display-Handler-Klassen müssen eine ähnliche Struktur aufweisen.

```
1  class DefaultHandler(BaseHandler):
2    '''
3    This is the default display handler, rendering the given scene or
4    image in simple non-stereoscopic OpenGL.
5
6    In panorama mode only the left image is rendered.
7    In scene mode the 3ds scene will be rendered from a centered position.
8    '''
9
10   def __init__(self, instance):
11     '''
12     initialization of the default display handler
13     '''
14     super(DefaultHandler, self).__init__(instance, DISPLAY_DEFAULT)
15
16   def leave(self):
17     '''
18     do clean up actions when leaving the default display handler
19     '''
20     super(DefaultHandler, self).leave()
21
22   def render(self, displayList):
23     glDrawBuffer(GL_BACK)
24     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
25     super(DefaultHandler, self).renderList(displayList)
26
27   def drawPanorama(self, displayList):
28     self.render(displayList)
29
30   def drawScene(self, displayList):
31     self.render(displayList)
```

Listing A.6: Minimalbeispiel einer Display-Handler-Klasse

A.7 Berechnung des Frustum

Die Implementierung dieser Methode basiert auf den Beschreibungen in Bourke (2007). Die für eine korrekte Konstruktion des Sichtvolumens entsprechend Abschnitt 3.3.4 notwendigen Berechnungen werden durchgeführt.

```
1  def setFrustum(self, eye = None):
2    '''
3    Set the correct frustrum, depending on the requested eye
4    '''
5
6    # near and far clipping plane, depending on maximum shape size
7    # which is defined in Application.py
8    near = -self.application.viewPos.z - self.shapeSize * 0.5
9    if (near < 0.1):
10     near = 0.1
11   far = -self.application.viewPos.z + self.shapeSize * 0.5
12
13   # ratio of window content
14   ratio  = float(self.screenWidth) / float(self.screenHeight)
15
16   # horizontal field of view
17   radians = self.DTOR * self.application.aperture / 2
18   wd2 = near * math.tan(radians)
19   ndfl  = near / self.application.focalLength
20
21   if self.getMode() == DISPLAY_DUAL:
22     ratio /= 2.0
23
24   if eye == GL_LEFT:
25     f =  0.5
26   elif eye == GL_RIGHT:
27     f =  -0.5
28   else:
29     f = 0
30
31   # create boundary distances
32   left   = - ratio * wd2 + f * self.application.eyeSep * ndfl
33   right  =   ratio * wd2 + f * self.application.eyeSep * ndfl
34   top    =   wd2
35   bottom = - wd2
36
37   # set frustum
38   glFrustum (left, right, bottom, top, near, far)
```

Listing A.7: `setFrustum`-Methode der BaseHandler-Klasse

A.8 Render-Methoden

```
1  def render(self, side, displayList, eye = None):
2
3    # activate writing of all color components into the framebuffer
4    glColorMask( True, True, True, True )
5
6    # read from and write into back buffer (read is important for glAccum)
7    glDrawBuffer( GL_BACK )
8    glReadBuffer( GL_BACK )
9
10   if (side == GL_LEFT):
11
12     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT )
13     glClear(GL_ACCUM_BUFFER_BIT)
14
15     # write only red color component into framebuffer
16     glColorMask( True, False, False, False )
17     super(Anaglyph, self).renderList(displayList, eye)
18
19     glAccum( GL_LOAD, 1.0)
20
21   else:
22
23     # don't clear color buffer and accum buffer!
24     glClear( GL_DEPTH_BUFFER_BIT )
25
26     # write only green and blue componenets into framebuffer
27     glColorMask( False, True, True, False )
28     super(Anaglyph, self).renderList(displayList, eye)
29
30     glAccum( GL_ACCUM, 1.0)
31     glAccum( GL_RETURN, 1.0)
32
33   glFlush()
34   glColorMask( True, True, True, True )
```

Listing A.8: `render`-Methode der Anaglyph-Klasse

```
1  def render(self, side, displayList, eye = None):
2    glDrawBuffer(GL_BACK)
3    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
4    super(FrameSequential, self).renderList(displayList, eye)
5    self.left = not self.left
```

Listing A.9: `render`-Methode der FrameSequential-Klasse

```
1  def render(self, side, displayList, eye = None):
2
3    if (side == GL_LEFT):
4      glStencilFunc(GL_NOTEQUAL, 1, 1)
5      glDrawBuffer(GL_BACK)
6      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
7    else:
8      glStencilFunc(GL_EQUAL, 1, 1)
9      glClear(GL_DEPTH_BUFFER_BIT)
10     glDrawBuffer(GL_BACK)
11
12   super(Interlaced, self).renderList(displayList, eye)
```

Listing A.10: `render`-Methode der Interlaced-Klasse

```
1  def render(self, side, displayList, eye = None):
2
3    if (side == GL_LEFT):
4      glStencilFunc(GL_NOTEQUAL, 1, 1)
5      glDrawBuffer(GL_BACK)
6      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
7    else:
8      glStencilFunc(GL_EQUAL, 1, 1)
9      glClear(GL_DEPTH_BUFFER_BIT)
10     glDrawBuffer(GL_BACK)
11
12   super(DLP3D, self).renderList(displayList, eye)
```

Listing A.11: `render`-Methode der DLP3D-Klasse

```
1  def setViewPort(self, side):
2    if (side == GL_LEFT):
3      glViewport(0, 0, self.screenWidth / 2, self.screenHeight)
4    else:
5      glViewport(self.screenWidth / 2, 0, self.screenWidth / 2, self.
         screenHeight)
6
7  def render(self, side, displayList, eye = None, viewPort = None):
8    glDrawBuffer( GL_BACK )
9    if (side == GL_LEFT):
10     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT )
11
12   super(Dual, self).renderList(displayList, eye, viewPort)
```

Listing A.12: `render`-Methode der Dual-Klasse

A.9 Konfigurationsdatei

Listing A.13 zeigt eine gültige XML-Konfigurationsdatei für die StereoViewer-Software. Die kommentierten Angaben sind weitestgehend selbsterklärend.

`scene` Pfade können absolut oder relativ angegeben werden.

`render` Die Attribute `aperture` und `horizontal_offset` erwarten Gradangaben. `focal_length` und `eye_separation` können mit beliebigen, aber übereinstimmenden, Maßeinheiten-angegeben werden.

`frame` Alle Angaben außer Transparenz sind in Grad anzugeben. `Transparenz` erwartet eine Zahl (Gleitkomma oder Integer) zwischen 0 und 1

`plugins` Jedes Plugin kann einen Knoten anlegen.

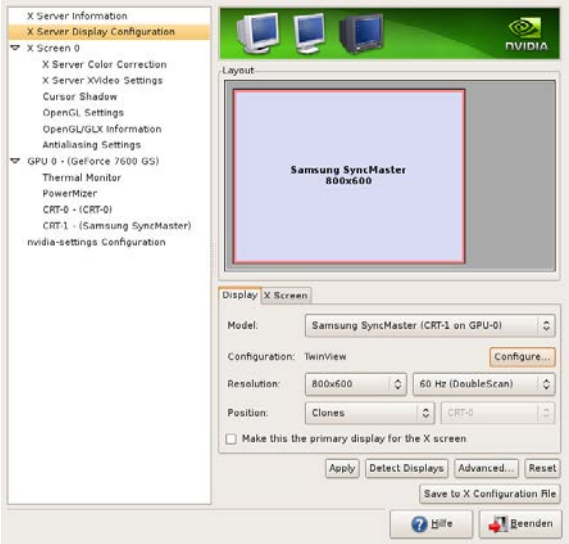
```
1  <?xml version="1.0"?>
2
3  <pyStereoViewer resolution="800x600">
4
5    <!-- image or scene file to use -->
6    <scene path="Data/textures/udk_0_left.tiff" />
7
8    <!-- render settings -->
9    <render aperture="55" focal_length="8" eye_separation="0.07" mode="
         default" horizontal_offset="0" />
10
11   <!-- frame settings -->
12   <frame width="160" height="70" transparency="0.4" horizontal_offset="0"
          vertical_offset="0" border_width="10" />
13
14   <!-- plugin settings -->
15   <plugins>
16
17     <!-- osc plugin attributes -->
18     <osccontrol adress="" port="58500" />
19
20     <!--statistic plugin attributes -->
21     <statistic visible="0" />
22
23     <!--capture plugin attributes -->
24     <capture dir="Data/captures" type="tiff" />
25
26   </plugins>
27
28 </pyStereoViewer>
```

Listing A.13: Beispiel einer Konfigurationsdatei im XML-Format

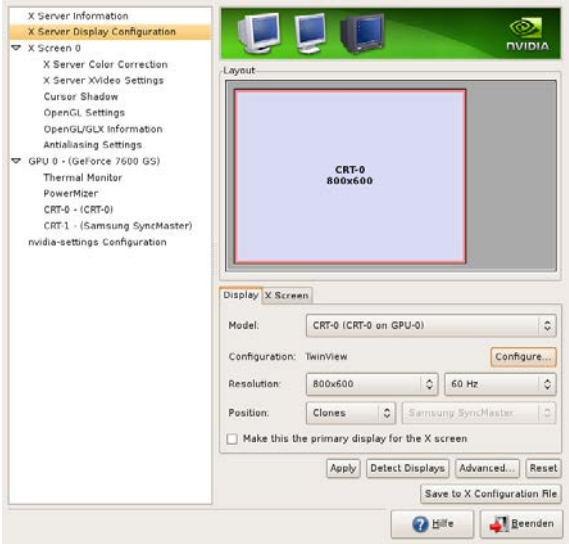
A.10 Konfiguration einer NVIDIA Grafikkarte

Im folgenden werden die Konfigurationseinstellungen einer NVIDIA Grafikkarte für die Verwendung mit zwei speziellen stereoskopischen Ausgabegeräten gezeigt. Es wird vorausgesetzt, dass das Ausgabegerät als Zweitmonitor unter dem Betriebssystem Ubuntu 8.10 betrieben wird.

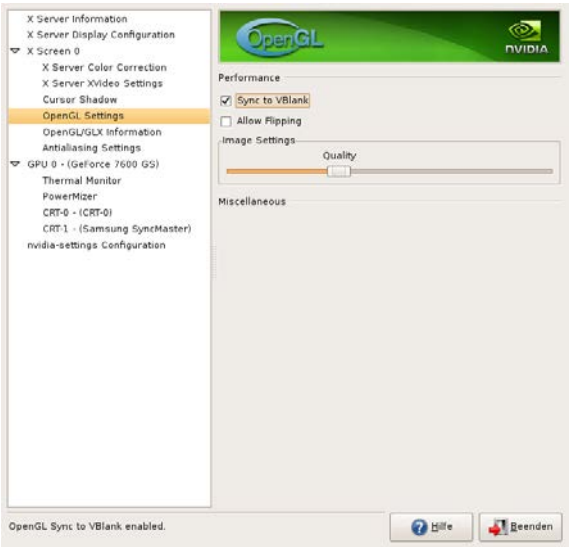
A.10.1 eMagin Z800 HMD



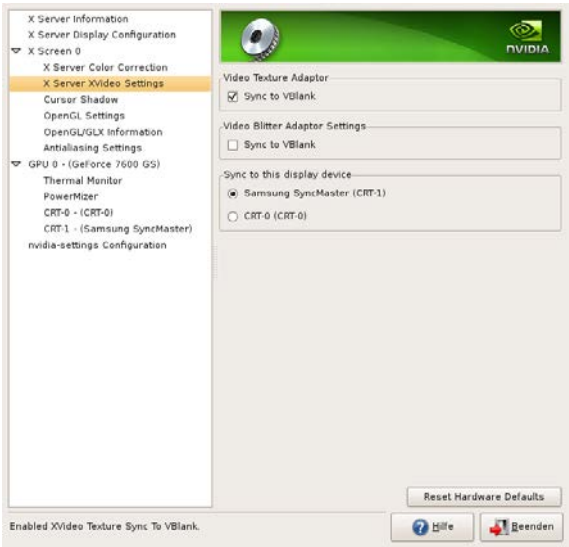
(a) Display-Einstellungen Hauptmonitor



(b) Display-Einstellungen eMagin Z800 HMD



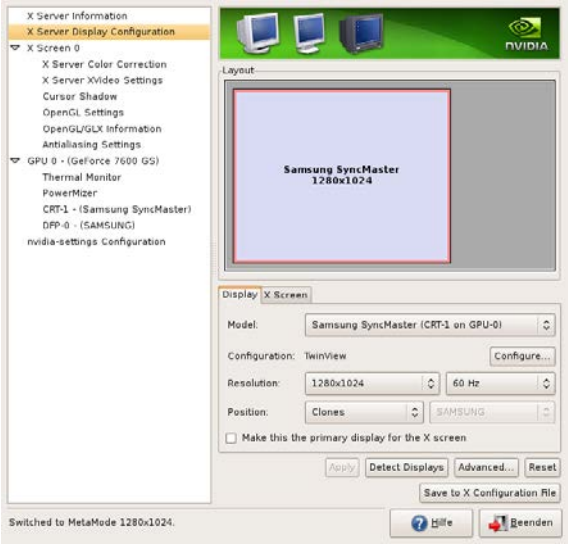
(c) OpenGL Sync



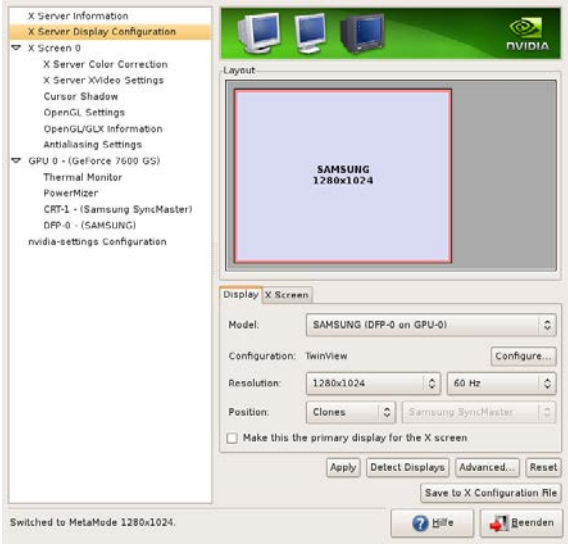
(d) X-Server sync

Abbildung A.5: Grafikkarteneinstellungen: eMagin Z800 HMD

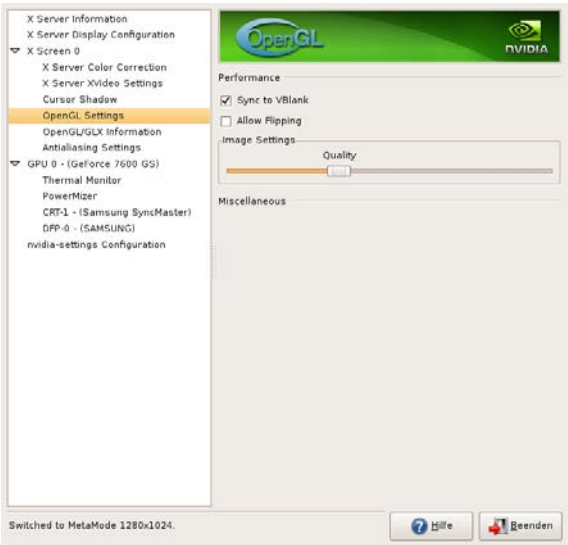
A.10.2 Samsung HL67A750 DLP Fernseher



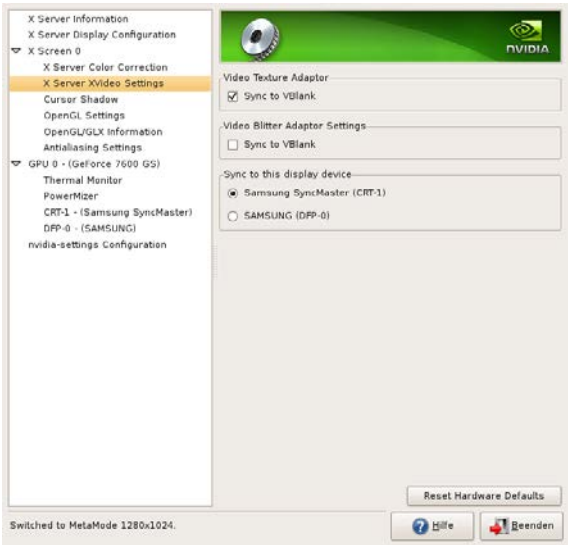
(a) Display-Einstellungen Hauptmonitor



(b) Display-Einstellungen Samsung DLP Fernseher



(c) OpenGL Sync



(d) X-Server sync

Abbildung A.6: Grafikkarteneinstellungen: Samsung DLP Fernseher

Bemerkungen Beide Geräte erfordern, dass die Bildschirmauflösung von Haupt- und Nebenmonitor identisch sind. Die Bildwiederholfrequenz muss in beiden Fällen 60 Hz betragen. In den OpenGL-Einstellungen muss die Option 'Sync to VBlank' aktiviert sein (siehe Abbildungen A.5c und A.6c). Der X-Server-Textur-Adapter muss mit dem Hauptmonitor synchronisiert werden (siehe Abbildungen A.5d und A.6d).

A.11 Captures

Die folgenden Bilder wurden mit dem Capture-Plugin während der Benutzung des Stereo-Viewers erstellt. Sie illustrieren verschiedene Stero-Modi bei Betrachtung eines Panoramas und einer 3D-Szene.

(a) monoskopisch

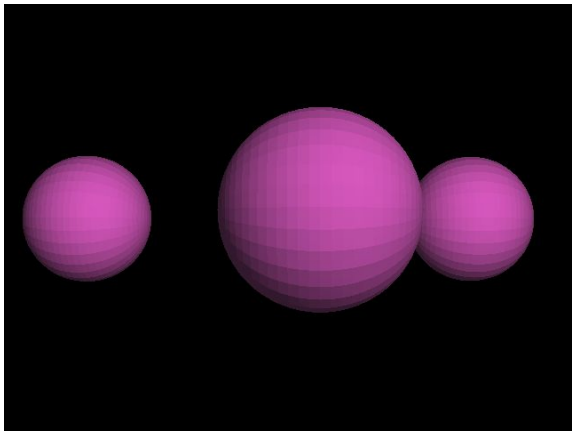
(b) anaglyph

(c) interlaced

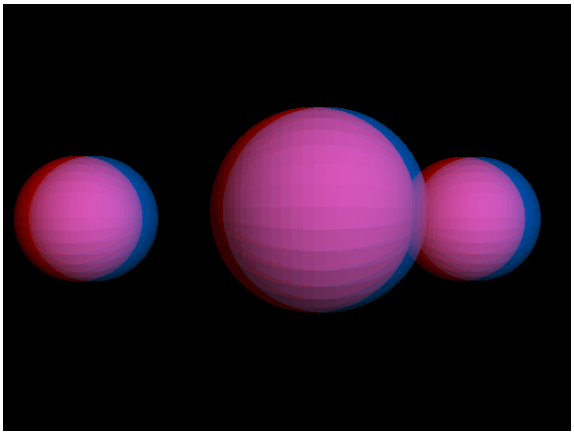
(d) dlp3d

(e) dual

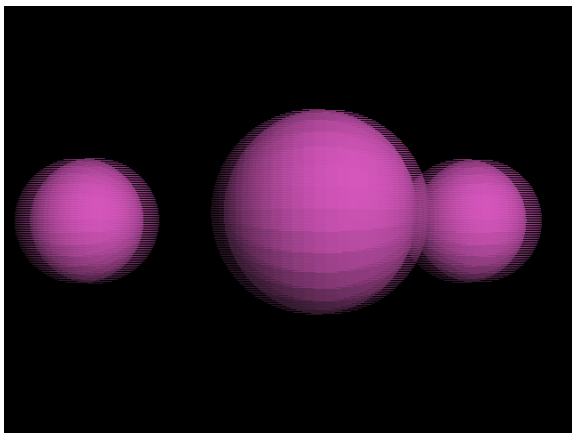
Abbildung A.7: Screenshots des StereoViewers in (a) Default- und (b) Anaglyph- (c) Interlaced- (d) DLP3d- und (e) Dual-Modus



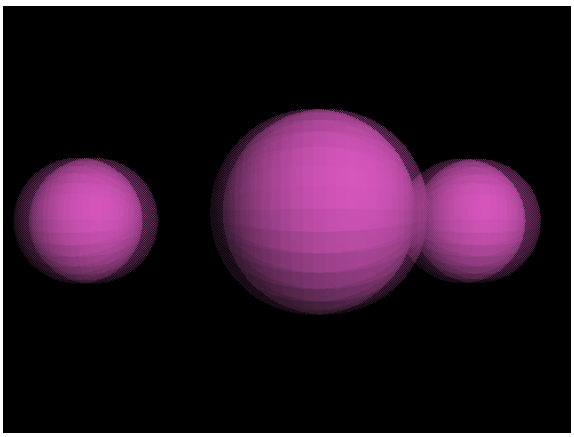
(a) monoskopisch



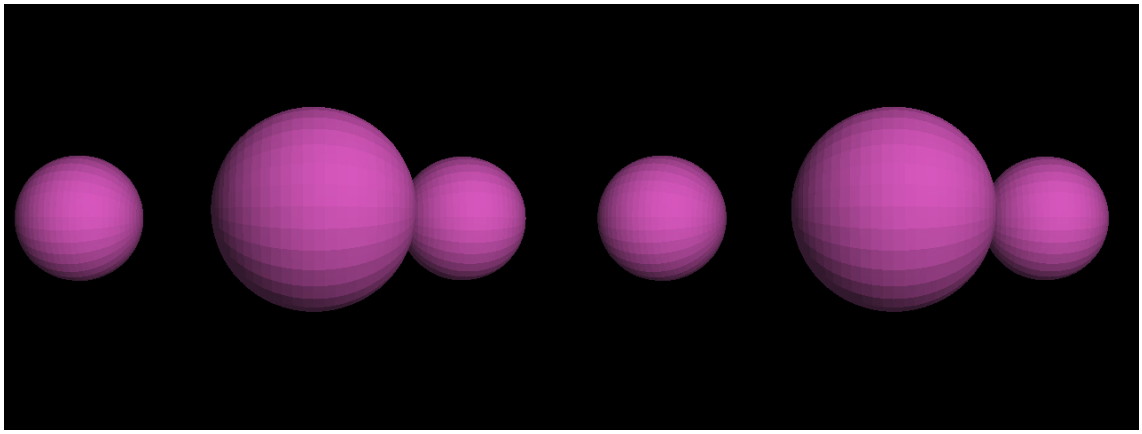
(b) anaglyph



(c) interlaced



(d) dlp3d



(e) dual

Abbildung A.8: Screenshots des StereoViewers in (a) Default- und (b) Anaglyph- (c) Interlaced- (d) DLP3d- und (e) Dual-Modus

Abbildungsverzeichnis

Tabellenverzeichnis

Listings