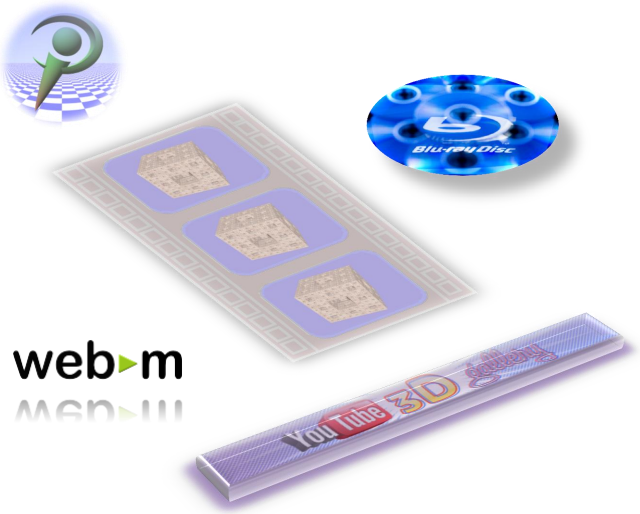
Werkzeuge zur Erstellung stereoskopischer Videosequenzen

Masterarbeit

von

Beat Trachsler

Universität Basel
3. Dezember 2010



Betreuer: Dr. Martin Guggisberg
Gutachter: Prof. Dr. Martin Lehmann

Danksagung

Ich möchte in erster Linie meinem Betreuer Dr. Martin Guggisberg für seine zahlreichen Anregungen und Tipps danken. Er war nicht nur ein wissenschaftlicher Begleiter sondern auch ein kollegialer Gastgeber am Institut für High Performance und Web Computing der Universität Basel, so dass es mir während meiner Basler Tage an nichts gefehlt hat. Unsere zahlreichen Streifzüge durch Basel werden mir in lebhafter Erinnerung bleiben.

Ausserdem danke ich meinem Gutachter und Zweitbetreuer Prof. Dr. Martin Lehmann für die zahlreichen Gespräche in Bern und für die Hardware zur Beamerpräsentation, die er mir nach Basel ausgeliehen hat. Seine offensichtliche Begeisterung für mein Thema wirkte ansteckend und half mir, die technischen Hürden bei der Vorbereitung der Beamerpräsentation zu nehmen.

Zudem danke ich Prof. Dr. Helmar Burkhart und seiner Forschungsgruppe für die Gastfreundschaft an ihrem Institut, wo ich während vier Monaten einen Hightech-Arbeitsplatz inklusive 55"-LED-Display für mich allein nutzen konnte. Einen besonderen Dank möchte ich dabei an Robert Frank, Madan Sathe und Florian Müller richten, die mich bei meinen Hard- und Softwarefragen stets bereitwillig unterstützt haben.

Dr. Tibor Gyalog danke ich für das POV-Ray Programm zum Kurzfilm mit den Blutkörperchen sowie für seine Ratschläge zur Berechnung des Sky-Vektors bei der Kamerafahrt durch den Menger-Schwamm. Hansruedi Hidber danke ich herzlich für die zahlreichen Gespräche über POV-Ray und über die Videokompression sowie für sein Feedback zu meinen S3D-Videos. Seine Tipps waren buchstäblich Gold wert. Greg Wallace, Commercial Director bei NetBlender, möchte ich dafür danken, dass er mir ermöglichte, die Software NetBlender im Rahmen dieser Arbeit kostenlos zu testen und damit eine Blu-ray 3D Disc herzustellen.

Ein besonderer Dank geht an Bernhard Egger, Patrik Huber, Flavio Bernasconi, Stephan Müller und Mauro Bünzli, welche mit ihrem Feedback zu den ersten S3D-Videos entscheidend zur Qualität dieser Arbeit beigetragen haben. Schliesslich möchte ich auch meinem Bruder, Dr. med. Stefan Trachsler, für seine Literaturtipps zur Neuroophthalmologie ganz herzlich danken.

Abschliessend danke ich Tobias Kohn für das Gegenlesen dieser Arbeit sowie für seine Tipps zur Typografie mit LaTeX und zur Optimierung meines Layouts.

Beat Trachsler

Zusammenfassung

Im Rahmen dieser Arbeit werden verschiedene Werkzeuge zur Erstellung von stereoskopischen Videosequenzen getestet und verglichen. Zur Erzeugung der Sequenzen wird das 3D-Visualisierungswerkzeug POV-Ray eingesetzt. Dabei wird ausgehend von bestehenden stereoskopischen Kameras eine für die Erstellung von stereoskopischen Webvideos optimierte Version beschrieben und anhand von fünf Beispielsequenzen getestet. Ausserdem wird der Weg von der Bildsequenz zum Webvideo im webM-Format oder alternativ zur Blu-ray 3D Disc aufgezeigt. Für die Transkodierung einer Bildsequenz ins webM-Containerformat, wird eine Erweiterung des Firefogg Add-ons zum Firefox Browser vorgestellt, welche der Community ab Firefox 4 zur Verfügung steht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1 Einleitung

Das Jahr 2010 war in mancher Hinsicht das Jahr des stereoskopischen Films. Nicht nur dass Hollywood nach dem gewaltigen Erfolg von Avatar im Herbst 2009 mit einer ganzen Reihe von S3D-Produktionen nachzog. Auch die Internationale Funkausstellung Berlin (IFA) stand in diesem Jahr unter dem Motto 3D. Von der neusten Generation der DLP-Projektoren mit Shuttertechnologie bis zum 3D-fähigen HDTV-Receiver wurden Dutzende von 3D-fähigen Produkten präsentiert. Da stellt sich natürlich die Frage, wie das neue Medium für wissenschaftliche Visualisierungsprozesse genutzt werden kann. Mit dem CAVE, einer automatischen virtuellen Umgebung, welche in ein fensterloses Zimmer eingebettet ist, wird bereits seit über einem Jahrzehnt experimentiert. Ein wichtiges Einsatzgebiet von CAVE ist die Molekularbiologie. Wie in [1] beschrieben, werden dabei auch stereoskopische Techniken eingesetzt um eine Raumwahrnehmung zu erzielen. Die zentrale Idee hinter der Stereoskopie ist im Prinzip sehr einfach. Den beiden Augen werden verschiedene Bilder gezeigt, welche aus unterschiedlichen Blickwinkeln aufgenommen oder gerendert wurden. Dabei soll der Augenabstand bei der Positionierung der Kameraobjektive berücksichtigt werden. Auf diese Weise entsteht beim Betrachter ein räumlicher Eindruck. Eine Einführung in die Stereoskopie sowie einen Überblick über 3D-fähige Hardware und die gängigen Videocodecs findet man im Kapitel 2.

Diese Arbeit konzentriert sich in Anlehnung an Arbeiten von Paul Bourke [2, 3] und Wolfgang Wieser [4] auf die Bildsynthese mittels des Raytracers POV-Ray [5]. Hier geht es darum, mithilfe der Version 3.7 von POV-Ray den parallelen Einsatz mehrerer Cores auszunützen. Ausserdem konnte eine gewaltige Zeitersparnis in Abhängigkeit der Bildauflösung erzielt werden, indem die Parameter der stereoskopischen Kamera bereits im Parser und nicht erst in der Render-Engine von POV-Ray berechnet werden. Sämtliche Berechnungen müssen dadurch nur noch einmal pro Teilbild gemacht werden, was besonders bei grossen Auflösungen stark ins Gewicht fällt. Es zeigte sich jedoch recht schnell, dass die eigentliche Herausforderung nicht die mathematische Modellierung der Raumwahrnehmung und deren Umsetzung in POV-Ray war. Vielmehr geht es inzwischen immer mehr darum, die S3D-fähige Hardware gewinnbringend einzusetzen und dabei vom gewaltigen Potenzial moderner Videocodecs zu profitieren. So lassen Bildsequenzen von über einem Gigabyte mit moderner Transkodierungssoftware auf wenige Megabyte reduzieren. Im Rahmen dieser Arbeit wurden dabei zwei verschiedene Workflows untersucht, der Weg zur Produktion einer Blu-ray 3D Disc und die Erstellung von Webvideos im webM- und im mp4-Container. Während man bei der Produktion von Blu-ray 3D Discs noch immer auf den Einsatz kostspieliger Authoring Software angewiesen ist, bieten die Webvideos hier echte Alternativen. Der vp8-Codec von Google kann mit diversen Open Source Tools kostenlos kodiert werden. Auf der anderen Seite bietet Apple mit QuickTime Pro eine kostengünstige Transkodierungssoftware für das H.264 Videoformat im mp4-Container an. Beide Formate kommen im Video-Tag, der im Rahmen von HTML5 eingeführt wird, bereits zur Anwendung. Ausserdem betreibt YouTube eine Testplattform, auf der stereoskopische Videos mit verschiedenen Wiedergabetechniken abgespielt werden können. Die technische Umsetzung vom POV-Ray Code bis zur Blu-ray 3D Disc beziehungsweise bis zum Webvideo wird im Kapitel 3 beschrieben.

Das Kapitel 4 ist den Videos gewidmet. Ausgehend von den Überlegungen von Paul Bourke [6] wird beschrieben, worauf man bei der Herstellung von stereoskopischen Videosequenzen achten sollte. Ausserdem werden ausgewählte Etappen aus dem Produktionsprozess der Kurzfilme beleuchtet und analysiert. Alle hier vorgestellten Kurzfilme liegen auch auf einer Blu-ray 3D Disc sowie auf der Webplattform YouTube im S3D-Format vor.

2 Grundlagen

"Man muss etwas Neues machen,
um etwas Neues zu sehen."

Georg Christoph Lichtenberg

2.1 Räumliches Sehen

Das räumliche Sehen, auch Stereopsis genannt, entsteht im Gehirn durch Kombination verschiedener Informationen, welche durch die beiden Augen gesammelt werden. Das entscheidende dabei ist, dass die Augen aufgrund ihrer unterschiedlichen Position[1] verschiedene Bilder liefern. Fixieren die beiden Augen einen nah gelegenen Gegenstand, kommt es zur Konvergenz, bei der sich die Blickachsen der beiden Augen beim betrachteten Gegenstand schneiden (Vgl. Abbildung 1).

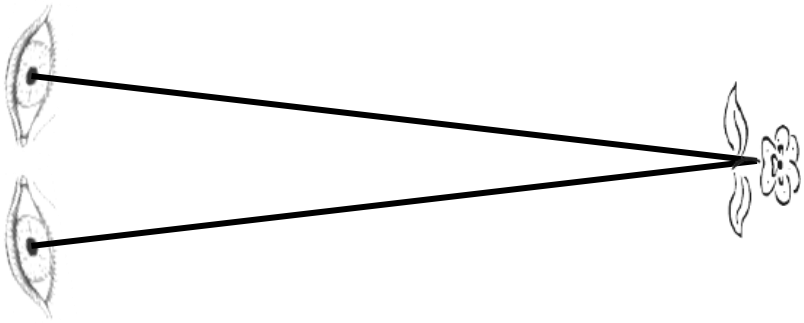


Abbildung 1: Konvergenz zweier Augen, welche eine Blume fixieren

Punkte im Gesichtsfeld, welche bei fester Augenstellung im linken und rechten Auge auf korrespondierende Stellen der Netzhaut abgebildet werden, bilden den Horopter. Definiert man solche Punkte über den Winkel des Sehstrahls zur Blickachse des Auges, erhält man den Vieth-Müller-Kreis (Vgl. Abbildung 2). Das zugehörige Kreissegment im sichtbaren Bereich wird theoretischer Horopter genannt. Empirische Studien haben jedoch gezeigt, dass der gemessene Horopter etwas vom Vieth-Müller-Kreis abweicht. In diesem Zusammenhang spricht man vom empirischen Horopter. Die binokulare Raumwahrnehmung erfolgt in einer Umgebung um den Horopter, dem sogenannten Panum-Areal. In diesem Bereich kommt es zur Fusion der beiden Teilbilder des linken und rechten Auges und damit zu einem räumlichen Eindruck. Liegt ein Objekt ausserhalb des Panum-Areals produziert das Gehirn Doppelbilder, da die Fusion der Teilbilder nicht mehr gelingt. Für einen Punkt ausserhalb des Horopters dient die binokulare Disparität als Mass für die Abweichung von der korrespondierenden Stelle auf der Netzhaut. Diese Abweichung wird als Winkel δ angegeben (Vgl. Abbildung 3). Weitere Informationen zu diesem Thema findet man in der neurologischen Fachliteratur [7, 8].

[1] Der Abstand der Augen beträgt beim Menschen 6-7cm.

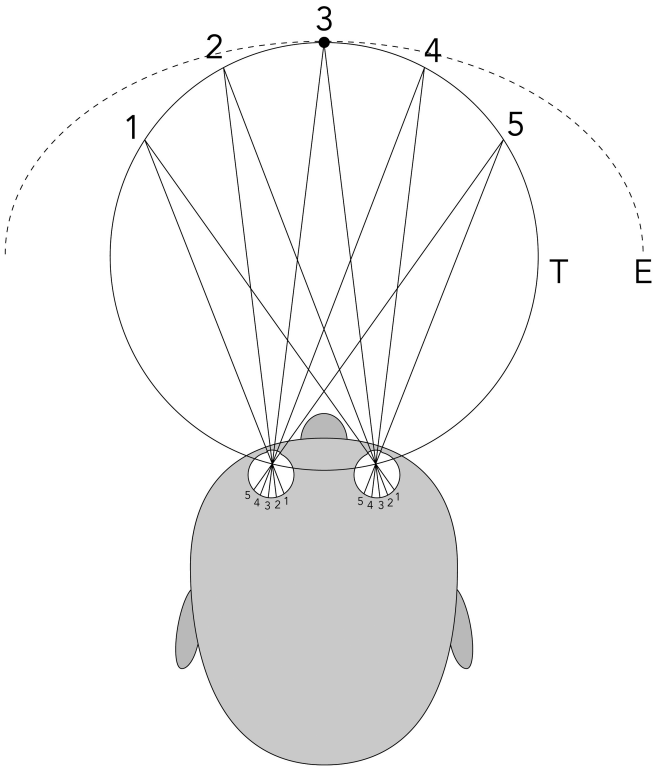


Abbildung 2: Die Augen fixieren den Punkt 3, so dass die Blickachse der Verbindung vom Punkt 3 zu seinem Bildpunkt auf der Netzhaut entspricht. Die übrigen vier Punkte stellen beliebige Punkte auf dem Vieth-Müller-Kreis dar, wobei die Verbindungslinie zwischen diesen Punkten und ihren Bildpunkten auf der Netzhaut bei beiden Augen gleiche Winkel zur Blickachse aufweisen. Die Punkte 1 bis 5 liegen also auf dem theoretischen Horopter (T). Gestrichelt eingezeichnet ist zudem der empirische Horopter (E). http://de.wikipedia.org/

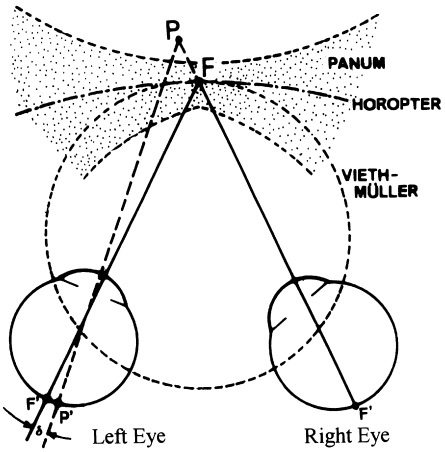


Abbildung 3: Die Abbildung zeigt ein Augenpaar, das den Punkt F fixiert, mit dem empirischen Horopter und dem Vieth-Müller-Kreis sowie dem Panum-Areal (gepunktete Fläche). Für den Punkt P ausserhalb des Horopters ergibt sich auf der Netzhaut des linken Auges die Disparität δ als Winkel zwischen dem Sehstrahl zum Punkt P' und dem Sehstrahl des zum rechten Auge korrespondieren Punktes F'. Aus [7].

Besitzt jemand nur ein Auge oder ist das Zusammenspiel der beiden Augen gestört, fällt die räumliche Wahrnehmung aus (Stereoblindheit). Die folgenden Faktoren ermöglichen aber gemäss [9, 10] auch in diesem Fall eine begrenzte räumliche Wahrnehmung:

- Akkommodation: Das Auge passt sich an unterschiedlichen Entfernungen an.
- Bewegungsparallaxe: Bewegt sich der Betrachter relativ zu Objekten mit unterschiedlicher Entfernung (beispielsweise wenn man durch das Fenster eines fahrenden Zuges schaut), so scheinen sich nahe gelegene Objekte schneller zu bewegen.
- Interposition: Näher gelegene Objekte verdecken Teile der weiter entfernten Objekte.
- Licht und Schatten: Fällt der Schatten eines Objektes auf ein anderes Objekt, muss das zweite Objekt weiter von der Lichtquelle entfernt sein.
- Perspektive: Nahe gelegene Objekte erscheinen grösser als weiter entfernte Objekte.
- Lufttrübung: In grosser Entfernung erscheinen die Objekte infolge der athmosphärischen Dämpfung durch Dunst trüb.
- Kopfbewegung: Bewegt der Betrachter den Kopf, erhält er stets neue Perspektiven der betrachteten Objekte.

Bei der Erforschung des räumlichen Sehens erfand Béla Julesz das Random Dot Stereogramm [11]. Daraus entwickelten sich in den 1980er Jahren die bekannten Autostereogramme, wie beispielsweise das Single Image Random Dot Stereogram (Vgl. Abbildung 4).

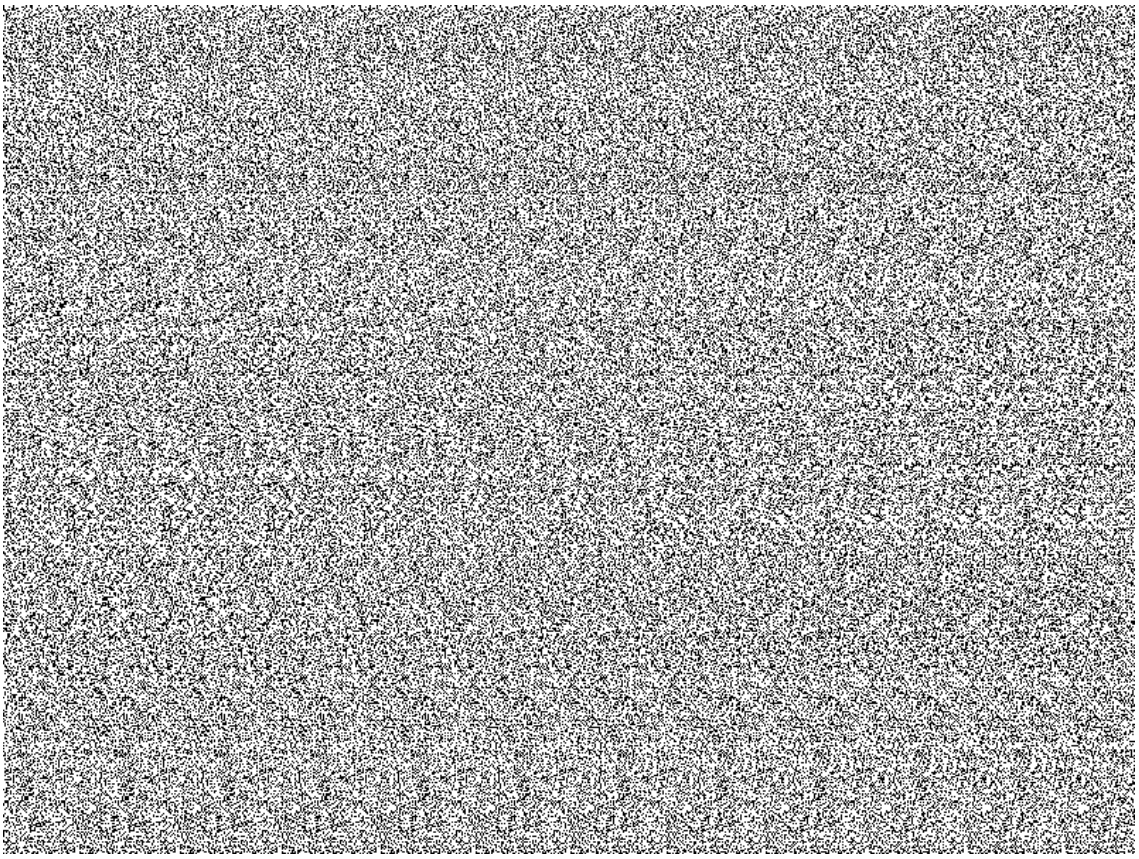


Abbildung 4: Single Image Random Dot Stereogram (SIRDS). Wenn der Betrachter seinen Blick ins Unendliche richtet, erkennt er auf diesem SIRDS den Schriftzug "3D".
http://upload.wikimedia.org/wikipedia/commons/c/c5/Mh_stereogramm_sirds.png

2.2 Stereoskopie

Bei der Stereoskopie wird die räumliche Wirkung einer Szene mit zwei verschiedenen Perspektiven erzeugt, welche den Positionen des linken und rechten Auges eines menschlichen Betrachters entsprechen. Dadurch entsprechen viele der im letzten Abschnitt beschriebenen Faktoren der realen Raumwahrnehmung. Wie wir später sehen werden, gilt dies allerdings nicht für alle Faktoren. Um die Entwicklung der Stereoskopie aufzuzeigen, wird zunächst die Geschichte des stereoskopischen Films kurz rekapituliert.

2.2.1 Der stereoskopische Film

Die Wurzeln des stereoskopischen Films reichen bis ins 19. Jahrhundert zurück. Bereits 1833 konstruierte der englische Physiker Charles Wheatstone ein Spiegelstereoskop, mit dem man beispielsweise Fotos der ersten stereoskopischen Zweiobjektiv-Kameras betrachten konnte (Vgl. Abbildung 5). In den 50er Jahren des letzten Jahrhunderts boomte der stereoskopische

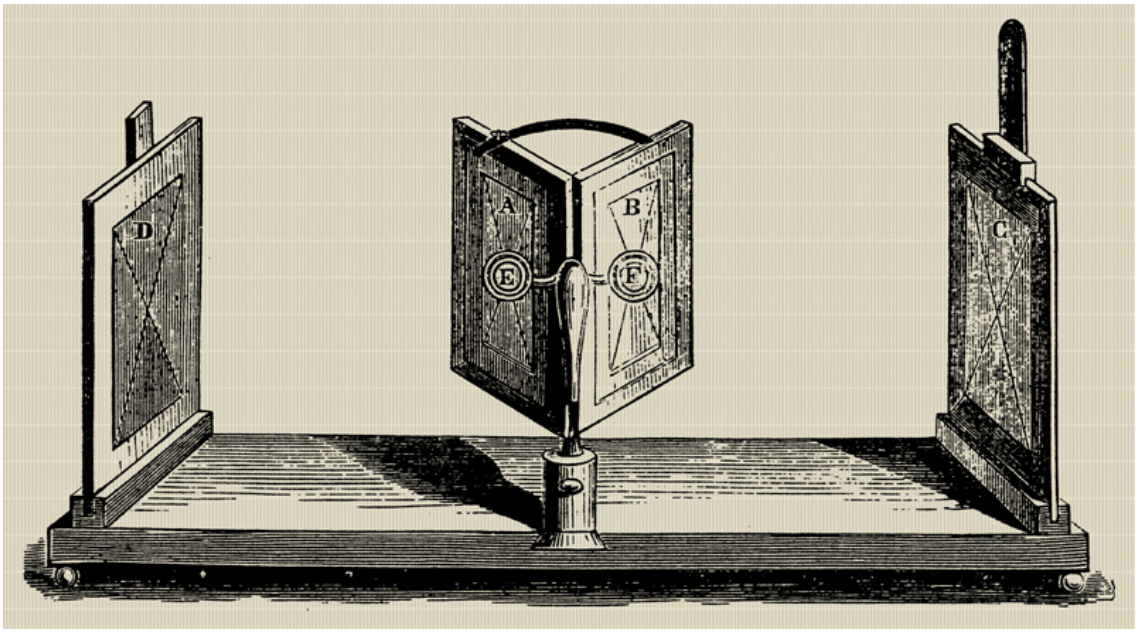


Abbildung 5: Das Spiegelstereoskop von Charles Wheatstone zeigt dem Betrachter mithilfe der Spiegel in der Mitte eine stereoskopische Ansicht der beiden Teilbilder links und rechts. http://wikimediafoundation.org/

3D-Film bereits ähnlich wie heute. 1952 wurden 3D-Filme basierend auf der Polarisationsfiltertechnik im Kino gezeigt (Vgl. Abbildung 6). Da die Herstellung der Filme jedoch technisch sehr aufwändig war, konnte sich die Technologie nicht durchsetzen. Ausserdem machten damals Kopfbewegungen der Zuschauer den 3D-Effekt zunichte oder führten zu Geisterbildern und im schlimmsten Fall gar zu Kopfschmerzen. Der eigentliche Durchbruch gelang erst James Cameron mit dem S3D-Film[2] Avatar im Jahr 2009. Entscheidend für den Erfolg war dabei vor allem die digitale Aufnahme- und Wiedergabetechnik, welche eine kostengünstigere Produktion und Präsentation der S3D-Filme ermöglichte. Zudem bleibt bei modernen S3D-Filme der 3D-Effekt auch bei Kopfbewegungen erhalten.

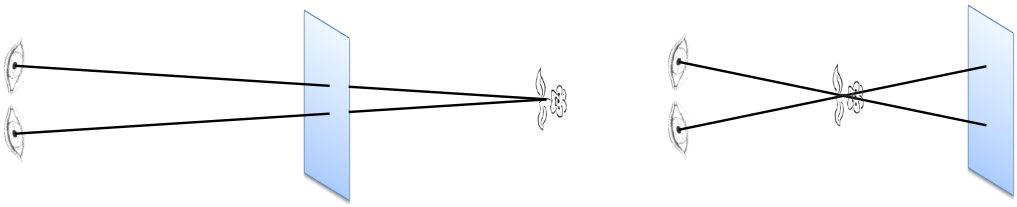
[2]Ein S3D-Film ist ein stereoskopischer 3D-Film.

Abbildung 6: Kinopublikum mit Polarisationsfilterbrillen aus den 50er Jahren des letzten Jahrhunderts, http://gizmologia.com/files/2009/11/3D-publico.jpg

2.2.2 Limitierende Faktoren

Akkomodation Das Hauptproblem der Stereoskopie besteht darin, dass der Betrachter stets auf das stereoskopische Display fokussieren muss, um ein scharfes Bild zu erhalten. Dies verletzt den im letzten Abschnitt erwähnten Faktor der Akkommodation, da die abgebildete räumliche Szene auch Objekte enthält, welche räumlich vor oder hinter dem Display liegen würden. Hier widerspricht die stets konstante Akkommodation des stereoskopischen Bildes der Erfahrung des Betrachters, ein Widerspruch, der bei einigen Menschen zu Kopfschmerzen oder Übelkeit führt (Vgl. auch [12, 9]).

Horizontale Parallaxe Objekte, welche hinter der Projektionsebene liegen, erscheinen für das linke Auge weiter links und für das rechte Auge weiter rechts. Dieses Phänomen heisst (positive) Parallaxe (Vgl. Abbildung 7(a)). Die zugehörige Abweichung auf der Netzhaut



(a) positive Parallaxe (b) negative Parallaxe

Abbildung 7: Objekte ausserhalb der Projektionsebene erscheinen infolge des Augenabstandes versetzt. Man spricht von einer Parallaxe.

gemessen als Winkel heisst ungekreuzte Disparität. Objekte, welche vor der Projektionsebene liegen, erscheinen hingegen für das linke Auge weiter rechts und für das rechte Auge weiter links. In diesem Fall spricht man von negativer Parallaxe (Vgl. Abbildung 7(b)), bzw. von gekreuzter Disparität. Für Objekte in der Projektionsebene tritt keine Parallaxe auf und die Disparität ist 0. Man spricht dann von der Nullparallaxe (engl. zero parallax). Wird die Disparität zu gross, erscheint dem Betrachter ein sogenanntes Geisterbild, weil die Fusion der beiden Teilbilder zu einer Raumwahrnehmung nicht mehr gelingt. Mit empirischen Testreihen wurden in [13] die Grenzwinkel für die binokulare Disparität ermittelt, bis zu denen die Fusion der Teilbilder bei einem durchschnittlichen Betrachter gelingen sollte (Vgl. Tabelle 1). Weitere

Informationen zu diesem Thema findet man in [14, 15].

Parallaxe	Rot	Weiss	Mittelwert
negativ (gekreuzte D.)	6.19°	3.66°	4.93°
positiv (ungekreuzte D.)	1.95°	1.20°	1.57°

Tabelle 1: Grenzwinkel der binokularen Disparität, unter dem die Fusion der beiden Teilbilder gerade noch gelingt. Die Tabelle enthält Disparitäten für rote und weisse Stimuli sowie einen Mittelwert über alle Tests.

Kopfbewegungen Ausserdem liefern Kopfbewegungen bei stereoskopischen Bildern ebenfalls nicht den vom Betrachter erwarteten Effekt, da ja nur zwei Ansichten der Szene vorliegen. Zur Lösung dieses Problems wird derzeit mit so genannten Multiview Displays experimentiert, die mehr als zwei parallele Ansichten verarbeiten können.

2.3 3D-Hardware

2.3.1 Stereoskopische Displays

Wir unterscheiden die folgenden vier Kategorien je nach Art des Displays: stereoskopische Displays mit Brillen, autostereoskopische Displays, head-mounted Displays und CAVEs (Vgl. Abbildung 8). Da derzeit die Displays mit Brillen am weitesten verbreitet sind, konzentrieren

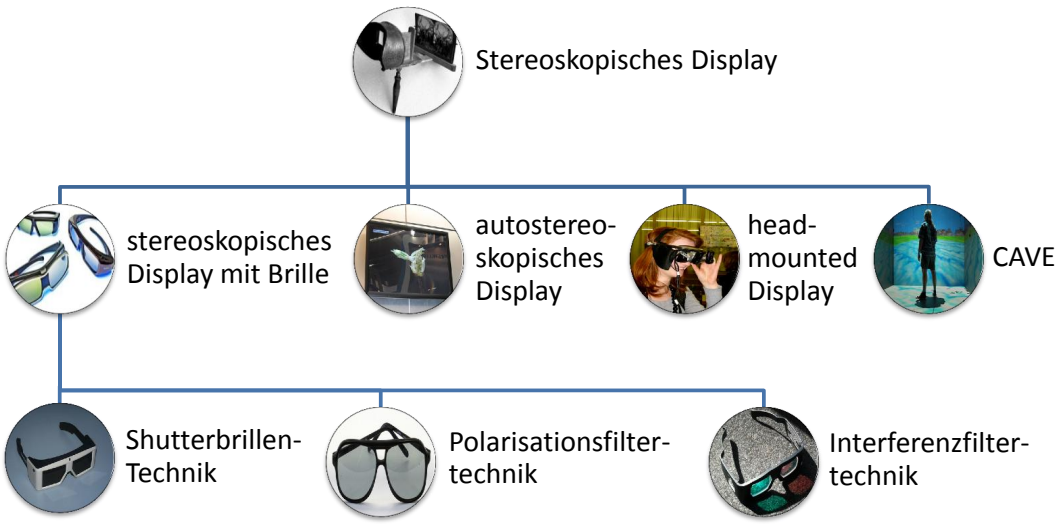


Abbildung 8: Stereoskopische Displays im Überblick

wir uns im Folgenden auf Vertreter dieser Kategorie. Bei den Schweizer Kinos waren im August 2010 die Shutterbrillentechnik und die Polarisationsfiltertechnik mit Anteilen von je ca. einem Drittel ungefähr gleich stark verbreitet. Die Interferenzfiltertechnik fiel mit einem Anteil von einem Sechstel etwas ab. Von den übrigen 3D-Kinos waren leider noch keine technischen Daten greifbar (Vgl. Anhang A).

Die Shutterbrillen-Technik Shutterbrillen besitzen zwei aktiv steuerbare LCD-Schirme, welche abwechselnd das linke und das rechte Auge abdunkeln. Das zugehörige Display zeigt dabei jeweils das Bild für das linke, bzw. rechte Auge und sendet ein entsprechendes Signal an

die Brille. Displays, welche die Shutterbrillen-Technik unterstützen, müssen demnach Taktfrequenzen von über 100 Hz aufweisen, da das Auge bei weniger als 50 Bildern pro Sekunde ein Flimmern wahrnehmen würde. Auf der Beamerseite bieten sich daher DLP-Projektoren[3] an, weil mit dieser Technologie hohe Taktfrequenzen leichter zu realisieren sind. Auch auf dem Monitor-Markt ist die Shutterbrillen-Technik weit verbreitet, obwohl die aktive Shutterbrille verglichen mit einer Polarisationsfilterbrille recht teuer ist. Da zu einem bestimmten Zeitpunkt nur ein Teilbild zu sehen ist, kann das Display dieses Teilbild in maximaler Auflösung präsentieren. Diese Methode heisst Vollbildverfahren (engl. **p**rogressive). 1080p steht also für die volle HDTV-Auflösung[4] von 1920×1080 Pixeln. Dies ist, wie wir gleich sehen werden, ein Vorteil gegenüber der Polarisationsfiltertechnik. Andererseits führen die relativ langen Dunkelphasen beim Wechseln vom linken zum rechten Teilbild zu einer Abdunklung des Filmes um über 50%. Daher werden Displays mit hoher Leuchtkraft benötigt, also beispielsweise LED-Monitore oder DLP-Projektoren.

Polarisationsfiltertechnik Im Gegensatz dazu wird bei der Polarisationsfiltertechnik eine passive Brille benötigt, welche mit zwei Polarisationsfiltern bestückt ist. Diese lassen nur polarisiertes Licht mit bestimmten Eigenschaften durch. Derzeit benützen die meisten Hersteller zirkulare Polarisation, bei der sich die Auslenkung spiralförmig um den Wellenvektor dreht (Vgl. Abbildung 9). Dabei wird das Teilbild für das linke Auge beispielsweise linksdrehend po-

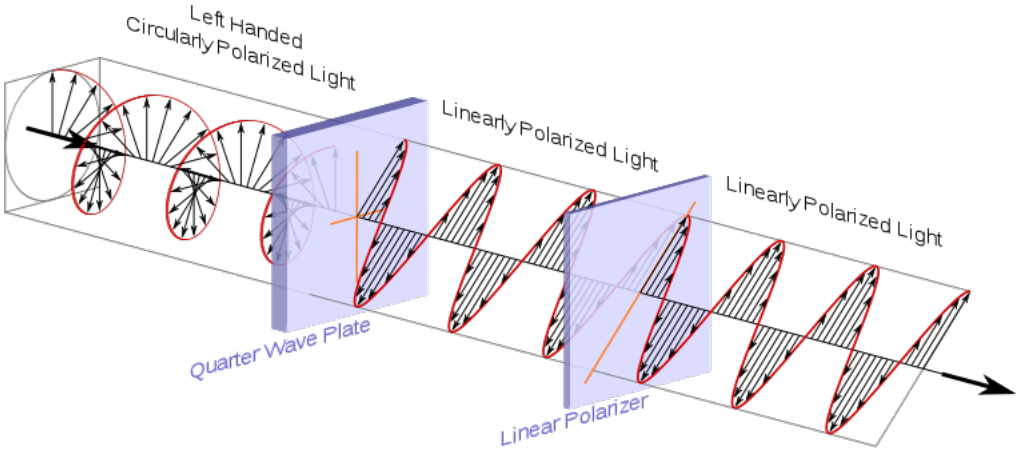


Abbildung 9: Das zirkular polarisierte Licht wird zuerst durch einen λ/4-Verzögerungsfilter (engl. quater wave polarizer) in linear polarisiertes Licht umgewandelt. Anschliessend werden durch den linearen Polarisationsfilter (engl. linear polarizer) alle anderen Lichtanteile ausgefiltert. http://de.wikipedia.org/wiki/3D-Brille

larisiert und jenes für das rechte Auge rechtsdrehend oder umgekehrt. Dies hat den Vorteil, dass selbst bei Kopfbewegungen des Zuschauers der 3D-Effekt erhalten bleibt. Da Polarisationsfilterbrillen relativ günstig herzustellen sind, ist diese Technik in den Kinos inzwischen weit verbreitet. Neben einem Projektor mit zwei Polarisationsfiltern wird dabei eine metallische Leinwand benötigt, da bei einer weissen Leinwand die Polarisation des Lichts verloren gehen würde. Obwohl die grossen Kinoausstatter Silberleinwände anbieten, reicht gemäss [16] auch eine mit Aluminium-Spray besprühte herkömmliche Leinwand für zufriedenstellende Resultate aus. Monitore mit Polarisationsfiltertechnik haben den Nachteil, dass das Bild notwendigerweise in zwei Halbbilder zerfällt, weil jeweils die Hälfte der Pixel gleich polarisiert werden muss. Dies wird beispielsweise mit dem Zeilensprungverfahren (engl. **i**nterlaced) gelöst, bei dem zeilenweise zwischen dem linken und dem rechten Halbbild abgewechselt wird. Eine Auf-

[3] DLP steht für **D**igital **L**ight **P**rocessing.
[4] HDTV steht für **H**igh **D**efinition **Tele**vision.

lösung von 1080i bedeutet demnach dass von den 1920×1080 Pixeln des HDTV-Formats für jedes Auge nur die Hälfte nämlich 1920×540 Pixel gebraucht werden, da immer nur ein Halbbild angezeigt werden kann.

Interferenzfiltertechnik Ein Interferenzfilter lässt Licht mit bestimmten Wellenlängen durch und filtert je nach Konstruktion auch dicht benachbarte Wellenlängen aus. Dazu werden Interferenzen zwischen den direkt einfallenden und den reflektierten Lichtstrahlen ausgenutzt. Wenn man für die Grundfarben im rechten Teilfilm andere Wellenlängen verwendet als für die Grundfarben im linken Teilfilm, lassen sich die beiden Teilfilme mit einer Interferenzfilterbrille ausfiltern. Damit dies nicht zu einer veränderten Farbwahrnehmung führt, müssen die Teilfilme beispielsweise im Projektor vorbearbeitet werden. Interferenzfilterbrillen sind zwar verglichen mit Polarisationsfilterbrillen relativ teuer. Dafür kann für die Präsentation auch eine herkömmliche weisse Leinwand eingesetzt werden.

Autostereoskopische Displays Der Nachteil der obigen Verfahren ist offensichtlich: Sie setzen voraus, dass der Zuschauer eine Brille aufsetzt. Dies ist nicht nur für Brillenträger mühsam und wird vor allem auch im Heimbereich als lästig empfunden. Daher sind inzwischen auch sogenannte autostereoskopische Displays auf dem TV Markt erhältlich. Diese kommen, wie der Name sagt, ohne Brillen aus, erzeugen die stereoskopische Wahrnehmung also von selber. Bereits um 1940 wurde dieser Effekt in einem Moskauer Kino mithilfe eines Metallgitters erreicht. Heute sind Monitore im Umlauf, welche mit so genannten Barrierefiltern arbeiten. Steht dabei der Betrachter am richtigen Platz, sieht das linke Auge infolge der Barriere ein anderes Halbbild als das rechte Auge (Vgl. Abbildung 10). Ähnlich funktionieren

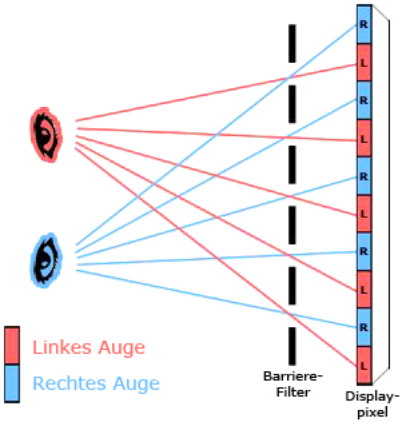


Abbildung 10: Barrierefilter, welcher die Pixel für das jeweils andere Auge ausfiltert, sofern der Betrachter am richtigen Platz steht. http://www.gizmag.com

auch Monitore mit Lentikularlinsen, wobei hier die Linsen die Teilbilder zum jeweiligen Auge ablenken. Eine detaillierte Beschreibung autostereoskopischer Techniken findet man in [17]. Ein aktiver Forschungsgegenstand der autostereoskopischen Displaytechnik sind Multiview Displays, welche mehr als zwei Ansichten anzeigen können. Wie in [18] beschrieben, werden dabei sogenannte Head-tracking Komponenten eingesetzt, welche den Blickwinkel des Betrachters anhand der Kopfstellung ermitteln sollen.

Head-Mounted Display Ein Head-Mounted Display (HMD) ist ein auf dem Kopf befestigtes Ausgabegerät, welches die beiden Augen meist über zwei Displays mit Bildern versorgt. Auf diese Weise lassen sich relativ leicht stereoskopische Bilder direkt zu den beiden Augen

übertragen. Wie das Patent in [19] zeigt, ist es mittlerweile möglich, durch geeignete Vorrichtungen den Horopter der Augen zu beeinflussen und damit dem Betrachter die Akkomodation zu erleichtern.

CAVE CAVE steht für **C**ave **A**utomatic **V**irtual **E**nvironment und bedeutet wörtlich übersetzt Höhle mit automatischer virtueller Umgebung. Gemeint ist also ein Raum, in dem beispielsweise mit Shutterbrillen-Technologie eine 3D-Szene künstlich generiert wird. Dabei werden die vier Wände und allenfalls auch der Boden oder die Decke des Raumes als Projektionsflächen eingesetzt. Eine sehr umfangreiche Anwendung eines CAVE in einem Tool zur Visualisierung von Molekülen findet der interessierte Leser in [1].

2.3.2 Blu-ray und Blu-ray 3D Discs

Die Blu-ray Disc (BD) ist genau wie die CD und die DVD ein optischer Datenträger, der im Lesegerät, dem Blu-ray Player, von einem Laserstrahl abgetastet wird. Gegenüber der CD und der DVD zeichnet sich die BD vor allem durch die viel höhere Übertragungsrate von ca. 60 Mbit/s aus. Neben einer optimierten Optik liegt dies vor allem am Material der Scheibe, welches deutlich härter ist als das Polycarbonat der CD. Dies erlaubt höhere Drehzahlen im Player. Handelsübliche Blu-ray Discs fassen im Single Layer Format 25 GB und im Double Layer Format 50 GB. Unter Laborbedingungen wurde aber bereits mit Kapazitäten bis 500 GB (20 Layer) experimentiert. Als Dateiformat wird das MPEG-2TS Format verwendet, ein Transportformat, das bisher vor allem beim digitalen Broadcasting eingesetzt wurde. Abbildung 11 zeigt die Dateistruktur einer Blu-ray Disc. Da die Schutzschicht einer Blu-ray Disc auf der Laserseite deutlich dünner ist als bei der DVD, sind die BDs mit einer speziellen Schicht namens Durabis[5] beschichtet. Technische Details können dem White Paper [20] der Blu-ray Disc Founders entnommen werden. Aufgrund ihrer grossen Kapazität eignen sich

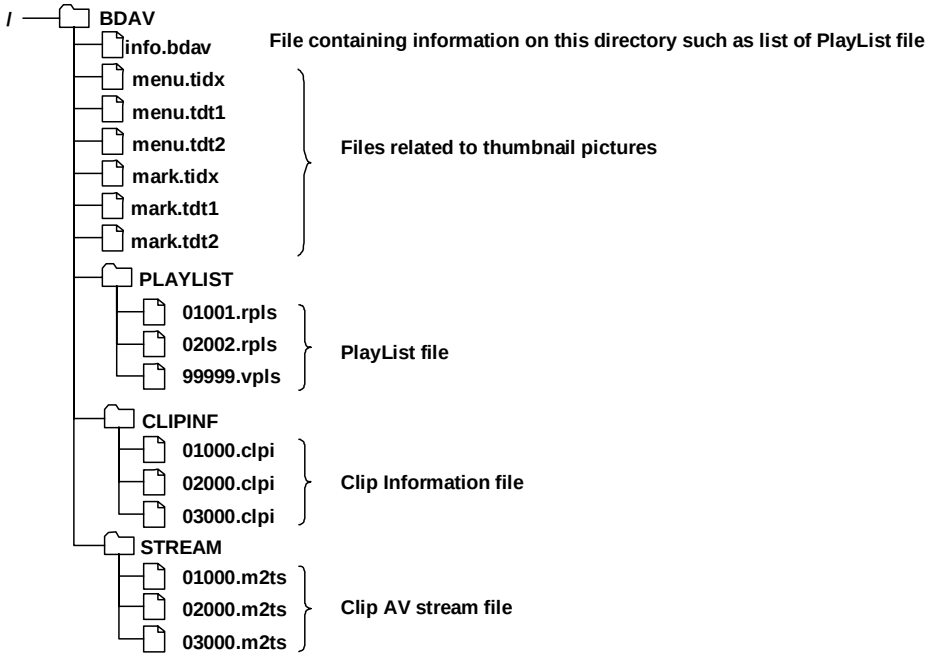


Abbildung 11: Die Abbildung zeigt die Dateistruktur einer Blu-ray Disc mit den MPEG-2TS-Videostreamdateien. Aus [20], Seite 19.

[5] Durabis kommt aus dem Lateinischen und bedeutet "du wirst bestehen".

Blu-ray Discs gut für das oben beschriebene Full HD Video-Format von 1920x1080 Pixeln. Enthält die Blu-ray Disc einen S3D-Film im MPEG-4 MVC Format[6], so spricht man von einer Blu-ray 3D Disc. Diese unterscheidet sich äusserlich nicht von einer normalen Blu-ray Disc. Allerdings muss der Blu-ray Player in der Lage sein, das MPEG-4 MVC Format zu dekodieren und den stereoskopischen Videostream an ein 3D-fähiges Display zu übermitteln. Dazu dient die Videoschnittstelle HDMI 1.4.

2.3.3 Die Videoschnittstelle HDMI 1.4

HDMI steht für **H**igh-**D**efinition **M**ultimedia **I**nterface und bezeichnet eine Schnittstelle zur Übertragung von Audio- und Videodaten. S3D-Videos werden ab HDMI 1.3 unterstützt. Mit einem Trick konnten DLP fähige TV Geräte und Beamer S3D-Filme sogar schon in HD Qualität präsentieren. Dazu wurde das sogenannte Schachbrettmuster (engl. checkerboard) eingesetzt, mit dem zwei Halbbilder im Rahmen eines Vollbilds übertragen werden konnten (Vgl. Abbildung 12).

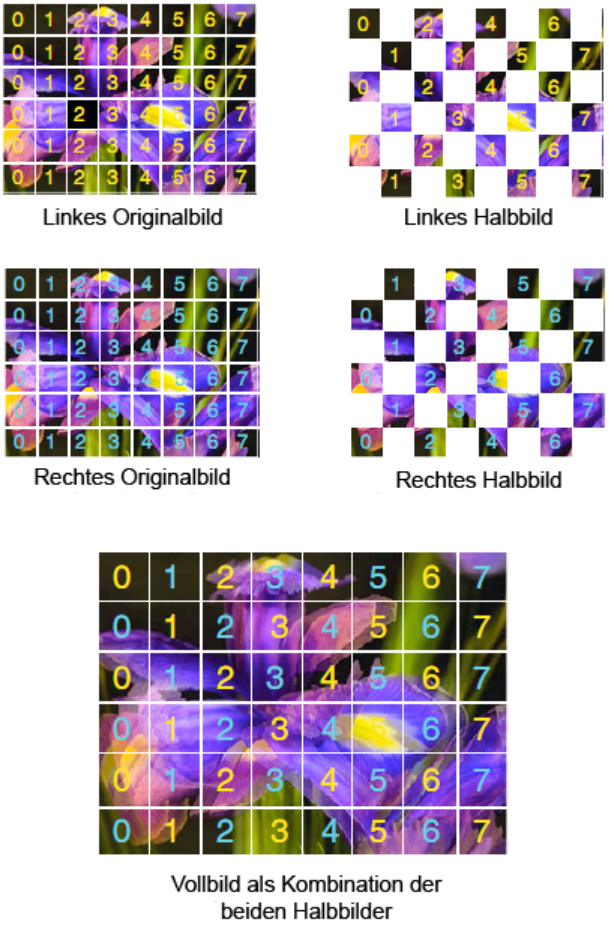


Abbildung 12: Die Abbildung zeigt, wie aus zwei Vollbildern für das linke und das rechte Auge je ein reduziertes Halbbild berechnet wird. Diese beiden Halbbilder werden hinterher schachbrettartig zu einem Vollbild zusammengesetzt. http://www.mtbs3d.com/

Mit HDMI 1.4 sind solche Tricks jedoch nicht mehr nötig, da nun die beiden Frames für das linke und das rechte Auge als Vollbilder übermittelt werden können. Dies geschieht mit dem so genannten Frame-Packing-Verfahren, bei dem die beiden HD Frames für das linke

[6] Details zum MPEG-4 MVC Format folgen im Abschnitt 2.4.1

und das rechte Auge in einen einzigen Frame gepackt werden. Die beiden Frames werden übereinander angeordnet und durch einen 45 Pixel breiten Streifen getrennt. Dies ergibt einen Megaframe der Auflösung 1920×2205 (Vgl. Abbildung 13). Diese Megaframes werden mit einer Übertragungsrate (engl. framerate) von 24 Frames pro Sekunde über das HDMI-Kabel an ein 3D-fähiges Gerät übermittelt. Der linke Frame liegt dabei oben und garantiert die

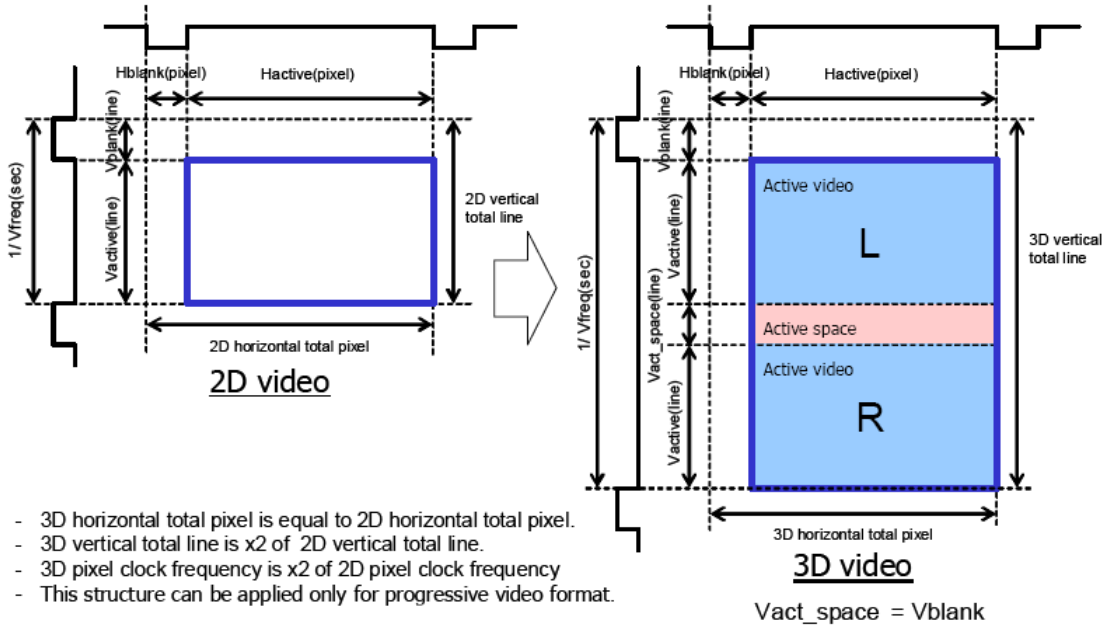


Abbildung 13: Die Abbildung zeigt, wie die beiden Frames zu einem Megaframe gepackt werden. Aus [21], Seite 8.

Rückwärtskompatibilität mit Displaygeräten, welche noch keine 3D-Funktionalität aufweisen.

Neben dem Frame-Packing-Verfahren unterstützt HDMI 1.4 auch das Side-by-Side Verfahren, bei dem die beiden Teilframes auf die halbe Breite reduziert und nebeneinander in einem Full HD Frame der Auflösung 1920×1080 abgelegt werden. Ausserdem wird auch das Top-and-Bottom Verfahren unterstützt, bei dem die beiden reduzierten Teilframes übereinander in einem Full HD Frame abgelegt werden.

2.4 Videokompression

2.4.1 H.264 und MPEG-4

Der Videokompressionstandard H.264 der ITU[7] gehört unter der Bezeichnung MPEG-4 Advanced Video Coding (AVC)[8] neben MPEG-2 und VC-1 zu den festen Bestandteilen des Blu-ray Disc Standards. Zudem ist das MPEG-4 Containerformat[9] bereits für den Video-Tag von HTML5 vorgesehen. Seit Quicktime 7 ist H.264 auch Teil des Multimedia-Frameworks von Apple (Vgl. Anhang B.5). Das x264-Team bietet auf [22] eine plattformunabhängige Open Source Implementierung von H.264 an. Auch der freie MPEG-4-Video-Codec Xvid [23] unterstützt Advanced Video Coding. Für die Blu-ray 3D Discs musste der MPEG-4

[7] ITU steht für **I**nternational **T**elecommunication **U**nion. Die ITU ist eine Sonderorganisation der Vereinten Nationen, welche sich mit technischen Aspekten im Telekommunikationsbereich befasst. Innerhalb der ITU ist die ITU-T für Standardisierungen zuständig. Der Teilbreich H befasst sich mit audiovisueller Kommunikation.

[8] MPEG steht für **M**oving **P**icture **E**xperts **G**roup, eine Expertengruppe, welche sich mit der Standardisierung von Videokompressionsverfahren beschäftigt.

[9] Mit Containerformat bezeichnet man ein Dateiformat, das verschiedene Datenformate (z.B. Videocodecs) enthalten kann. Das MPEG-4 Containerformat besitzt die Endung .mp4.

AVC Standard ergänzt werden. Dies geschah 2008 mit dem Multiview Video Coding (MVC), welches die Kodierung von S3D-Filmen im Rahmen von MPEG-4 regelt. Dabei werden die Frames für das linke Auge wie gewohnt mit MPEG-4 AVC kodiert. Für das rechte Auge werden hingegen nur die Differenzinformationen zum Frame für das linke Auge gespeichert. Dies ermöglicht in der Regel eine starke Reduktion der Dateigrösse (Vgl. Abbildung 14). Details zu H.264 AVC und MVC sind in [24] zu finden.

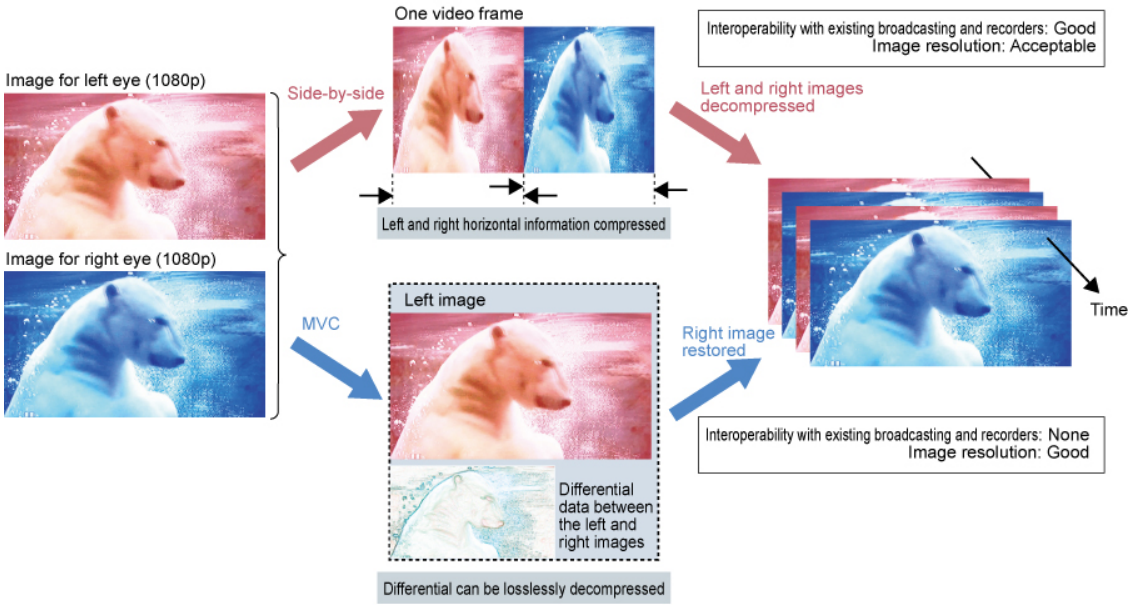


Abbildung 14: Beim Side-by-Side Format werden die beiden Frames zu Halbbildern reduziert und in einem Frame abgelegt. Die MVC Kompression speichert den linke Frame als Vollbild und als Erweiterung die Differenzinformationen des rechten Frames zum linken Frame. http://3dcinecast.blogspot.com/

2.4.2 webM

Das Containerformat webM beherbergt den Videocodec VP8, der von Google gekauft und anschliessend der Open Source Community offengelegt wurde. Genau wie MPEG-4 ist auch webM für den Video-Tag von HTML5 vorgesehen und stellt somit eine interessante Alternative für Webvideos dar. Die Kompressionsrate von VP8 ist laut ersten Tests vergleichbar mit jener von H.264-Codecs. Die folgenden Webbrowser unterstützen derzeit bereits Videos im webM-Format:

- Mozilla Firefox 4
- Opera 10.60
- Google Chrome 6

Offensichtlich fehlen auf dieser Liste der Safari-Browser und der Internet Explorer. Dies wird sich wohl mit der Einführung von HTML5 ändern, zumal YouTube bereits einen HTML5 Testmodus betreibt, in dem auch webM-Videos abgespielt werden können. Ausserdem kann man durch Installation von DirectShow-Filtern[10] unter Windows und Quicktime-Komponenten[11] unter Mac OS X dafür sorgen, dass webM-Videos auch auf dem systemeigenen Videoplayer abgespielt und im systemeigenen Editor bearbeitet werden können. Weitere Informationen zum webM-Format findet man in [25].

[10] Siehe http://xiph.org/dshow/.
[11] Siehe http://code.google.com/p/webm/downloads/list.

2.5 3D-Computergrafik

S3D-Animationsvideos basieren im Gegensatz zum mit einer 3D-Kamera gefilmten Video auf einer virtuellen Szene. Diese Szene muss vorgängig im Computer modelliert werden. Geht es darum, detailgetreue Animationsvideos zu erstellen, werden die Computergrafiken in der Regel mittels Raytracing gerendert.

2.5.1 Raytracing

Beim Raytracing werden ausgehend vom virtuellen Augpunkt Sehstrahlen durch sämtliche Pixel der Bildebene geschickt. Diese Sehstrahlen werden rekursiv durch die virtuelle Szene verfolgt, wobei die Gesetze der Strahlenoptik (Reflexion, Brechung, etc.) berücksichtigt werden (Vgl. Abbildung 15). Diese Technik ist zwar in der Anwendung sehr zeitintensiv, ermöglicht

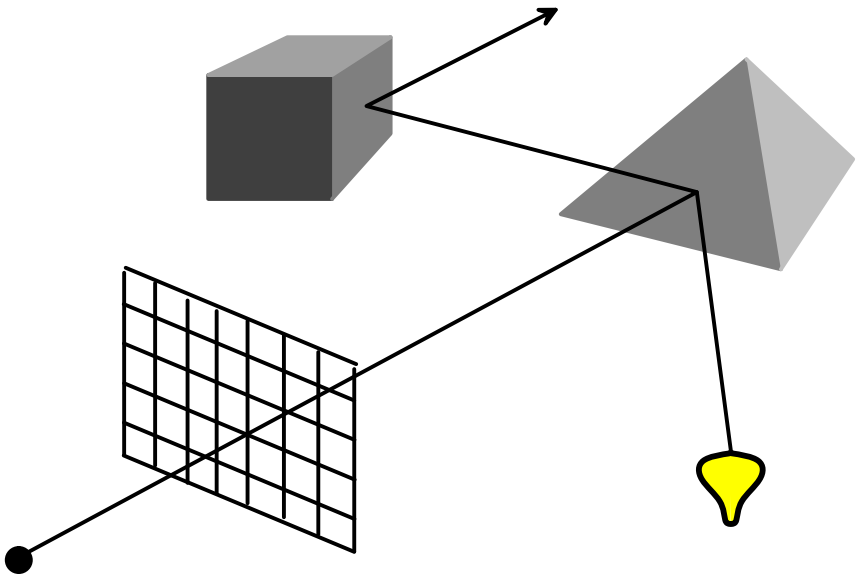
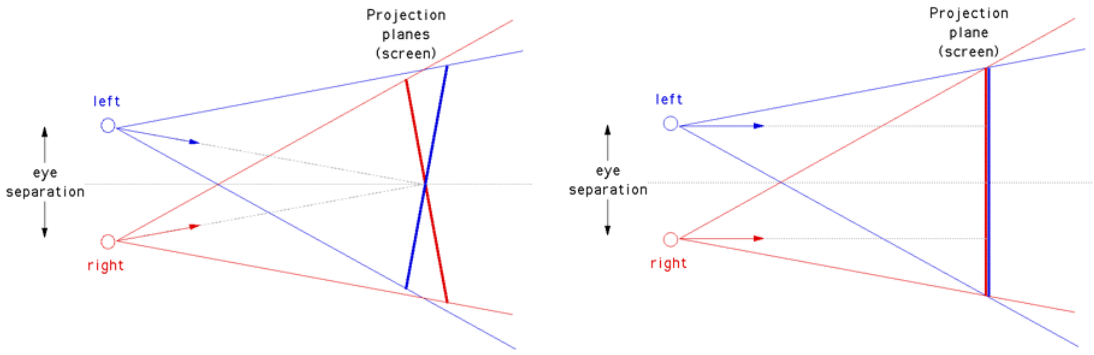


Abbildung 15: Rekursives Ray-Tracing für realistischere Spiegelreflexion aus [9], S. 248

aber mittlerweile nahezu fotorealistische Bilder. Eine detaillierte Beschreibung verschiedener Rendering Techniken findet man beispielsweise in [9].

2.5.2 Stereoskopische Graphiken

Für stereoskopische Graphiken benötigt man zwei Kamerapositionen, welche den Augenpositionen des menschlichen Betrachters entsprechen. Die erste Frage, die sich dabei stellt, ist, wie diese beiden virtuellen Kameras ausgerichtet werden sollen. Eine sehr detaillierte und in diesem Zusammenhang oft zitierte Anleitung dazu findet man in [12]. Der Vollständigkeit halber werden die beiden wichtigsten Einstellungen hier kurz vorgestellt.



(a) Toe-in Verfahren (b) Off-axis Verfahren

Abbildung 16: Verfahren zur Ausrichtung der stereoskopischen Kamera nach [12].

Richtet man die Kameras durch eine Drehung nach Innen (engl. toe-in) auf einen gemeinsamen Fokuspunkt (Vgl. Abbildung 16(a)), so kommt es zusätzlich zur bereits beschriebenen horizontalen Parallaxe noch zu einer vertikalen Parallaxe, welche von der Bildmitte nach aussen hin zunimmt (Vgl. Abbildung 17). Dies gilt es zu vermeiden. Dazu dient das Off-axis Verfahren. Die Projektionsrichtung der beiden Kameras ist dieselbe, da beide senkrecht auf die Projektionsebene gerichtet sind. Damit entfällt die vertikale Parallaxe. Trotzdem können die beiden Bildbereiche zur Deckung gebracht werden, wenn man darauf verzichtet, dass der senkrecht zur Projektionsebene verlaufende Projektionsstrahl durch die Bildmitte geht (Vgl. Abbildung 16(b)). Dadurch entsteht ein asymmetrisches Frustum[12]. Für den Augenabstand

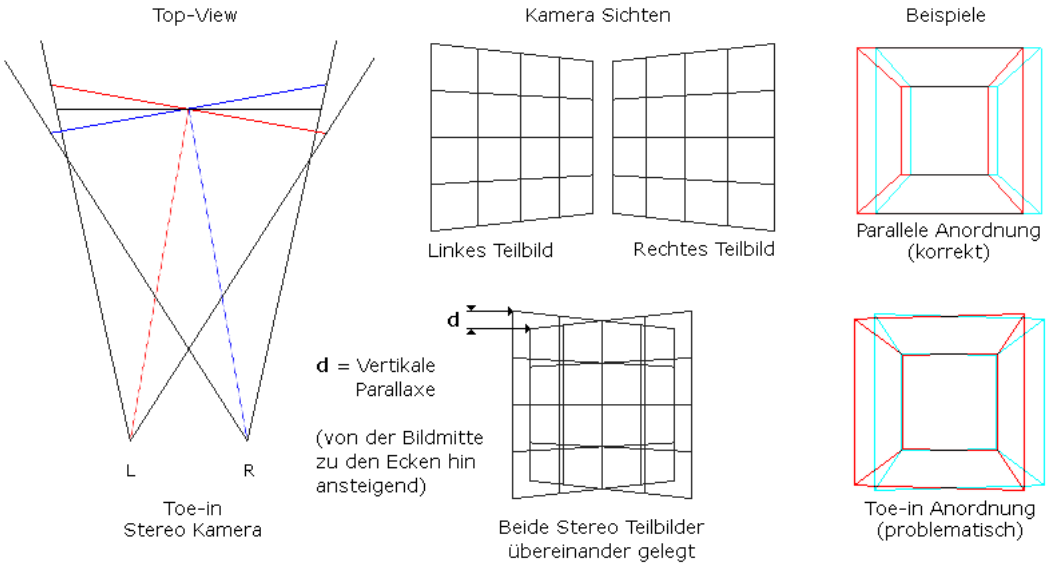


Abbildung 17: Vertikale Parallaxe beim Toe-in Verfahren, http://noeol.de/s3d/

könnte man theoretisch den anatomischen Mittelwert von 6.5cm wählen. Da die Einheiten in einer virtuellen Szene in der Regel nach anderen Kriterien gewählt werden, macht dies meist keinen Sinn. Das folgende Gedankenexperiment aus [26] macht die gängige Faustregel plausibel, nach der man für den Augenabstand gerade einen Dreissigstel des Abstandes zum Fokuspunkt wählen soll.

[12] Frustum bezeichnet in der Computergrafik den Pyramidenstumpf der perspektivischen Projektion als Teil der Sehpyramide. Ist die Sehpyramide schief, spricht man von einem asymmetrischen Frustum.

> "Why 1/30? If you stand in front of a window, which opens to a landscape to the horizon, you will notice that you cannot see clearly both the horizon AND the window itself if you stand within two meters away from the window.
>
> When you are two or more meters away from the window, you can view all the scene comfortably from the nearest point (the window) to infinity (the horizon). This value of two meters depends on the person but is a statistical value. The fact is that 6.5 cm (inter-ocular distance) is about 1/30 of two meters."

Mit den beiden Grenzwinkeln aus Tabelle 1 auf Seite 8 können wir dies gemäss [14] noch etwas genauer anschauen. Dort werden die folgenden Formeln für den kleinsten Abstand d_{min} und den grössten Abstand d_{max} eines Objektes von der Kameraposition beschrieben, sodass dem Betrachter die Fusion für das Objekt gerade noch gelingt.

$$d_{\text{min}} = \frac{e}{2} \cdot \tan\left(\arctan\frac{2f}{e} - \frac{\delta_{\text{crossed}}}{2}\right) \quad (1)$$

$$d_{\text{max}} = \frac{e}{2} \cdot \tan\left(\arctan\frac{2f}{e} + \frac{\delta_{\text{uncrossed}}}{2}\right) \quad (2)$$

Gibt man hier für den Augenabstand e gerade $6.5cm$ ein und für den Abstand f zum Fokuspunkt das Dreissigfache, so erhält man mit den Grenzwinkeln $\delta_{\text{crossed}} = 4.93°$ und $\delta_{\text{uncrossed}} = 1.57°$ aus Tabelle 1 den Bereich von $54cm$ bis $11.96m$, womit sich das obige Gedankenexperiment empirisch untermauern lässt.

3 Von der POV-Ray-Szene zum S3D-Animationsfilm

> "Programmieren ist wie küssen: Man kann darüber reden, man kann es beschreiben, aber man weiss erst, was es bedeutet, wenn man es getan hat."
>
> Andrée Beaulieu-Green

3.1 Stereoskopische Kameras für POV-Ray

POV-Ray verfügt in der Version 3.6 bereits über diverse Kameratypen von Fischauge bis zur sphärischen Kamera. Da liegt es nahe, eine Kamera für stereoskopische Bilder zu definieren. Paul Bourke war der erste, der dieses Problem in [2] systematisch angegangen ist. Etwa zeitgleich veröffentlichte Hermann Voßeler die Erweiterung StereoPOV [27] zu POV-Ray 3.5. Dieses Compilat ermöglicht die Generierung von stereoskopischen Ansichten, welche aus zwei Halbbildern bestehen. Diese werden bereits durch StereoPOV direkt im Side-by-Side Verfahren abgelegt, so dass ein StereoPOV-Frame beide Halbbilder enthält. Wolfgang Wieser veröffentlichte daraufhin ein POV-Ray Makro [4], mit dem man stereoskopische Teilbilder für das linke und das rechte Auge einzeln erzeugen kann. In [3] stellte schliesslich auch Paul Bourke ein eigenes Compilat von POV-Ray 3.6.1 vor, das POV-Ray um eine ganze Reihe von stereoskopischen Kameras erweitert. Die wichtigsten Etappen dieser Entwicklung werden im Folgenden kurz vorgestellt. Zum Schluss wird aufgezeigt, wie man bei POV-Ray 3.7 eine stereoskopische Kamera einbauen kann, welche im Rahmen dieser POV-Ray Version die Vorteile moderner Multi-Core Rechner auszunützen in der Lage ist.

3.1.1 Kameradefinition in POV-Ray

Bevor die erwähnten Modifikationen vorgestellt werden, soll das allgemeine Kameramodell von POV-Ray kurz rekapituliert werden. Entscheidend für die Definition einer Kamera in

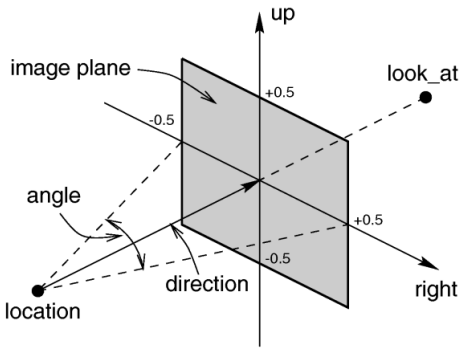


Abbildung 18: Schlüsselwörter für die Kameradefinition in POV-Ray aus [5]

POV-Ray sind die vier Vektoren `location`, `right`, `up` und `direction`. Die letzten drei Vektoren bilden ein Linkssystem (Vgl. Abbildung 18), während `location` einen Ortsvektor für den Kamerastandort liefert. Die Länge der Vektoren `right` und `up` definieren die Breite und Höhe des Bildbereichs der Abbildung, wobei es eigentlich nur auf das Seitenverhältnis der Breite zur Höhe ankommt (engl. image ratio). Die Länge von `direction` definiert den sichtbaren Bildausschnitt, wobei ein kleiner Wert einem Weitwinkel entspricht. Sind zusätzlich der Ortsvektor `look_at` und der Richtungsvektor `sky` definiert, wird das linksseitige Dreibein so gedreht, dass der Vektor `direction` in Richtung des Punktes `look_at` zeigt und dass die horizontale Achse der Kamera normal zur Ebene aufgespannt durch die Vektoren `direction`

und `sky` verläuft. Falls der Öffnungswinkel `alpha` gesetzt ist, wird die Länge d des Vektors `direction` wie folgt neu berechnet

$$d = \frac{w}{2 \tan \frac{\alpha}{2}}, \tag{3}$$

wobei w die Bildbreite und somit die Länge des Vektors `right` bezeichnet.

3.1.2 Parallele Kameraausrichtung nach Bourke

Wie in Abschnitt 2.5.2 beschrieben, müssen die beiden Kameras parallel ausgerichtet sein. Verwendet man dabei ein symmetrisches Frustum, so muss der Öffnungswinkel der Kamera erweitert werden. Nur so lässt sich sicherstellen, dass der Bildbereich mit beiden Kameras vollständig erfasst wird (Vgl. Abbildung 19). Bei diesem Ansatz muss die stereoskopische

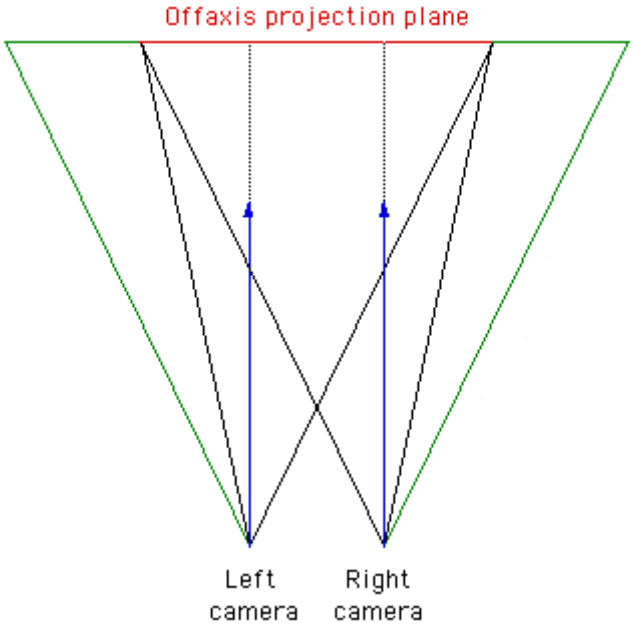


Abbildung 19: Parallele Kameraausrichtung mit erweitertem Frustum nach [2]

Erweiterung Δ der Bildbreite w beim Rendern berücksichtigt werden. Es gilt

$$\Delta = \frac{ew}{2f \tan \frac{\alpha}{2}}, \tag{4}$$

wobei α den Öffnungswinkel, e den Augenabstand und f den Abstand zum Fokuspunkt mit Nullparallaxe bezeichnet. Der Öffnungswinkel der beiden Kameras muss wie in [2] beschrieben angepasst werden. Ausserdem muss diese Anzahl Pixel hinterher auf allen Einzelbildern am linken, bzw. rechten Rand abgeschnitten werden (Vgl. Abbildung 20). Diese Methode liefert zwar zufriedenstellende Bilder, ist aber verfahrenstechnisch aufwändig, da sämtliche Bilder nachbearbeitet werden müssen. Ausserdem muss die Bildbreite im Voraus um den Betrag Δ erweitert werden. Diese Berechnungen können nicht in POV-Ray durchgeführt werden, da die Bildbreite in POV-Ray als Voreinstellung beim Start des Skriptes angegeben werden muss.

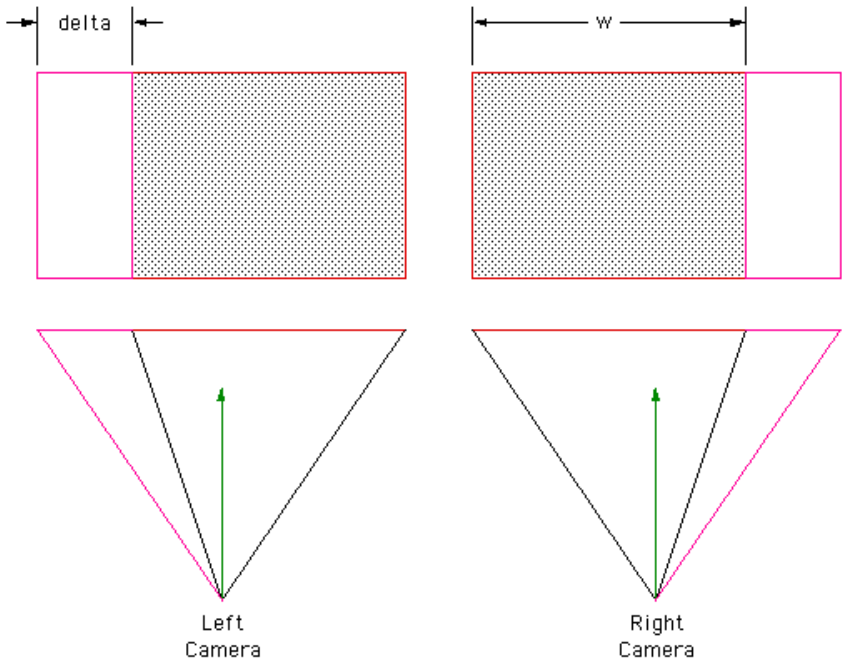


Abbildung 20: Bei paralleler Kameraausrichtung nach [2] muss bei beiden Teilbildern die Erweiterung Δ weggeschnitten werden.

3.1.3 Asymmetrisches Frustum nach Wieser und Bourke

Die entscheidende Vereinfachung bringt die Idee des asymmetrischen Frustums aus Abschnitt 2.5.2. Dabei werden die beiden Richtungsvektoren `direction` der linken und rechten Kamera auf den gemeinsamen Fokuspunkt `look_at` gerichtet (Vgl. Abbildung 21).

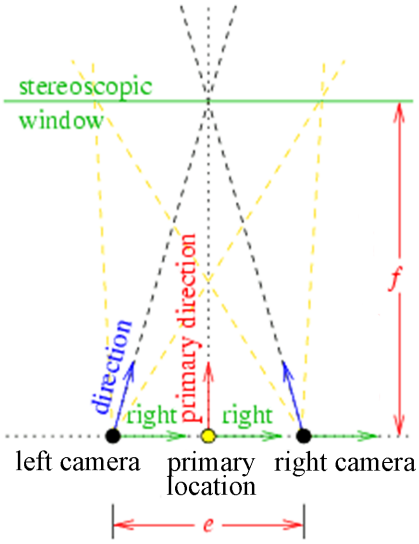


Abbildung 21: Beim asymmetrischen Frustum werden die Richtungsvektoren `direction` der Kameras auf den gemeinsamen Fokuspunkt gerichtet. Aus [4].

Eine Erweiterung der Bildbreite ist damit nicht mehr nötig. Allerdings nimmt man in Kauf, dass das Dreibein bestehend aus `right`, `up` und `direction` nun nicht mehr rechtwinklig ist.

Um diese stereoskopische Kamera realisieren zu können, muss $\mathbf{d}$ = `direction` wie folgt angepasst werden.

$$\mathbf{d}' = \mathbf{d} \mp \frac{ed}{2f} \cdot \mathbf{r}_0 \tag{5}$$

Dabei bezeichnet $\mathbf{r}_0$ einen Einheitsvektor in Richtung `right`.

3.1.4 Stereoskopische Kamera für POV-Ray 3.7

Die stereoskopische Kamera, welche hier vorgestellt wird, basiert auf dem Modell von Wieser und Bourke aus dem vorherigen Abschnitt. Gegenüber [4] stellt sie eine Verallgemeinerung für beliebige Positionen und Ausrichtungen der Kamera dar. Die entsprechenden Makros findet man im Anhang F. Daneben wurde auch ein Compilat für POV-Ray 3.7 erstellt. Abweichend zu [3] werden die neuen Orts- und Richtungsvektoren nicht im Render-Modul sondern im Parser-Modul von POV-Ray berechnet. Dadurch lässt sich Zeit einsparen, da die Berechnungen für jedes Teilbild nur einmal im Parser statt für jeden Sehstrahl in der Render-Engine durchgeführt werden müssen. Weil POV-Ray 3.7 Multithreading unterstützt, kann die Geschwindigkeit gegenüber älteren Compilaten durch den Einsatz eines Multicore-Prozessors nochmals deutlich erhöht werden.

Codesequenz aus dem POV-Ray Parser

```
1  if (New.Type == STEREOSCOPIC_CAMERA)
2  {
3    if((New.Zero_Parallax != HUGE_VAL) && (New.Eye_Offset != HUGE_VAL))
4    {
5      // new camera location (eye shift)
6      VNormalize(tempv, New.Right);
7      VAddScaledEq(New.Location, New.Eye_Offset, tempv);
8      // new camera direction (pointing to the focal point)
9      VLength(Direction_Length, New.Direction);
10     VAddScaledEq(New.Direction,
11        -New.Eye_Offset * Direction_Length / New.Zero_Parallax, tempv);
12   }
13   else
14     Error("The stereoscopic camera needs an eye offset
15         and a zero parallax!");
16 }
```

Codesequenz 1: C++-Code für die stereoskopische Kamera

`VNormalize` auf Zeile 6 setzt den Vektor `tempv` auf einen Einheitsvektor in Richtung des Vektors `New.Right`. Dies entspricht also dem Vektor $\mathbf{r}_0$ aus (5). Die Methode `VAddScaledEq` (a,s,b) setzt den Vektor $\mathbf{a}$ auf $\mathbf{a}+s\cdot\mathbf{b}$, wobei s ein Skalar sein muss. Die Länge `New.Eye_Offset` bezeichnet analog zur Definition aus [3] den halben Augenabstand e, mit dem auf Zeile 7 die neue Kameraposition berechnet wird. Die Zeilen 10 und 11 stellen eine Implementierung von (5) dar, wodurch ein asymmetrisches Frustum definiert wird. Dabei bezeichnet `New.Zero_Parallax` wie üblich den Abstand zum Fokuspunkt, d.h. zur Nullparallaxe.

Einsatz in POV-Ray Eingebaut wir diese stereoskopische Kamera wie in POV-Ray üblich innerhalb der Kameradefinition mit dem Schlüsselwort `stereoscopic`. Ausserdem kommen wie oben beschrieben die stereoskopischen Schlüsselwörter `zeroparallax` und `eyeoffset` hinzu. Im Folgenden ein Beispiel zur Illustration.

```
1  #declare Cam_Location= <12,7,0>;
2  #declare Look_at= <0,0,0>;
3  #declare EYE = -1;           // 1 for right eye, -1 for left eye
4  #declare FL = vlength(Look_at - Cam_Location);  // focal length
5  #declare EYESEP = FL / 30;  // eye separation
6
7  camera {
8      stereoscopic
9      location Cam_Location
10     up y
11     right image_width*x/image_height
12     angle 60
13     sky y
14     look_at Look_at
15     zeroparallax FL
16     eyeoffset EYE * EYESEP / 2
17 }
```

Codesequenz 2: Beispiel zur Anwendung der stereoskopischen Kamera

Als Fokuspunkt wird hier gerade der Ortsvektor `Look_at` verwendet. Der Augenabstand `EYESEP` wird gemäss der gängigen Faustregel auf einen Dreissigstel des Abstandes zum Fokuspunkt gesetzt (Vgl. Abschnitt 2.5.2). Der obige Auszug definiert auf Zeile 3 die Kamera für das linke Auge. Für das rechte Auge muss das Bild demnach mit `#declare EYE = 1;` nochmals neu gerendert werden, so dass man am Ende wie üblich zwei Bilder erhält, bzw. zwei Bildsequenzen im Falle einer Animation. Wie sich im nächsten Abschnitt herausstellen wird, ist dies für die meisten Anwendungen eine gute Ausgangslage, da man sich so die grösste Flexibilität erhält.

3.2 Erstellen einer Blu-ray 3D Disc

Der Workflow zum Erstellen einer Blu-ray 3D Disc besteht aus den folgenden Etappen, die nachfolgend detailliert beschrieben werden:

1. Erstellen zweier Videodateien im MOV- oder AVI-Container für die beiden Augen
2. Transkodierung der Teilfilme ins MPEG-4 MVC Format (MVC Encoding)
3. Erstellen der Dateistruktur für die BD mit MPEG-2TS Videodateien (BD Authoring)
4. Brennen der Blu-ray Disc

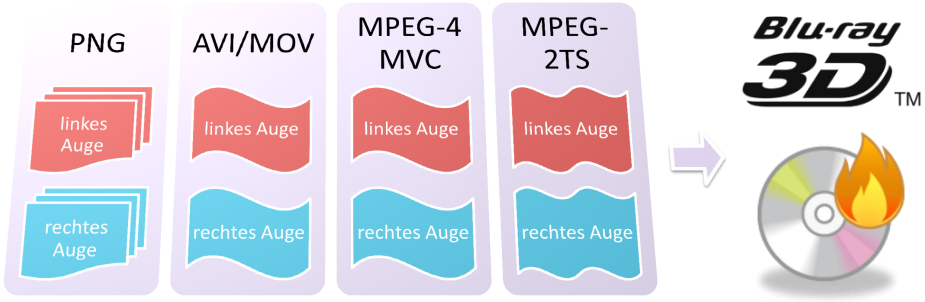


Abbildung 22: Workflow zur Erstellung einer Blu-ray 3D Disc

3.2.1 Erstellen von Videodateien aus Bildsequenzen

Für das Erstellen von Videodateien aus Bildsequenzen sind diverse Tools im Umlauf, vom Kommandozeilenprogramm bis zur GUI-Applikation. Eine Auswahl wird im Anhang B vorgestellt. Im Folgenden einige grundlegenden Überlegungen, welche bei all diesen Tools zum Tragen kommen. Zunächst sollte man darauf achten, dass das verwendete Containerformat von der MVC-Encoding-Software unterstützt wird. Dies gilt derzeit beispielsweise für Videosequenzen, welche mit Apple QuickTime Pro im MOV-Containerformat abgelegt wurden (Vgl. Anhang B.5). Einige der getesteten Tools hatten zudem Schwierigkeiten mit PNG-Frames, welche eine Farbtiefe von 16 Bit pro RGB-Kanal aufwiesen. Falls dies zu verantworten ist, sollte man deshalb mit einer Farbtiefe von 8 Bit arbeiten. Abschliessend nochmals eine Liste mit Bildformaten und Übertragungsraten, welche von der Blu-ray 3D Disc unterstützt werden. Insbesondere gilt es zu beachten, dass für die Full HD Auflösung von 1920×1080 eine Übertragungsrate von 24 Frames pro Sekunde (fps) zu wählen ist.

Auflösung	Übertragungsrate in fps
1920×1080	23.976 ≈ 24
1280×720	59.94 ≈ 60
1280×720	50

Tabelle 2: Auflösung und Übertragungsraten der Blu-ray 3D Disc

3.2.2 MVC Encoding

Liegen die beiden Teilfilme als Videodateien für das linke und das rechte Auge im passenden Containerformat vor, können sie mit einem MVC Encoder ins MPEG-4 MVC Format transkodiert werden. Wie im Abschnitt 2.4.1 beschrieben, wird der Teilfilm für das linke Auge ins MPEG-4 AVC Format transkodiert, während für das rechte Auge nur die Differenzinformationen gespeichert werden. Deshalb ist die Datei mit dem Stream für das rechte Auge normalerweise erheblich kleiner. Leider sind bislang keine Open Source Encoder im Umlauf, sodass man für diesen Schritt auf eine proprietäre Lösung zurückgreifen muss. Im Folgenden eine Liste mit einer Auswahl solcher Encoder.

Name	Hersteller	Homepage
Reference 2.1 Blu-ray 3D/MVC	MainConcept	http://www.mainconcept.com/
DoStudio MVC 3D Encoder	NetBlender	http://www.netblender.com/
Dualstream 3D	Sony	http://www.sonycreativesoftware.com/
CineVision 3.5	Sonic	http://www.sonic.com/

Tabelle 3: Proprietäre MVC Encoding Applikationen

3.2.3 Blu-ray 3D Disc Authoring

Sind die beiden Teilfilme im MVC Format kodiert, kann die Dateistruktur der Blu-ray Disc erstellt werden. Dazu benötigt man eine Blu-ray Authoring Software, mit welcher sich die MVC-Streams ins MPEG-2TS Transportformat transkodieren und in die Dateistruktur der Blu-ray Disc einbetten lassen. Auch dafür liegt derzeit noch keine Open Source Lösung vor. Die folgende Liste gibt eine Übersicht über die proprietären BD Authoring Tools mit 3D Unterstützung.

Name	Hersteller	Homepage
DoStudio Authoring	NetBlender	http://www.netblender.com/
Blu-print 6	Sony	http://www.sonycreativesoftware.com/
Scenarist BD 5.5	Sonic	http://www.sonic.com/

Tabelle 4: Proprietäre BD Authoring Applikationen mit 3D Unterstützung

3.2.4 Blu-ray Disc brennen

Für den letzten Schritt, das Brennen der Blu-ray Disc, gibt es eine ganze Reihe von Freeware Applikationen, die allerdings voraussetzen, dass die Dateistruktur der Disc bereits mit einem Authoring Programm erstellt wurde. Das bekannteste dieser Programme heisst ImgBurn[13]. Eine Liste mit weiteren Blu-ray Disc Burning Applikationen findet man in [28].

3.3 Erstellen eines S3D-Webvideos

Zum Erstellen eines Webvideos sind aus heutiger Sicht zwei Workflows gangbar. Zum einen kann man auch hier zwei Teilvideos für das linke und das rechte Auge erstellen (Vgl. Abschnitt 3.2.1). Da man für das Webvideo jedoch einen kombinierten S3D-Stream braucht, müssen diese Teilvideos hinterher noch zusammengefügt werden. Dies lässt sich beispielsweise mit dem StereoMovie Maker[14] bewerkstelligen. Allerdings werden bei diesem Prozess die beiden bereits komprimierten Teilvideos wieder dekomprimiert und frameweise zusammengefügt. Hinterher wird der S3D-Stream nochmals neu komprimiert. Da die Bildqualität unter dieser zweimaligen Kompression leidet, empfiehlt sich stattdessen der folgende Workflow.

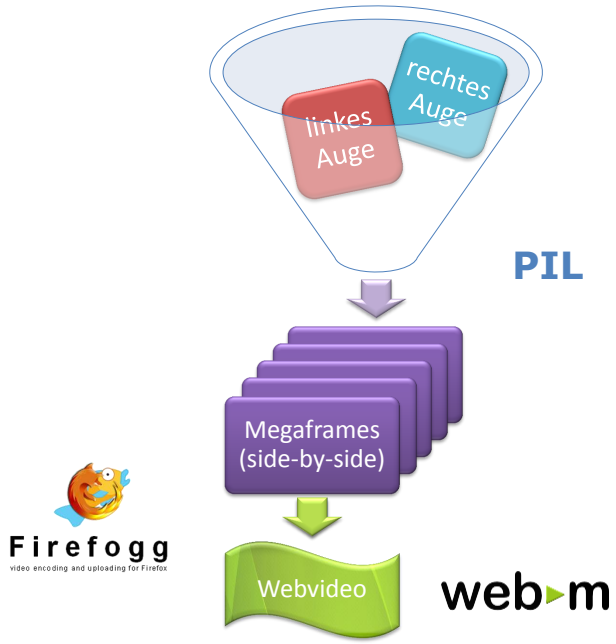


Abbildung 23: Workflow zur Erstellung von Webvideos: Die Frames werden mit der Python Imaging Library (PIL) zu Megaframes zusammengefügt und hinterher mit dem Firefogg ins webM-Format transkodiert.

Alternativ kann man die einzelnen Frames in den Bildsequenzen für das linke und das rechte Auge einzeln zu Megaframes zusammenfügen. Diese neue Bildsequenz aus Megafra-

[13] http://www.imgburn.com/
[14] Siehe http://stereo.jpn.org/ger/stvmkr/.

mes wird hinterher mit dem Firefogg Add-on im Mozilla Firefox in ein Webvideo transkodiert (Vgl. Abbildung 23). Da das Firefogg Add-on auf dem Kommandozeilenprogramm FFmpeg basiert, sollte man bei der Erstellung der Bildsequenzen darauf achten, dass man mit einer Farbtiefe von 8 Bit pro Kanal arbeitet, da FFmpeg derzeit mit 16 Bit PNG-Dateien nicht klarkommt.

3.3.1 Zusammenfügen der Teilframes zu Megaframes

Für das Zusammenfügen der Teilframes zu einem Megaframe kann man aus einer ganzen Reihe von Tools auswählen. Der Einfachheit halber habe ich mich für die Python Imaging Library (PIL) [29] entschieden. Das folgende Skript fügt sämtliche Frames der beiden Teilsequenzen einzeln zu Megaframes der doppelten Bildbreite zusammen. Die Teilframes werden also ohne Reduktion der Auflösung im Side-by-Side-Format nebeneinandergelegt. Eine Version mit File Dialog zum Öffnen der beiden Bildsequenzen ist im Anhang E.1 zu finden.

```
1  from PIL import Image
2
3  num_of_frames = 1151
4  limages=[ "schwamm_l%04d.png"%(i) for i in range(1,num_of_frames+1)]
5  rimages=[ "schwamm_r%04d.png"%(i) for i in range(1,num_of_frames+1)]
6
7  im = Image.open(limages[0])
8  imageWidth=im.size[0]
9  imageHeight=im.size[1]
10
11 for i in range(num_of_frames):
12      bilda = Image.open(limages[i])
13      bildb = Image.open(rimages[i])
14
15      im = Image.new("RGB", (2*imageWidth, imageHeight))
16
17      im.paste(bilda, (0,0,imageWidth,imageHeight))
18      im.paste(bildb, (imageWidth,0,2*imageWidth,imageHeight))
19      im.save("schwamm_stereo%04d.png" % (i+1), "PNG")
20      print (i+1)
```

Codesequenz 3: Python-Skript zur Erzeugung von Megaframes im Side-by-Side-Format

Die Stringformatierungsmaske `%04d` auf Zeile 4 erzeugt vierstellige Zahlen, wobei leere Stellen von links mit Nullen aufgefüllt werden. Dies entspricht der Namenskonvention für Ausgabedateien in POV-Ray. Die Zeilen 7 - 9 dienen dazu, die Bildbreite und Bildhöhe zu ermitteln. Die Schleife auf den Zeilen 11 - 20 lädt die beiden Teilframes in die Variablen `bilda` und `bildb`. Auf Zeile 15 wird der Megaframe definiert, in den auf den Zeilen 17 und 18 die beiden Teilframes eingefügt werden, bevor er auf Zeile 19 wieder mit der bereits erwähnten Stringformatierungsmaske abgespeichert wird.

Will man das Webvideo direkt über HDMI 1.4 auf ein stereoskopisches Display übertragen, müssen die beiden Teilframes in einen Frame der gleichen Auflösung gepackt werden, also in unserem Fall in einen Full HD Frame (1920×1080). Dazu müssen die Teilframes vorgängig auf die halbe Auflösung reduziert werden. Dies geschieht mit dem folgenden PIL-Befehl:

```
resize((imageWidth/2,imageHeight),Image.ANTIALIAS)
```

Dadurch wird der Frame auf die halbe Bildbreite reduziert, was gerade dem Side-by-Side Format aus dem Abschnitt 2.3.3 entspricht. Der zweite Parameter gibt dabei den Filter an, wobei `ANTIALIAS` gemäss Dokumentation ein hochwertiger Downsampling-Filter ist. Auch hierfür findet man im Anhang E.2 eine Version mit File Dialog zum Öffnen der beiden Bildsequenzen.

3.3.2 Erstellung von Webvideos mit dem Firefogg Add-on

Das Firefox Add-on namens Firefogg (Vgl. Anhang B.3) eignet sich vortrefflich, um die Bildsequenzen aus Megaframes in ein Webvideo zu transkodieren. Praktisch dabei ist vor allem, dass man sich weder um Installationspfade noch um Updates kümmern muss. Beides übernimmt der Firefox-Browser, sobald das Add-on installiert ist.

Das Add-on konnte im Rahmen dieser Arbeit erweitert und optimiert werden, so dass ab Version 2.0, welche unter Firefox 4 läuft, auch Bildsequenzen ins webM-Format transkodiert werden können. Die Methode, welche dieser Erweiterung zugrunde liegt, wird auf den folgenden Zeilen vorgestellt. Details zur Optimierung des Add-ons findet man im Anhang C.

```
1  _detect_image_sequence: function(path) {
2    var ppii = path.lastIndexOf(".");
3    var extension = path.substring(ppii, path.length);
4    //ffmpeg path for image sequence:
5    if(extension == ".bmp" || extension == ".png" || extension == ".gif"
6        || extension == ".jpg" || extension == ".jpeg") {
7      var prefix = path.substring(0, ppii);
8      var digits = 0;
9      while(prefix.substr(prefix.length-1, 1) == "0"
10         || (digits == 0 && prefix.substr(prefix.length-1, 1) == "1")) {
11       prefix = prefix.substring(0, prefix.length-1);
12       digits++;
13     }
14     if(digits)
15       path = prefix + "%0" + digits + "d" + extension;
16   }
17   var file = Cc["@mozilla.org/file/local;1"]
18                  .createInstance(Ci.nsILocalFile);
19   file.initWithPath(path);
20   return file;
21 }
```

Codesequenz 4: JavaScript-Methode zur Erkennung von Bildsequenzen

Auf den Zeilen 2 und 3 wird die Endung der Datei ermittelt. Falls auf den Zeilen 5 und 6 eins der kompatiblen Bildformate erkannt wird, wird auf den Zeilen 9 - 13 die Anzahl Stellen der Bildsequenz berechnet. Damit kann gegebenenfalls auf Zeile 15 die zugehörige Stringformatierungsmaske konstruiert werden. Auf den Zeilen 17 - 19 wird eine nsILocalFile-Instanz zum neu konstruierten Pfad definiert, welche es dem Firefogg ermöglicht, die Dateien der Bildsequenz mit dem Kommandozeilenprogramm FFmpeg zu transkodieren (Vgl. Anhang B.2). FFmpeg ist Teil des Add-ons und wird als Subprozess aus dem Add-on heraus gestartet.

Eine Besonderheit des Firefogg ist, dass das GUI, auf dem die Einstellungen für die Transkodierung vorgenommen werden, unter der Adresse http://firefogg.org/make/ auf der Firefogg Website abgelegt ist (Vgl. Abbildung 24). Damit bleiben die GUI-Komponenten flexibel modifizierbar, ohne dass dabei die Clients mit Updates nachgerüstet werden müssen. Dies bedeutet jedoch nicht, dass die Videos zur Transkodierung auf den Firefogg Webserver geladen werden. Die Transkodierung findet wie oben beschrieben lokal statt, wobei die Daten zwischen dem serverseitigen GUI und dem clientseitigen Add-on im JSON Format übertragen werden. Damit der Benutzer mit den Transkodierungseinstellungen nicht bei Null anfangen muss, wird FFmpeg zunächst einmal mit der Option `--info` aufgerufen. Die dadurch ermittelten Daten werden an den Firefogg Server übermittelt und als Grundeinstellungen für die Einstellungsparameter im GUI verwendet.

Abbildung 24: GUI zum Firefogg, einem Firefox Add-on zur Transkodierung von Webvideos ins webM-Format

3.3.3 Veröffentlichung eines Webvideos auf YouTube 3D

Seit Mitte 2009 verfügt YouTube über spezielle Tags, die yt3d-Tags, welche die Wiedergabe von S3D-Webvideos auf stereoskopischen Displays zulassen. Dabei können die Betrachter der Videos im 3D-Menü des YouTube Webplayers die passende Einstellung auswählen (Vgl. Abbildung 25).

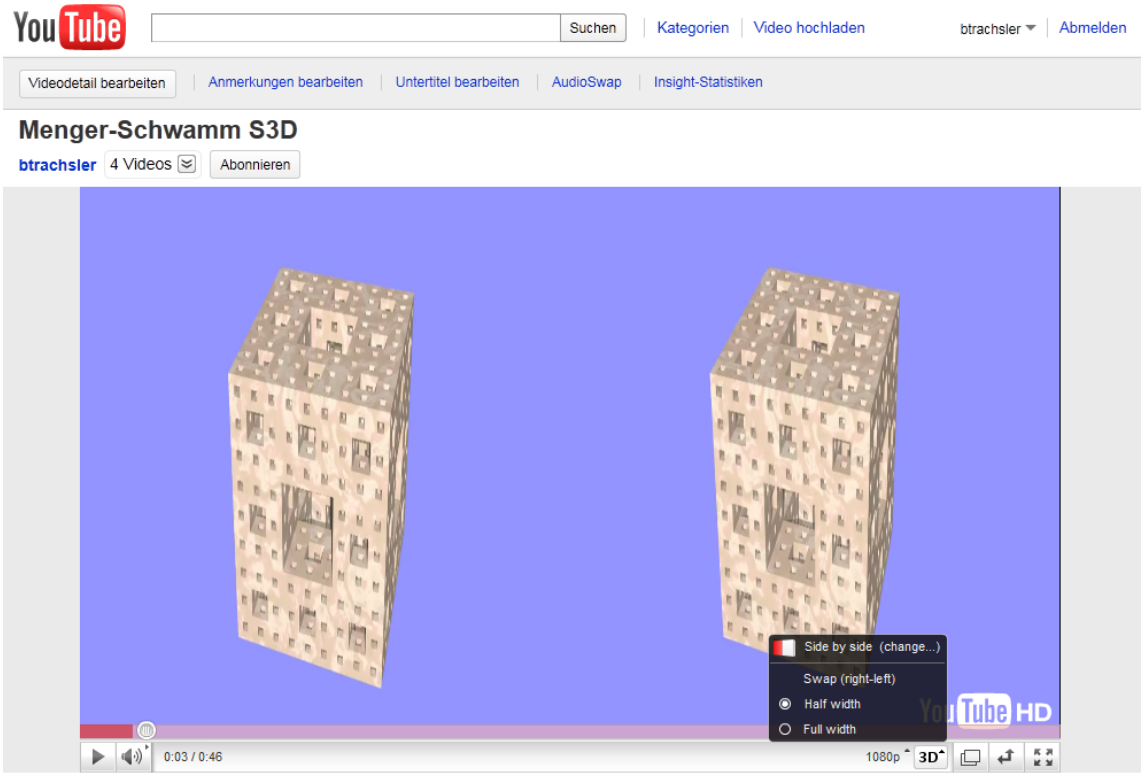


Abbildung 25: Webvideo auf YouTube 3D mit dem 3D-Menü des Webplayers, auf dem unter anderem "Side by Side, Half width" gewählt werden kann.

Zum Abspeichern eines Webvideos auf YouTube 3D genügt ein normaler YouTube Account. Wie zu Beginn dieses Abschnitts beschrieben, muss das S3D-Webvideo im Side-by-Side-Format vorliegen. Es kann dann wie gewohnt hochgeladen werden. Bei den Einstellungen muss zusätzlich im Textfeld "Tags" der yt3d-Tag `yt3d:enable=true` gesetzt sein, damit der

3D-Modus des Webplayers aktiviert wird. Eine detaillierte Dokumentation der yt3d-Tags existierte noch nicht, als diese Arbeit verfasst wurde. In [30] sind jedoch die wichtigsten Tags beschrieben. Im Folgenden eine kurze Zusammenfassung:

- `yt3d:enable=true` aktiviert den 3D-Modus des Webplayers.
- `yt3d:aspect=16:9` legt das Seitenverhältnis des Webvideos fest, in der Regel 16:9.
- `yt3d:swap=true` vertauscht das linke und das rechte Teilbild. Da der YouTube Player das rechte Teilbild in der linken Hälfte des Megaframes erwartet, ist die Standardeinstellung für unsere Videos `true`.

3.3.4 Veröffentlichung eines Webvideos mit dem Video-Tag von HTML5

Sollen die S3D-Videos direkt über den Video-Tag von HTML5 publiziert werden, empfiehlt es sich, sie wie im Abschnitt 3.3.1 beschrieben in einem Full HD Frame zu zeigen. Dadurch wird zwar die Auflösung der beiden Teilframes auf die Hälfte reduziert. Dafür lassen sich solche Videos mit jedem Player abspielen. Dazu muss man lediglich bei den Einstellungen des stereoskopischen Displays angeben, dass die Teilframes im Side-by-Side Format vorliegen. Da zurzeit noch nicht alle Webbrowser das webM-Format unterstützen, empfiehlt es sich, das Webvideo in zwei Versionen anzubieten. Die naheliegende Alternative zum webM-Format ist aus heutiger Sicht MPEG4 AVC im mp4 Containerformat. Im Anhang B.4 wird der Miro Video Converter beschrieben, mit dem Videos ins mp4-Containerformat transkodiert werden können. Das folgende Beispiel zeigt, wie man das Webvideo namens "schwamm" auf diese Weise mit dem Video-Tag in ein HTML-Dokument integrieren kann.

```
1 <video id="movie" width="1920" height="1080" autoplay controls>
2   <source src="schwamm.webm" type='video/webm;codecs="vp8,vorbis"'/>
3   <source src="schwamm.mp4" />
4 </video>
```

Codesequenz 5: Video-Tag mit einem Webvideo im webM- und mp4-Container

Dabei bedeutet `autoplay`, dass das Video direkt geladen und abgespielt werden soll. Mit `controls` zeigt der Browser Einstellungsmöglichkeiten zum Abspielen des Videos an. Mit dem `type` Attribut des `source` Tags kann man das Containerformat, den Video- und den Audio-Codec spezifizieren. Dieses Attribut ist jedoch optional. Weitere Informationen zum Video-Tag in HTML5 findet man in [31, 32].

4 Resultate

"Film = Wahrheit, 24 mal pro Sekunde."

Jean-Luc Godard

In diesem Kapitel geht es um die eigentlichen Videos. Nach einer kurzen Einleitung mit Empfehlungen, welche bei der Herstellung von S3D-Videos beachtet werden sollten, werden die Videos einzeln behandelt. Sie sind ausserdem auf der beiliegenden Blu-ray 3D Disc und auf YouTube[15] im 3D Modus verfügbar.

4.1 Worauf man beim Erstellen von S3D-Filmen achten sollte

Auf der Grundlage von [6, 26] werden hier einige Empfehlungen abgegeben, die das Auftreten von Geisterbildern minimieren und den Betrachtern einen problemlosen Genuss der S3D-Filme ermöglichen sollen.

4.1.1 Horizontale Parallaxe und Disparität

Wie im Abschnitt 2.5.2 dargelegt gelingt die Fusion der Stereobilder nur bis zu einem bestimmten Grenzwinkel. Ist die Disparität zu gross, entstehen Geisterbilder. Mit (1) und (2) auf Seite 17 lassen sich für jede Szene Grenzwerte für die Tiefe nach vorne und nach hinten berechnen. Für Profis sind dazu Applikationen wie der Stereo3D Calculator[16] oder der Stereo Base Calculator[17] im Umlauf, welche ausgehend von den optischen Eigenschaften der (realen) 3D-Filmkameras die optimalen Einstellungen berechnen.

Wenn man allerdings in der Computergrafik schnell zu guten 3D-Einstellungen kommen will, empfiehlt sich wie erwähnt die Faustregel, dass der Abstand zum Fokuspunkt ungefähr das Dreissigfache des Augenabstandes umfassen sollte. Dies stellt für die meisten Szenen eine gute Ausgangslage dar. Wählt man den Faktor zu gross, entstehen so genannte Hypostereo-Bilder, bei denen die Objekte verglichen mit dem Betrachter zu gross erscheinen. Umgekehrt spricht man von hyperstereo, wenn der Faktor zu klein ist, der Augenabstand im Verhältnis zur Distanz somit zu gross gewählt wird. Auf diese Weise entsteht auch bei weiter entfernten Objekten ein Stereoeffekt, der für den Betrachter ungewohnt wirkt. Da bei modernen Produktionen oft fast ausschliesslich mit positiven Parallaxen gearbeitet wird, hört man oft auch die folgende Variante der obigen Faustregel: Der Augenabstand soll einen Dreissigstel der Distanz zum räumlich nächsten Objekt der Szene betragen. Damit tritt die negative Parallaxe gar nicht auf und der Betrachter empfindet das Display als eine Art Fenster zur virtuellen Realität.

Manchmal kann es jedoch entgegen aller Theorie reizvoll sein, eine Einstellung mit starker negativer Parallaxe nah am Grenzwert zu rendern und dabei in Kauf zunehmen, dass Teile der Hintergrundszenerie eine laut Theorie zu grosse positive Parallaxe aufweisen. Die Idee ist dabei, dass der Benutzer die Handlung im Vordergrund verfolgen wird und den Hintergrund quasi ausblendet.

4.1.2 Häufiger Wechsel des Abstandes zum Fokuspunkt

Wenn sich der Abstand zum Fokuspunkt in einem stereoskopischen 3D-Film zu oft ändert, löst dies beim Betrachter Stress aus, vor allem wenn dies mit harten Cuts geschieht. Bei Betrachtern mit wenig Erfahrung mit dem 3D-Medium versuchen die Augen dann ständig neu zu akkommodieren, was zu Ermüdungserscheinungen führen kann.

[15] http://www.youtube.com/user/btrachsler
[16] Der Stereo3D Calculator ist eine iPhone App von RealD.
[17] http://www.stereoeye.jp/software/sbcalc_e.html

4.1.3 Kontrast

Einstellungen mit hohem Kontrast verstärken leider in der Regel Geisterbilder. Wenn möglich sollte man die Objekte also nicht vor einem schwarzen Hintergrund inszenieren. Ist dies dennoch nötig, empfehlen sich dunkle Grautöne anstelle von Schwarz.

4.1.4 Bewegungsparallaxe

Wie im Abschnitt 2.1 dargelegt, ist die Bewegungsparalaxe ein wichtiger Faktor für das räumliche Sehen. Daher können animierte Objekte auf verschiedenen Bildtiefen die räumliche Wahrnehmung verstärken. Gleiches kann natürlich auch durch eine geschickt gewählte Kamerafahrt erreicht werden.

4.1.5 Vertikale Strukturen

Vertikale Strukturen wie die Textur auf einer Hauswand verstärken den räumlichen Eindruck. Daher ist es beispielsweise schwierig, einzelne flache Objekte wie den Schriftzug im Vor- oder Abspann räumlich wirken zu lassen. Gleiches gilt für verschwommene Gebilde wie Nebel. Man braucht dann in der Regel Referenzobjekte, die klarer hervortreten, um den 3D-Effekt zu erzielen.

4.1.6 Texturen mit starkem Rauschen

Texturen, welche ein starkes Rauschen aufweisen wie zum Beispiel eine Rasentextur, können die räumliche Wirkung einer Szene beeinträchtigen. Kommt ausserdem noch eine Bewegung hinzu, kann dies zu einem hochfrequenten Flimmern führen, das für den Betrachter sehr unangenehm wirkt.

4.1.7 Reflexion

Wenn die vorkommenden Reflexionen im Rahmen eines sogenannten Radiosity Modells mit rekursivem Raytracing berechnet werden, treten keine Probleme auf. Werden die Reflexionen allerdings nur approximativ berechnet, beispielsweise mit neu ausgerichteten Normalenvektoren, welche einer 2D-Textur ein Relief verleihen sollen, können bei der stereoskopischen Darstellung der Szene störende Wechselwirkungen zwischen den beiden Teilbildern entstehen.

4.1.8 Glanzlichter

Glanzlichter auf Oberflächen mit starker Spiegelreflexion erscheinen manchmal nur in einem der beiden Teilbilder, weil deren Berechnung natürlich von der Kameraposition abhängt. Dies kann den Betrachter verwirren.

4.1.9 Bildränder bei negativer Parallaxe

Objekte mit negativer Parallaxe sollten nicht vom Bildrand abgeschnitten werden, weil sonst der räumliche Eindruck leidet. Die Ursache dafür ist, dass der Bilderrahmen für den Betrachter auf dem Abstand der Nullparallaxe liegt, also räumlich gesehen weiter hinten. Deshalb sollte man nach Möglichkeit darauf achten, dass abgeschnittene Objekte wie beispielsweise Wände von Gebäuden im Bereich der positiven Parallaxe liegen. Lässt sich dies nicht einrichten, arbeitet man in der Postproduktion mit sogenannten Floating Stereo Windows. Dabei wird der jeweilige Bildrand im linken oder rechten Teilbild um einen schwarzen Streifen verlängert (Vgl. Abbildung 26).

Abbildung 26: Cyan-Rot-Anaglyphe mit einem Floating Stereoscopic Window. Die Streifen sind an beiden Rändern gut sichtbar. http://www.flickr.com/photos/depthography/2771101448/lightbox/

4.1.10 Interferenz horizontaler Bildstrukturen mit der Parallaxe

Horizontale Strukturen, welche ein periodisches Muster aufweisen, können die räumliche Wirkung stören und den Betrachter verwirren, wenn die Periode des Musters gerade der Parallaxe der beiden Teilbilder entspricht. Dieser Effekt tritt auch in der natürlichen (nicht-stereoskopischen) Wahrnehmung auf und kann beispielsweise zur Erzeugung von optischen Täuschungen eingesetzt werden (Vgl. Abbildung 27).

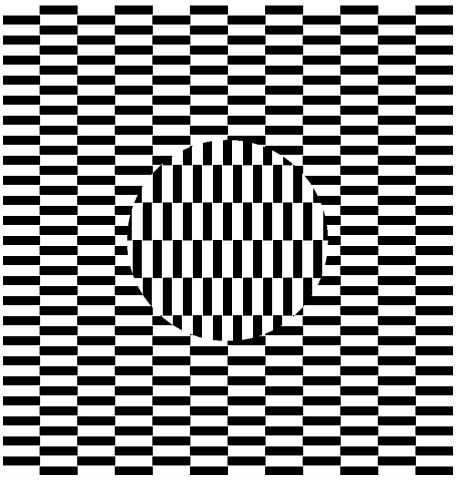


Abbildung 27: Die Ouchi Illusion erweckt den Eindruck, dass sich die kreisrunde Struktur in der Mitte räumlich von ihrer Umgebung abhebt. Solche Effekte können in einem stereoskopischen Bild die räumliche Wahrnehmung stören. http://www.cfar.umd.edu/~fer/cmsc426/cmsc426.html

4.1.11 Divergenz

Von Divergenz oder divergenter Parallaxe spricht man, wenn die positive Parallaxe so gross wird, dass sich die beiden Augäpfel nach aussen drehen müssten, eine Augenbewegung, die in der Natur nicht vorkommt und daher vermieden werden sollte (Vgl. Abbildung 28). Dies kann passieren, wenn stereoskopische Bilder bei gleichbleibendem Abstand auf einem viel breiteren Display gezeigt werden als vorgesehen. Zudem sollte man auch in der Postproduktion beachten, dass die positive Parallaxe nicht beliebig vergrössert werden darf, weil es sonst ebenfalls zur Divergenz der Sichtachsen kommen kann.

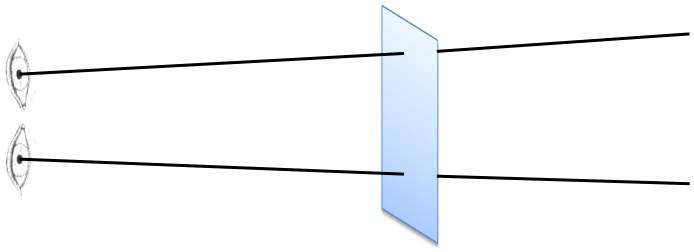


Abbildung 28: Wird die positive Parallaxe zu gross, divergieren die beiden Sichtachsen und es kommt zur divergenten Parallaxe.

4.2 Kaleidozyklen

Der Kurzfilm zur Visualisierung der Kaleidozyklen stammt von Sven Rizzotti und Martin Guggisberg[18]. Da der Film keine Kamerafahrt enthält, war die stereoskopische Nachbearbeitung sehr einfach. Es genügte, die folgende stereoskopische Kamera einzufügen um den S3D-Kurzfilm zu erzeugen. Dies lässt sich durchaus verallgemeinern: Wenn die Kamera statisch bleibt, ist die stereoskopische Nachbearbeitung in der Regel einfach.

```
1  #declare FL = camdist*vlength(<0.0 , -10.0 ,1.0>);
2  #declare Camera_14 = camera {// top view
3       stereoscopic
4       angle 15
5       location camdist*<0.0 , -10.0 ,1.0>
6       right x*image_width/image_height
7       look_at <0.0 , 0.0 , 0.0>
8       zeroparallax FL
9       eyeoffset EYE * FL / 60
10      rotate <90,0,0>
11 }
```

Codesequenz 6: Stereoskopische Kamera für den Kurzfilm Kaleidozyklen

Wie im Abschnitt 4.1.1 vorgeschlagen, wird der Augenabstand auf einen Dreissigstel des Abstands der Kamera zum Fokuspunkt gesetzt. Die horizontale Verschiebung des Auges, das eyeoffset, beträgt dadurch einen Sechzigstel dieses Abstandes, wobei der Fokuspunkt hier gerade dem Nullpunkt des Koordinatensystems entspricht.

4.3 Rote Blutkörperchen

Der Kurzfilm mit den roten Blutkörperchen wurde von Tibor Gyalog erstellt. Die Szene beginnt mit einer Kamerafahrt durch einen Haufen mit 11 Blutkörperchen. Die Kamera ist

[18] http://goodpractice.epistemis.com/kaleidozyklen.html

dabei nach hinten gerichtet, so dass die Blutkörperchen plötzlich von hinten her aufzutauchen scheinen. Es folgt eine längere Phase mit konstanter Kameraeinstellung, in der sich die Blutkörperchen zufällig bewegen und drehen (Vgl. Abbildung 29). Schliesslich wird exponentiell ausgezoomt. Bei der stereoskopischen Nachbearbeitung des Films wurde der Fokus für den grössten Teil des Films konstant gehalten. Dies mag auf den ersten Blick gewagt erscheinen, verleiht dem Video aber erst seine starke Dynamik in den ersten Sekunden. Beim Auszoomen gegen Ende des Films bleibt der Fokus auf den Blutkörperchen, da sonst die positiven Parallaxen zu gross würden. Das grosse Blutkörperchen in der Mitte der Szene musste für die stereoskopische Nachbearbeitung ebenfalls animiert werden, da die Szene sonst im S3D-Video zu statisch wirkte. Dabei musste darauf geachtet werden, dass es nicht zu einer Kollision mit der Kamera kam und dass die negativen Parallaxen nicht zu gross wurden. Den Code zu diesem Kurzfilm findet man im Anhang F.2.

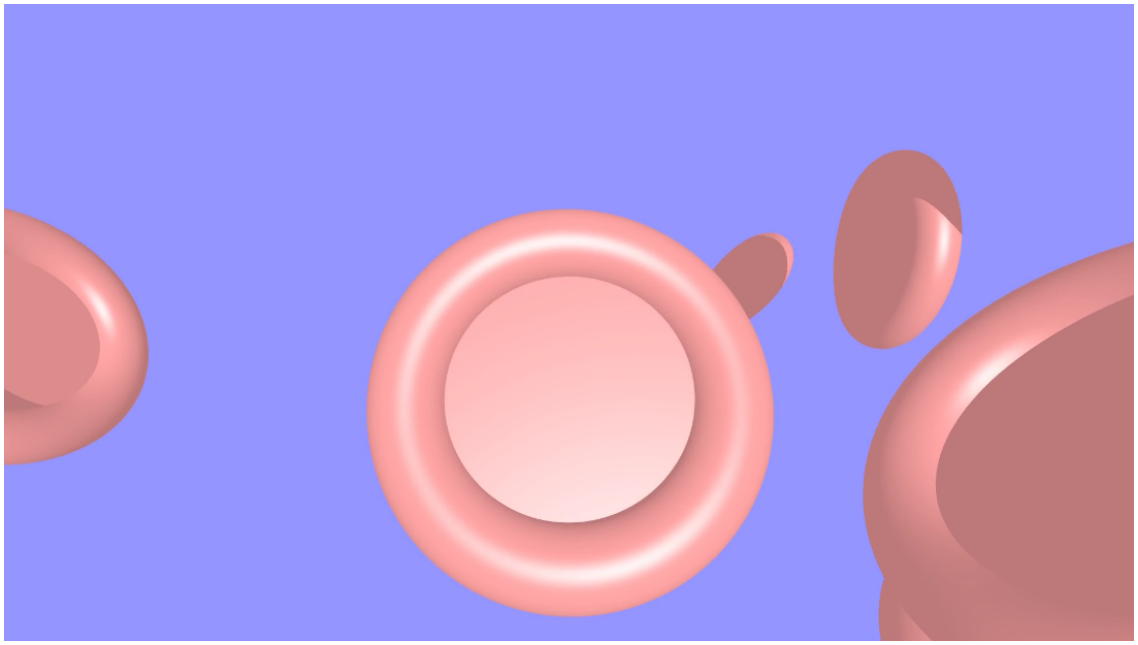


Abbildung 29: Einstellung aus dem Kurzfilm über Blutkörperchen von Tibor Gyalog

4.4 Das Szilassi-Polyeder – Experimente zur horizontalen Parallaxe

Das Szilassi Polyeder (Vgl. Abbildung 30) wurde 1977 vom ungarischen Mathematiker Lajos Szilassi entdeckt. Es handelt sich dabei um ein nichtkonvexes Polyeder vom Geschlecht 1.

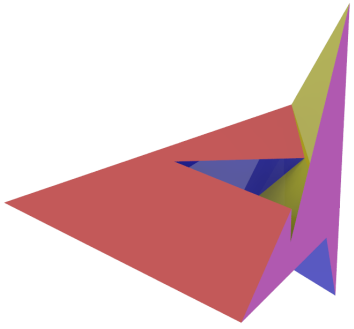


Abbildung 30: Szilassi Polyeder

Das bedeutet, dass das Szilassi Polyeder topologisch äquivalent ist zum Torus. Aufgrund der mehrheitlich nichtkonvexen Seitenflächen gilt das Szilassi Polyeder in der Computergrafik als Herausforderung. Ausgehend von der frei verfügbaren Datei szilassi.obj im Wavefront Object Format[19] wurde mit der Software PoseRay [33] ein POV-Ray Modell des Szilassi Polyeders erstellt. Die stereoskopische Kamera wurde genau wie bei den Kaleidozyklen im Abschnitt 4.2 statisch gewählt. Das Video zeigt das Szilassi Polyeder, wie es sich einmal um die vertikale Achse dreht.

Mithilfe einer Umfrage wurde nun untersucht, welche Form der Parallaxe (negativ oder positiv) beim durchschnittlichen Betrachter besser ankommt. Dazu wurden drei Versionen des obigen Videos erstellt, wobei das Polyeder einmal hinter der Displayebene (positive Parallaxe), einmal vor der Displayebene (negative Parallaxe) und einmal genau in der Mitte dazwischen erzeugt wurde. Diese drei Teilvideos wurden mit einem Python Skript (Vgl. Anhang E.3) zu einem Video zusammengeschnitten, das zunächst die drei Polyeder einzeln rotieren lässt und anschliessend nochmals alle drei miteinander (Vgl. Abbildung 31). Dieses Video wurde im

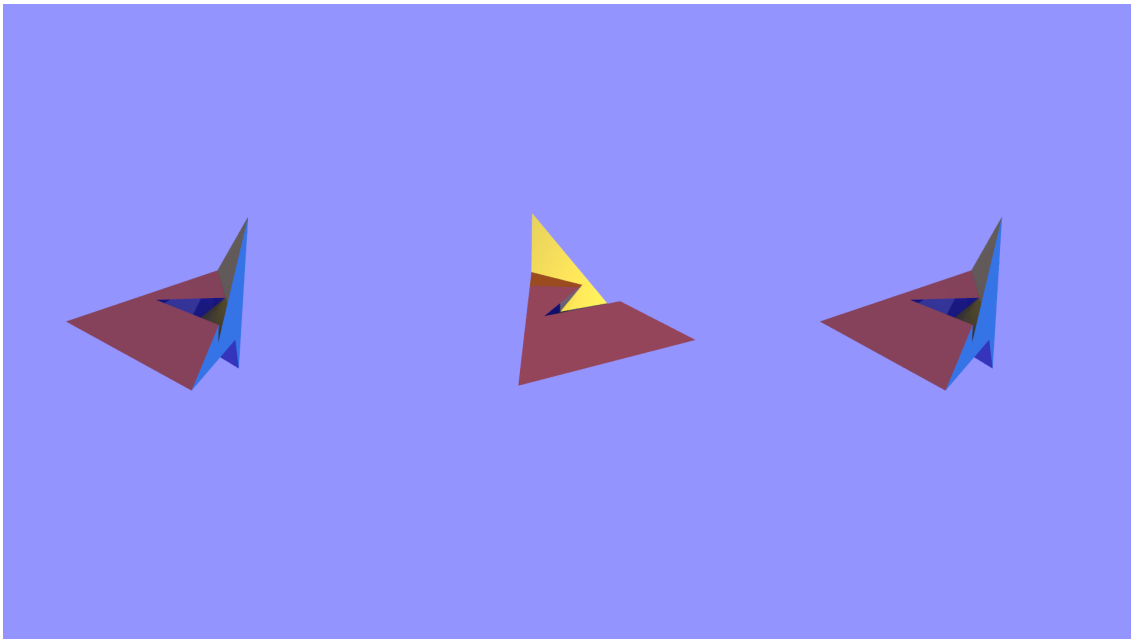


Abbildung 31: Video zur Umfrage über die Parallaxe: Das linke Polyeder wurde mit positiver Parallaxe erzeugt, das mittlere mit negativer Parallaxe und das rechte in der Mitte zwischen den beiden, so dass der Mittelpunkt des Polyeders in der Projektionsebene liegt (Nullparallaxe).

Rahmen der Roadshow von *fitinIT* an der Kantonsschule Solothurn am Mittwoch, dem 17. November 2010, vorgeführt. Dazu wurde die folgende Frage gestellt:

> "Bei welchem der drei Körper erscheint der 3D-Effekt am schönsten?"

Dabei konnten die Befragten zwischen den folgenden Antworten auswählen:

1. Der Körper am linken Rand
2. Der Körper in der Mitte der Szene
3. Der Körper am rechten Rand

Wie die Abbildung 32 zeigt, ergab sich ein klares Bild. Mit 30 von 53 Personen wählten 56.6% der Befragten das mittlere Video mit der negativen Parallaxe aus. Dabei gilt es noch zu

[19] Die Datei szilassi.obj findet man beispielsweise auf http://packages.debian.org/de/sid/hppa/inkscape.

berücksichtigen, dass sich die Befragung an Jugendliche zwischen 15 und 20 Jahren richtete. Ausserdem handelte es sich bei den gezeigten Videosequenzen um Kurzfilme mit maximaler Länge von 3 Minuten. Für einen Film in Spielfilmlänge würde man vermutlich andere Resultate erhalten, da die Augen bei negativer Parallaxe schneller ermüden.

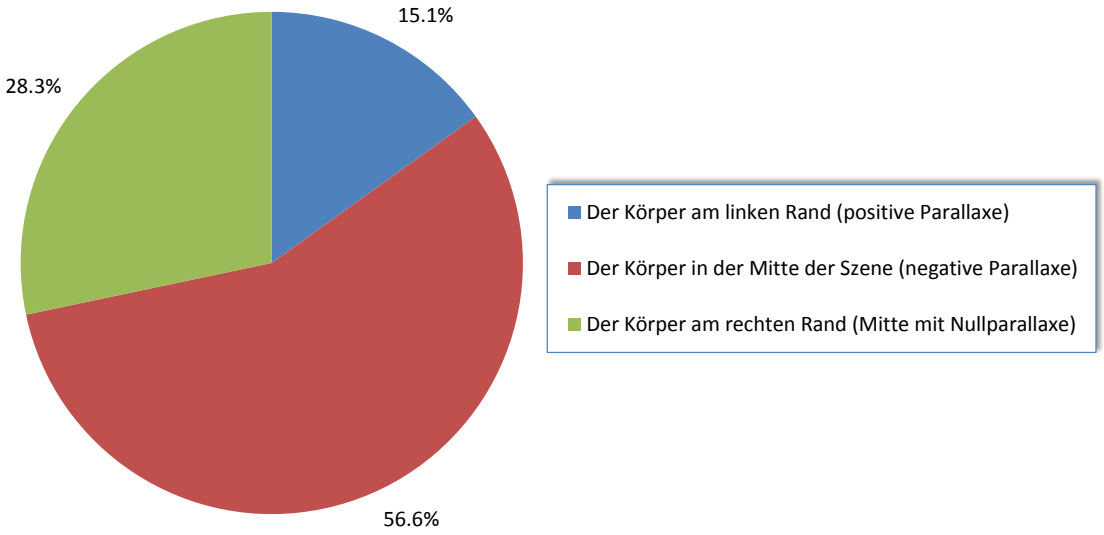


Abbildung 32: Auswertung der Umfrage zur horizontalen Parallaxe:
Bei welchem der drei Körper erscheint der 3D-Effekt am schönsten?

Neben der obigen Frage wurde auch die Frage untersucht, ob der Betrachter bei einem weissen Objekt tatsächlich eher Geisterbilder erkennt als bei einem roten Objekt. Dies würden wir aufgrund der Werte aus Tabelle 1 auf Seite 8 erwarten. Da mit 25 Personen nur knapp die Hälfte der 53 Personen überhaupt Geisterbilder wahrgenommen haben, ist die Datenbasis bei dieser Frage leider zu schwach. Die Antworten decken sich jedoch mit den Resultaten aus der Theorie, da von diesen 25 Personen 20 angaben, dass sie das Geisterbild beim weissen Körper gesehen haben. Es wurden insgesamt etwa gleich viele Schülerinnen wie Schüler befragt. Geschlechtsspezifische Unterschiede konnten keine festgestellt werden. Den Fragebogen und die vollständige Auswertung findet der interessierte Leser im Anhang G.

4.5 Der Menger-Schwamm

Der Menger-Schwamm ist ein Fraktal, das wie folgt aus einem Würfel erzeugt werden kann: Man unterteilt jede Seitenfläche in ein Gitter aus neun flächengleichen Quadraten. In diesem Gitter stanzt man durchgehend von einer Seitenfläche zur gegenüberliegenden ein quaderförmiges Loch durch das mittlere Quadrat. Wiederholt man diese Konstruktion für jeden der verbleibenden zwanzig Teilwürfel rekursiv über mehrere Stufen, erhält man den Menger Schwamm (Vgl. Abbildung 33). Dieses Fraktal kann in POV-Ray mit dem folgenden rekursiven Makro definiert werden. Dabei ist zu beachten, dass ein Dual-Core-Rechner bereits acht Stunden brauchte um die Abbildung 33 zu rendern.

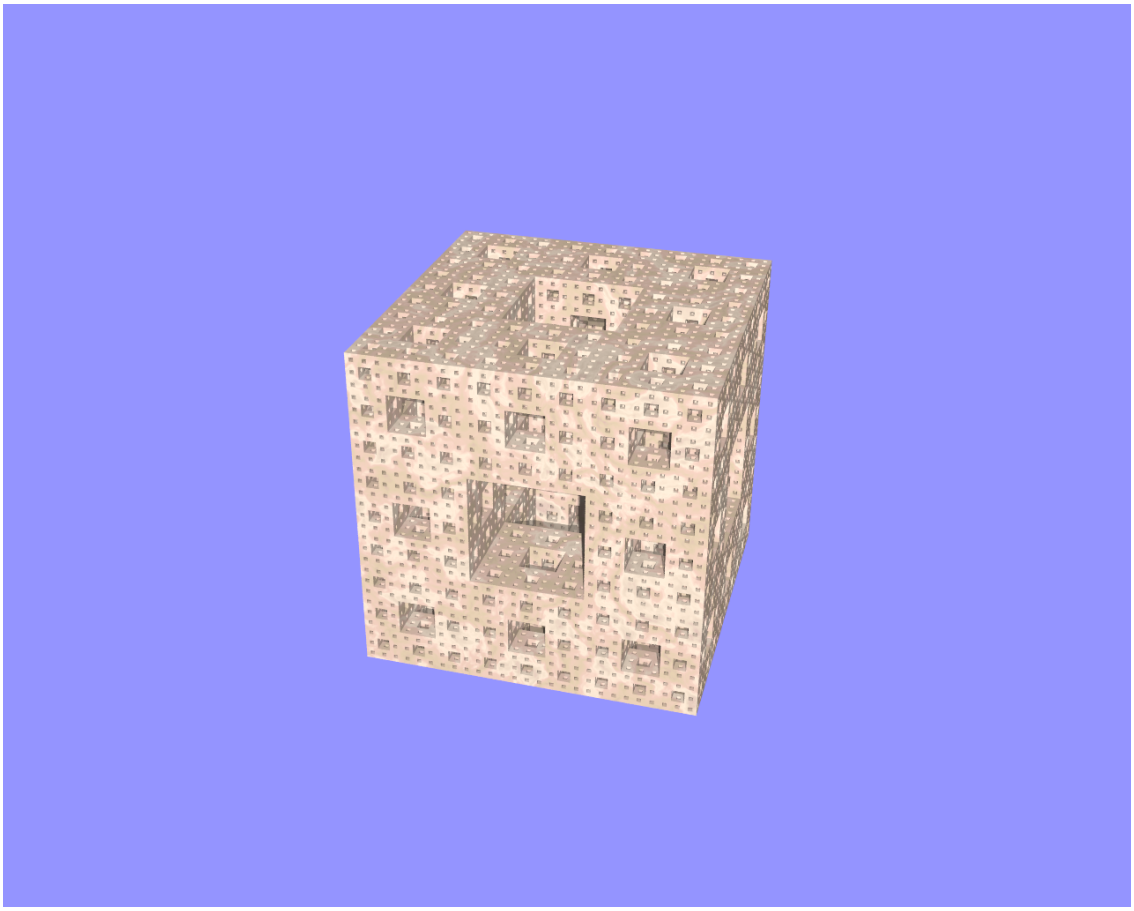


Abbildung 33: Darstellung des Menger-Schwamms mit Rekursionstiefe 4

```
#macro menger_schwamm(p1,p2,tiefe)
  #if(tiefe > 0)
    #local length = p2.x - p1.x;
    #local height = p2.y - p1.y;
    #local depth  = p2.z - p1.z;
      box{
        <p1.x+1/3*length,p1.y+1/3*height,p1.z-D>,
        <p1.x+2/3*length,p1.y+2/3*height,p2.z+D>
      }
      box{
        <p1.x-D,p1.y+1/3*height,p1.z+1/3*depth>,
        <p2.x+D,p1.y+2/3*height,p1.z+2/3*depth>
      }
      box{
        <p1.x+1/3*length,p1.y-D,p1.z+1/3*depth>,
        <p1.x+2/3*length,p2.y+D,p1.z+2/3*depth>
      }

      #local ii = 0;
      #while(ii < 3)
        #local jj = 0;
        #while(jj < 3)
          #local kk = 0;
          #while(kk < 3)
            #if(!((ii = 1 & jj =1) | (ii = 1 & kk = 1) | (jj=1 & kk =1)))
              menger_schwamm(p1+<ii*length/3,jj*height/3,kk*depth/3>,
                p1+<ii*length/3,jj*height/3,kk*depth/3>+(p2-p1)/3,tiefe-1)
```

```
28              #end
29              #local kk = kk + 1;
30            #end
31            #local jj = jj + 1;
32          #end
33          #local ii = ii + 1;
34        #end
35    #end
36  #end
37
38  object{
39    difference{
40      box{ <-1,-1,-1>, <1,1,1>}
41      menger_schwamm(<-1,-1,-1>, <1,1,1>, Iteration_Depth)
42    }
43    texture {
44      T_Grnt6
45    }
46  }
```

Codesequenz 7: Makro zur Definition des Menger-Schwamms

Im Video fährt die Kamera entlang einer Spline-Kurve durch das Fraktal. Dabei musste der Augenabstand vor dem Eindringen ins Fraktal verkleinert werden, da sonst die Parallaxen zu gross würden. Da diese Veränderung während des Films zu einem harten Schnitt führte, musste ich an einer Stelle mit einem Python Skript überblenden (Vgl. Anhang E.4). Bei der Kamerafahrt entlang einer gewundenen Spline-Kurve muss man stets darauf achten, dass sich die Kamera nicht fortwährend um ihre optische Achse dreht. Dies wurde beim Erstellen des Videos durch eine explizite Definition des `sky`-Vektors verhindert, wobei die `sky`-Vektoren zwischen zwei Definitionsstellen interpoliert werden. Zum Rendern dieses Videos benötigte die Workstation I (Vgl. Anhang D.1) 352 CPU Stunden, mit den zwei Dual-Core Prozessoren also etwa vier Tage. Den Code zu diesem Kurzfilm findet man im Anhang F.3.

4.6 Der DNA-Film

Der DNA-Film nahm seinen Anfang als Projektarbeit im Modul Visualisierung im Rahmen des Nachdiplomstudiums für das Ergänzungsfach Informatik. Ziel war die Visualisierung der Doppelhelix und der Nukleotide. Ausserdem sollte die semikonservative Verdoppelung der DNA dargestellt werden. Dies führte zu einem Animationsfilm für den Biologieunterricht[20]. Der Animationsfilm wurde mit POV-Ray 3.7 erstellt. Die räumliche Struktur der Moleküle stammte aus [34], wo die Daten im Molfile Format vorlagen. Diese Daten wurden für den Import in POV-Ray aufbereitet und zur Visualisierung der DNA verwendet. Das Echo auf diesen Kurzfilm war recht positiv, im Oktober 2010 verzeichnete die Seite ca. 3000 Aufrufe. Allerdings wurden bei der Ansicht der benachbarten Nukleotide in der zweiten Szene wiederholt die Darstellung der Wasserstoffbrücken gewünscht.

Auf dieser Grundlage wurde der S3D-Film zur DNA entwickelt. Da es bei diesem Kurzfilm in erster Linie um die räumliche Struktur der Moleküle geht, wurde der Fokuspunkt in der Regel in der Mitte der Doppelhelix oder der beiden Nukleotide gewählt. Dadurch kommt es zu Teilen mit negativer und positiver Parallaxe, was den räumlichen Eindruck verstärkt, da für den Betrachter gewisse Objekte vor der Projektionsebene zu liegen scheinen. Das Hauptproblem, das es zu lösen galt, war die Kamerafahrt durch die Doppelhelix gegen Ende der ersten Szene. Da der Augenabstand aus der ersten Einstellung zu gross war, kollidierte die Kamera für den rechten Teilfilm fortwährend mit DNA-Molekülen. Ausserdem war die negative Parallaxe viel zu gross, so dass auch geübte Betrachter mit Schwindelgefühlen zu

[20] http://www.youtube.com/watch?v=ndql5dsinl4

kämpfen hatten. Schliesslich sollten die vorbeiziehenden DNA-Moleküle wie im Abschnitt 4.1.9 beschrieben keine negativen Parallaxen aufweisen. Die Lösung dieses Problems besteht darin, dass der Augenabstand während des Films verkleinert wird und zwar während sich die Kamera zum höchsten Punkt über der DNA hinauf bewegt, also bevor die Fahrt durch die Doppelhelix beginnt.

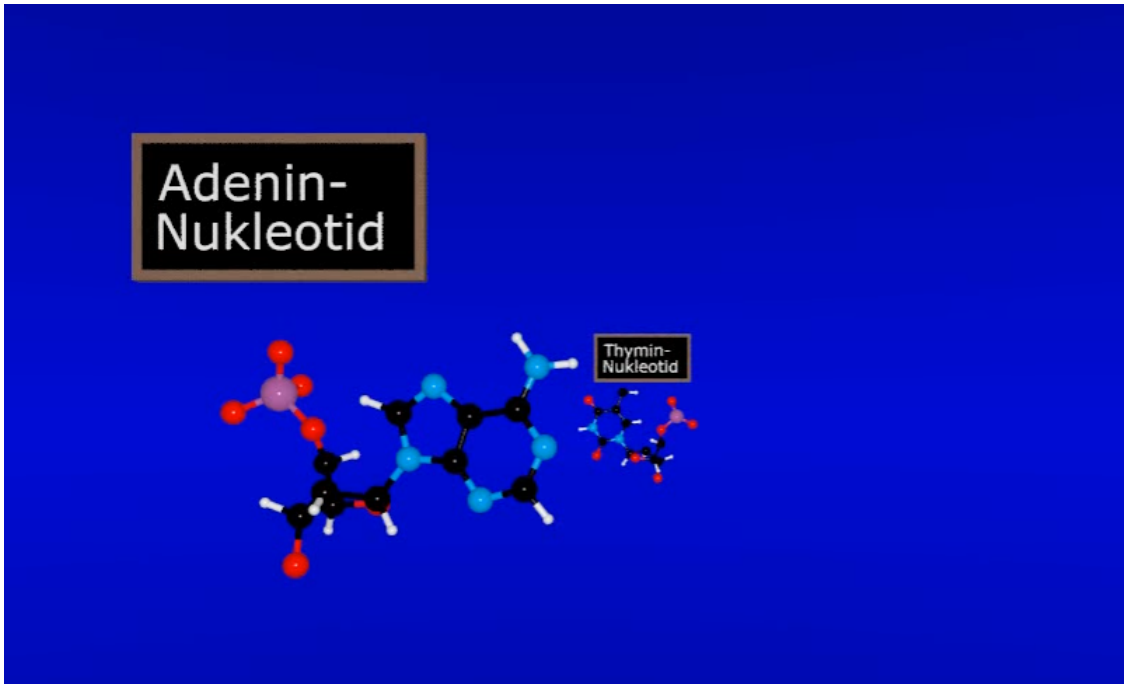


Abbildung 34: Das Adenin-Nukleotid ist bereits an seinem Platz, während das Thymin-Nukleotid langsam näher kommt.

Ein weiteres Problem stellte die zweite Szene dar, weil die zusammengehörenden Nukleotide nicht paarweise auftraten sondern hintereinander erschienen. Dies bedeutete, dass das eine Molekül bereits im Vordergrund vorhanden war, während das zweite langsam von hinten in Erscheinung trat (Vgl. Abbildung 34). Fokussiert die Kamera dabei das langsam auftauchende neue Molekül, erhält das bereits vorhandene Molekül eine starke negative Parallaxe. Lässt man hingegen den Fokus auf dem bereits vorhandenen Molekül, so tritt beim neuen Molekül eine starke positive Parallaxe auf. Daher musste die Szene so geändert werden, dass die beiden Moleküle gleichzeitig auftauchen. Zudem wurden die fehlenden Wasserstoffbrücken ergänzt.

Bei der letzten Szene, bei der es um die semikonservative Verdoppelung geht, erkannte man kleine Fehler der ursprünglichen Version aus der Projektarbeit: Die neu andockenden Moleküle durchdrangen auf ihren Bahnen stellenweise bereits vorhandene Moleküle. Solche Durchdringungen erkannte man in der S3D-Version natürlich sofort. Daher mussten die Bahnkurven dieser Moleküle grundsätzlich überarbeitet werden.

Beim Vorspann und bei den Zwischentiteln wird der Text von hinten nach vorne geschoben, wodurch die Schriftgrösse kontinuierlich zunimmt, bis der Text am Ende wieder ausgeblendet wird. Beim Abspann erscheint der Text auf der Mantelfläche eines um eine horizontale Achse rotierenden Zylinders (Vgl. Abbildung 35). Dabei kommt der räumliche Effekt sehr gut zur Geltung.

Den Code zur ersten Szene und zum Abspann findet man im Anhang F.4. Da jede der drei Szenen sowie Vor-, Abspann und Zwischentitel als einzelne POV-Ray Dokumente vorlagen, mussten die Bilder am Ende in zwei Teilsequenzen mit fortlaufender Nummerierung umgewandelt werden. Das entsprechende Python-Skript findet man im Anhang E.5.

Abbildung 35: Der Abspann des DNA-Films zeigt den Text auf der Mantelfläche eines um eine horizontale Achse rotierenden Zylinders.

5 Diskussion

"Was haben wir getan?
Was tun wir jetzt?
Was sollten wir noch tun?"

Georg Christoph Lichtenberg

Wie diese Arbeit aufzeigt, sind Webvideos inzwischen auch qualitativ eine ernst zu nehmende Alternative zu den Blu-ray Discs, zumal der Video-Tag von HTML5 dem Webdesigner eine einfache Integration von Webvideos erlaubt. Dadurch wird die Nachfrage nach guten Videocodecs weiter steigen. Derzeit scheint das H.264 Videoformat im mp4-Container noch immer qualitativ an der Spitze zu stehen, wobei beispielsweise der vp8-Codec im webM-Container durchaus auch beachtliche Ergebnisse liefert. Bleibt die spannende Frage, welcher Videocodec sich am Ende für Webvideos etablieren wird. Ausserdem führt die rasante Entwicklung der Webvideos derzeit zu Webplattformen, auf denen Videos gekauft oder ausgeliehen werden können, eine ähnliche Entwicklung also, wie sie sich in der Musikindustrie mit den Online Stores bereits etabliert hat.

Bereits ist neben der gängigen Auflösung für Kinofilme (2K = 2048×1080) von 4K = 4096×2160 die Rede. Solche Auflösungen sprengen derzeit noch die Möglichkeiten der Graphikkarten und Displaygeräte, weshalb sie beispielsweise vom HDMI 1.4 Standard noch nicht unterstützt werden. Neuere GPUs werden aber eine noch viel schnellere Verarbeitung der Bilddaten erlauben und somit höhere Auflösungen ermöglichen. Auch wird es im Zuge dieser Entwicklung möglich, stereoskopische Ansichten einer Szene mit Raytracern wie POV-Ray direkt zu rendern, was vor allem für interaktive Anwendungen wie beispielsweise Computerspiele interessant sein dürfte. Nach der Einschätzung von Experten stehen wir demnach kurz vor einem gewaltigen Qualitätssprung. Für solche Anwendungen sollte also bald auch die hier beschriebene stereoskopische Kamera eingesetzt werden können, sofern POV-Ray als Echtzeit Render-Engine in Frage kommt. Wollte man diese Technologie für Head-Mounted Displays oder im CAVE einsetzen, müssten dafür weitere Kameras implementiert werden. Hinweise dazu findet man in [3].

Laut Ansicht von Ralf Schäfer vom Fraunhofer Heinrich-Hertz-Institut werden sich autostereoskopische Displays im Heimbereich mittelfristig durchsetzen. Als Hauptvorteil dieser Technologie fügt er an, dass man dafür keine Brillen mehr benötigt. Tatsächlich waren an der IFA 2010 bereits zahlreiche Displays dieser Art zu sehen, wobei der Betrachter natürlich nach wie vor nur in einem bestimmten Abstand zum Display in den Genuss einer räumlichen Ansicht kommt. In den Kinos scheint sich RealD in der Schweiz als Marktführer durchzusetzen. Offenbar überzeugt die zirkulare Polarisationsfiltertechnik mit ihren günstigen Brillen letztlich trotz teurer Silberleinwand.

Die derzeitige Forschung zur Stereoskopie konzentriert sich laut Alexander Hornung, Forscher bei Disney Research in Zürich, darauf, den stereoskopischen Film von seinen Kinderkrankheiten zu befreien. Offenbar leiden selbst bei grossen Hollywood S3D-Produktionen noch immer rund 10% der Zuschauer unter Übelkeit. Dies gilt es beispielsweise durch die Entwicklung von Algorithmen in der Postproduktion zu verhindern. Was die Verbreitung des S3D-Films anbelangt, glaube ich durchaus, dass dieses Medium einen festen Platz in der Filmindustrie einnehmen wird. Steven Spielberg soll dazu einmal gesagt haben, dass es sich beim S3D-Film unmöglich um einen Hype handeln könne, da die Mehrheit der Zuschauer die Raumwahrnehmung aus dem Alltag gewohnt sei und diese daher auch im Kino letztlich erwarten werde. Was mich betrifft, kann ich diesen Effekt durchaus bestätigen: Bei einem 2D-Film fehlt mir bereits etwas. Daher habe ich die Gelegenheit ergriffen und die hier eingesetzte Technologie auf http://goodpractice.epistemis.com/menger_schwamm.html für den Einsatz im Informatikunterricht veröffentlicht. Mögen weitere S3D-Videos folgen!

A 3D Kinos in der Schweiz

Im September 2010 waren gemäss [35] schweizweit bereits sechzig 3D Kinos in Betrieb. Die folgende Grafik zeigt die Verteilung der verschiedenen Technologien, soweit entsprechende Informationen vorlagen.

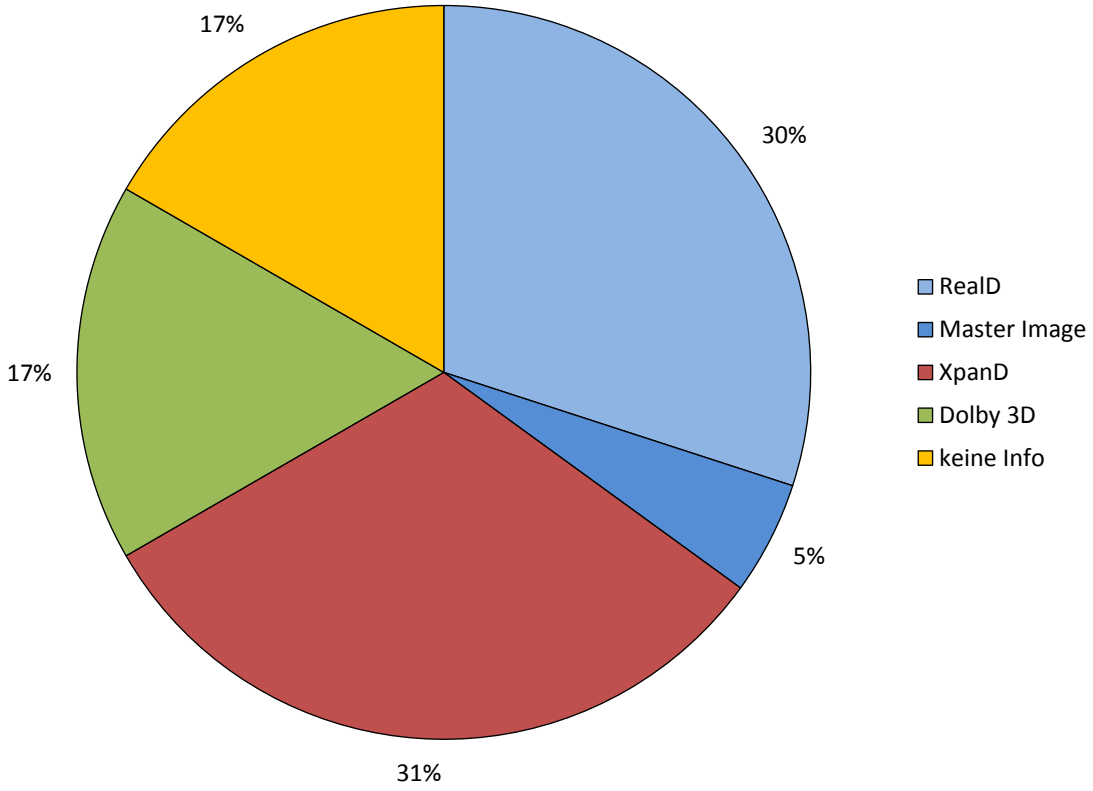


Abbildung 36: Die Grafik zeigt die Verteilung der 3D-Techniken im September 2010.

Dabei basieren RealD und Master Image auf zirkularer Polarisationsfiltertechnik. Der Kunde kann die Brille für wenig Geld beim Ticketkauf erwerben und darf sie hinterher behalten. XpanD ist ein Shutterbrillen-Anbieter. In Kinos mit diesem Standard werden die aktiven Shutterbrillen den Kunden ausgeliehen und hinterher wieder eingesammelt. Dolby 3D basiert auf der Interferenzfiltertechnik. Die passiven Interferenzfilterbrillen sind erheblich teurer als Polarisationsfilterbrillen und werden daher vom Kino in der Regel auch an die Kunden ausgeliehen.

B Programme zur Bearbeitung von Videodateien

In diesem Anhang werden verschiedene Tools vorgestellt, mit denen man Videodateien bearbeiten oder transkodieren kann. Die nachfolgende Liste ist keineswegs vollständig, sondern hat lediglich exemplarischen Charakter mit dem Ziel, gangbare Wege aufzuzeigen. Dabei wurden Applikationen bevorzugt, welche ohne Kosten direkt ab dem Internet installiert werden können.

B.1 VirtualDub

Um eine Bildsequenz in eine Videodatei zu verwandeln, bietet sich unter Windows die Software VirtualDub [36] an. Mit diesem Programm öffnet man den ersten Frame der Sequenz per Menü-Befehl, wonach VirtualDub gleich die gesamte Sequenz auf einer Zeitlinie anzeigt, sofern die Bildsequenz fortlaufend nummeriert ist. Nun lassen sich über weitere Menü-Befehle im Menü Video die nötigen Einstellungen vornehmen (Vgl. Abbildung 37). Am Schluss kann man das Video im AVI-Containerformat abspeichern. Bei der Auswahl des Videocodecs ist

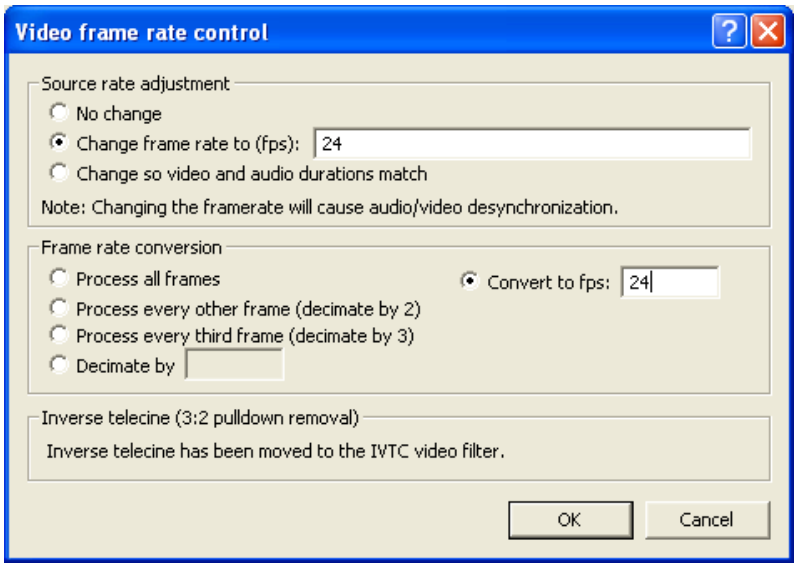


Abbildung 37: Dialogfenster zur Einstellung der Bildübertragungsrate in VirtualDub

Vorsicht geboten, da mit den vorinstallierten Codecs oft entweder schlechte Qualität resultiert oder die Dateien viel zu gross werden. Wie im Kapitel 3.2.1 beschrieben, sollte man vorgängig einen MPEG-4 AVC Codec installieren. Für unsere Zwecke empfiehlt sich der x264-Codec[21]. Dabei ist noch zu beachten, dass die Bildbreite und die Bildhöhe gerade Zahlen sein müssen, da der Codec sonst leider nicht funktioniert. Ausserdem sollte man auf dem Dialogfenster mit den Einstellungen zum x264-Codec die Checkbox "VirtualDub Hack" auswählen, da es sonst zu fehlenden Frames kommen kann (Vgl. Abbildung 38).

[21] Der x264-Codec ist hier zu finden: http://sourceforge.net/projects/x264vfw/files/.

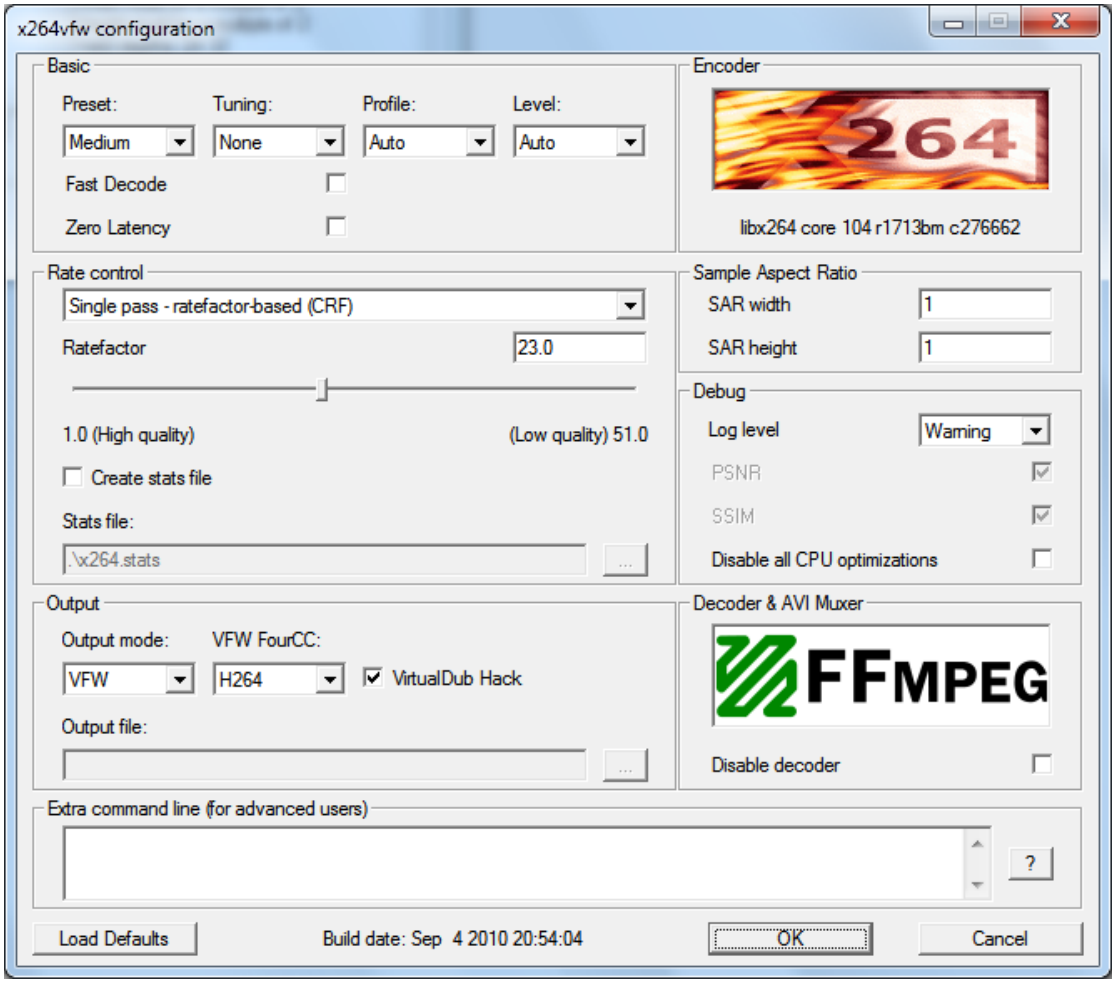


Abbildung 38: Dialogfenster zur Konfiguration des x264-Codec: Die Checkbox "VirtualDub Hack" muss ausgewählt sein.

B.2 FFmpeg

Das Kommandozeilenpgrogramm FFmpeg [37] dient in erster Linie zur Transkodierung von Videodateien. Es liegt in Versionen für Mac OS X, Windows und Linux vor. Wie die folgende Zeile zeigt, kann man damit auch Bildsequenzen in Videos umwandeln.

```
ffmpeg -r 25 -b 2500 -sameq -i filename%0xd.png filename.mov
```

Dabei definiert die Option -r die Bildübertragungsrate (engl. frame rate) und die Option -b die Bitrate des Ausgabevideos. Die Option -sameq sorgt dafür, dass das Ausgabevideo die gleiche Qualität erhält wie die Eingabedaten. Das Konstrukt `%0xd` ist eine Stringformatierungsmaske für Zahlen und bedeutet, dass an dieser Stelle eine Zahl mit x Stellen (engl. **d**igits) steht, wobei leere Stellen von links mit Nullen aufgefüllt werden. Das Containerformat der Ausgabedatei – in unserem Fall mov – bestimmt den Videocodec. Bei der Kodierung von Bildsequenzen mit FFmpeg gilt es zu beachten, dass derzeit keine 16 Bit PNG-Dateien unterstützt werden.

B.3 Firefogg

Das Firefogg Add-on[22] zum Mozilla Firefox Webbrowser ist eine Erweiterung, mit der Videodateien ins webM-Format transkodiert werden können. Im Hintergrund wird dazu das Transkodierungsprogramm FFmpeg eingesetzt. Firefogg kann somit als eine Art GUI zu FFmpeg betrachtet werden. Im Folgenden werden die Schritte zur Transkodierung eines Full HD Webvideos anhand von Abbildungen dokumentiert.

Add-on aufrufen Das Firefogg Add-on wird über den Menübefehl "Make Web Video" im Extras-Menü aufgerufen. Alternativ kann man mit dem Webbrowser auch direkt auf das GUI zugreifen: http://firefogg.org/make/.

Datei(en) auswählen Mit dem Button "Datei auswählen" werden die Dateien zur Transkodierung ausgewählt. Zur Transkodierung einer Bildsequenz wählt man dazu das erste Bild der Sequenz aus.

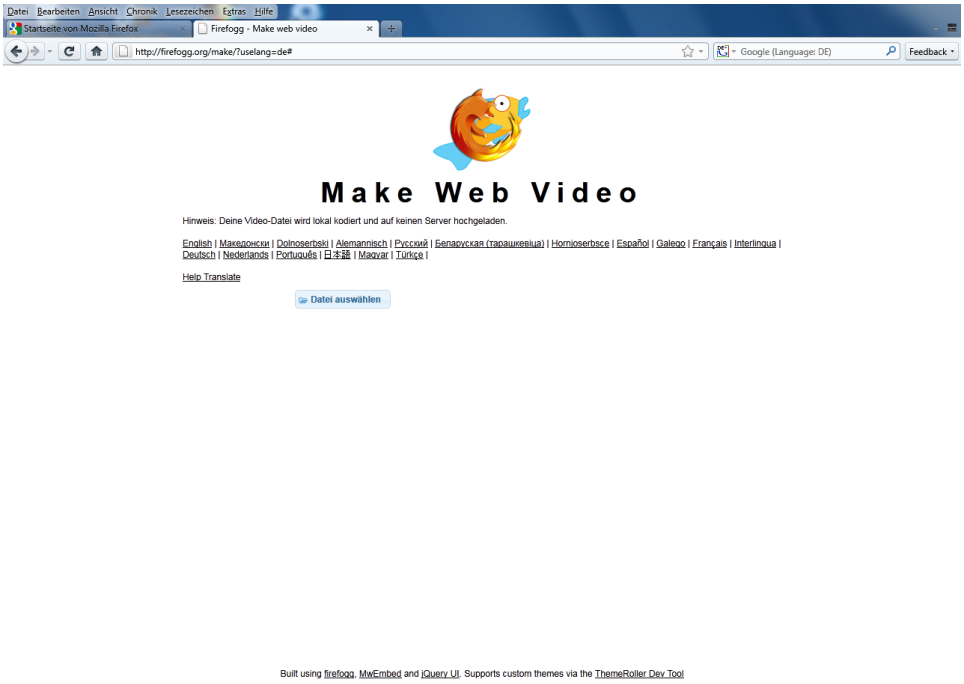


Abbildung 39: Firefogg GUI mit dem Button zur Auswahl der Datei(en) zur Transkodierung

[22] http://firefogg.org/

Voreinstellung: WebM Video (hochwertig) Im Menü Voreinstellung wird das Item "WebM Video (hochwertig)" ausgewählt.

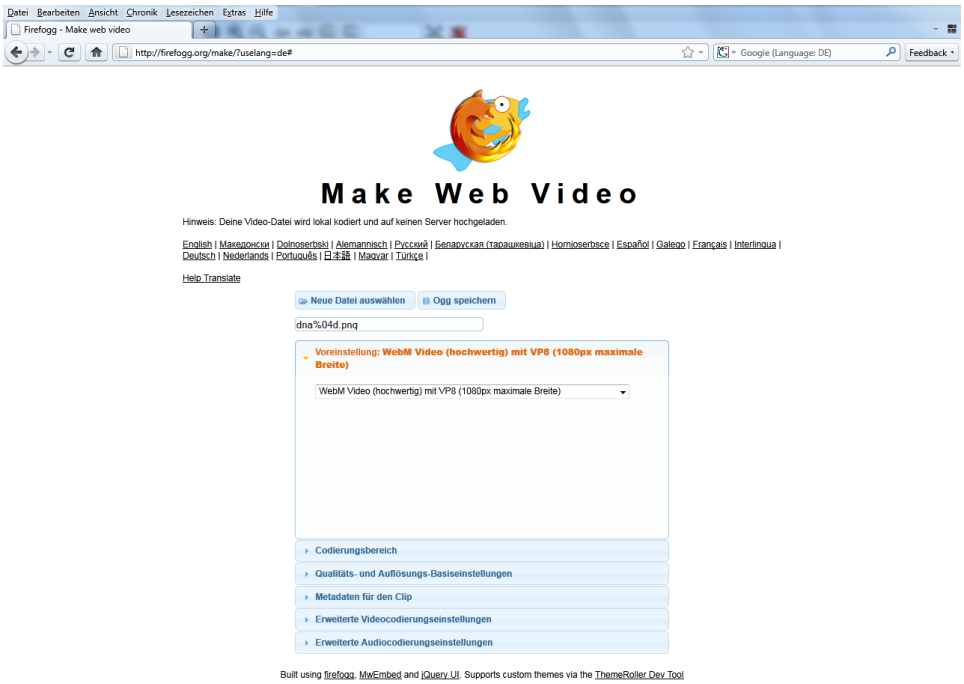


Abbildung 40: Firefogg GUI mit dem Menü zur Wahl der Videokompression

Qualitäts- und Auflösungs-Basiseinstellungen Unter Qualitäts- und Auflösungs-Basiseinstellungen wird die Videoqualität auf Stufe 9 gestellt. Als Videocodec wird "vp8" ausgewählt (Vgl. Abbildung 41).

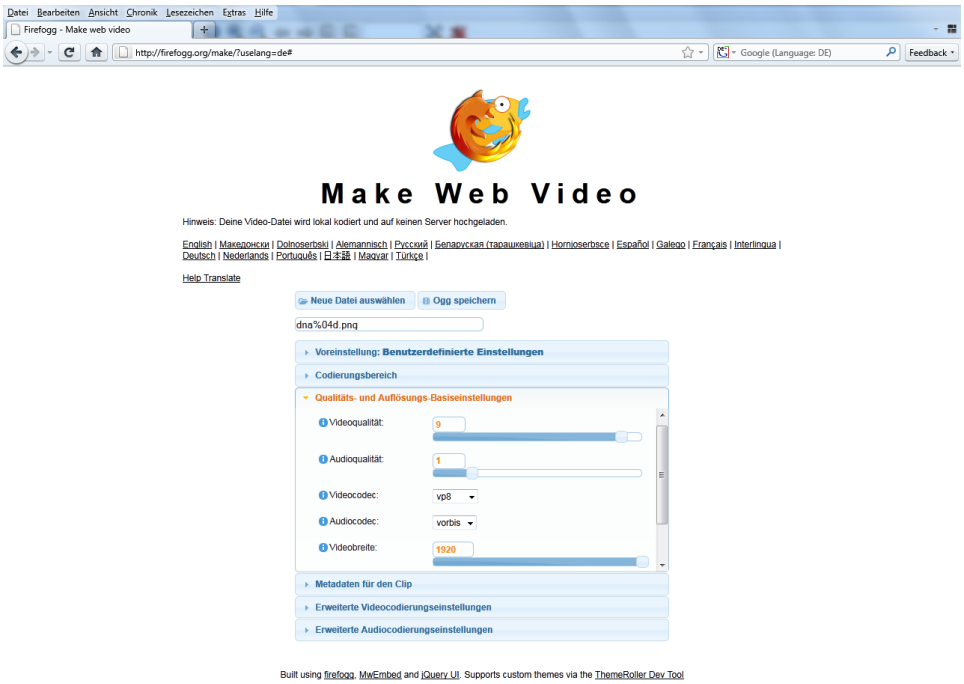


Abbildung 41: Firefogg GUI mit den Qualitäts- und Auflösungs-Basiseinstellungen

Erweiterte Videokodierungseinstellungen Unter den erweiterten Videokodierungseinstellungen wird die Bitrate auf einen Wert über 5000 eingestellt und die Bildfrequenz auf 24 gesetzt (Vgl. Abbildung 42). Für optimale Resultate sollte das Schlüsselbild-Intervall ausserdem auf einen Wert unter 10 gesetzt werden.

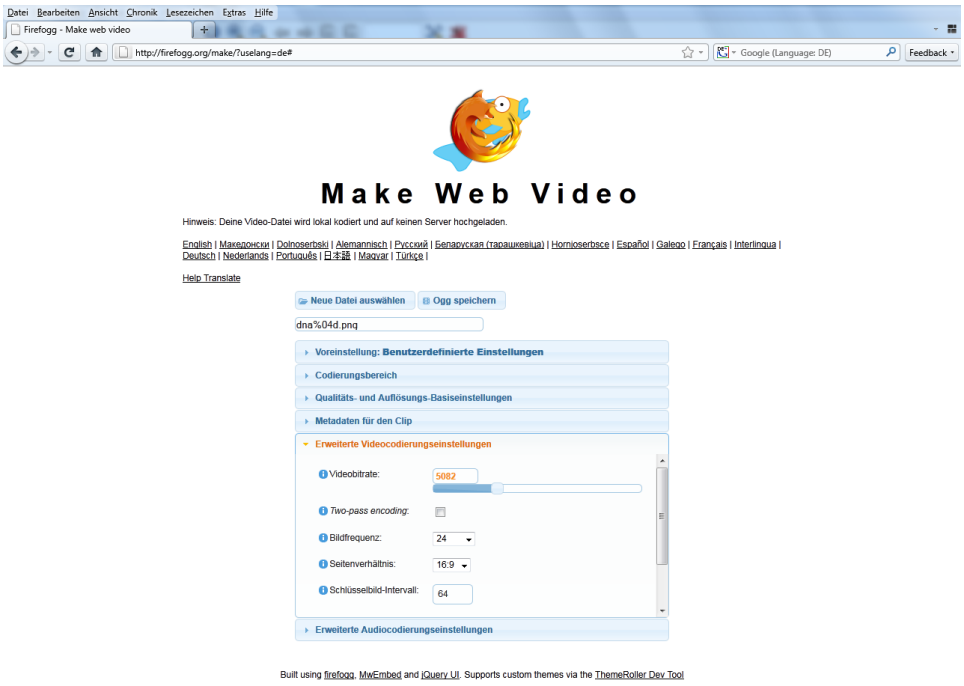


Abbildung 42: Firefogg GUI mit den erweiterten Videokodierungseinstellungen

Video speichern Schliesslich wird das Webvideo mit dem Button "speichern" transkodiert und unter dem angegebenen Namen abgespeichert.

B.4 Miro Video Converter

Der Miro Video Converter läuft unter Windows und Mac OS X. Er verfügt über ein Drag-And-Drop-GUI zur Transkodierung diverser Videoformate. Als Ausgabeformate werden unter anderem webM und mp4 unterstützt. Wie viele anderen Transkodierungsapplikationen basiert der Miro Video Converter auf dem Kommandozeilenprogramm FFmpeg.

B.5 Apple QuickTime Pro

Der Apple QuickTime Player kann auf dem Web kostenlos bezogen werden[23]. Damit man Bildsequenzen transkodieren kann, benötigt man allerdings die kostenpflichtige Version QuickTime Pro. Diese ist für wenig Geld ebenfalls auf dem Web verfügbar. Mit dem Menübefehl "Datei → Bild-Sequenz öffnen..." wird zunächst das erste Bild der Sequenz angegeben. Hat das Programm die Sequenz geladen, kann sie mittels "Datei → Exportieren..." in den mp4-Container exportiert werden. Dabei sollte man unbedingt darauf achten, dass unter Optionen das Videoformat H.264 ausgewählt ist. Ausserdem kann man die Bildsequenz auch direkt mit dem Menübefehl "Speichern" im mov-Containerformat ablegen.

[23] http://www.apple.com/de/quicktime/

C Weiterentwicklung des Firefogg Add-ons

Das Firefogg Projekt verfügt unter https://launchpad.net/firefogg über eine umfangreiche Webplattform für Softwareentwickler. Ausserdem haben interessierte Entwickler die Möglichkeit, von der Firefogg API Gebrauch zu machen, sobald das Add-on im Firefox installiert ist. Eine recht umfangreiche Dokumentation der API findet man unter http://firefogg.org/dev/. Meine ersten Schritte zur Weiterentwicklung des Firefogg machte ich denn auch mit der API. Dabei musste ich feststellen, dass die API zum damaligen Zeitpunkt einen recht gravierenden Fehler aufwies: Wenn man die Bildübertragungsrate angeben wollte, stürzte das Add-on ab. Beim Durchforsten des JavaScript Codes konnte ich schliesslich die Fehlerquelle ausmachen und auf der Entwicklungsplattform ein entsprechendes Bugfix posten (Vgl. Abbildung 43). Als mein Fix schon nach wenigen Stunden vom Betreiber der Entwicklungsplattform, einem

Abbildung 43: Das Bugfix behebt einen Fehler bei der Angabe der Bildübertragungsrate in der Firefogg API.

Webprogrammierer mit dem vielsagenden Pseudonym `j^`, bestätigt und verdankt wurde, beschloss ich, noch etwas mehr Zeit in die Weiterentwicklung des Firefogg zu investieren. Die folgenden Codezeilen sollten den Firefogg für den Umgang mit Bildsequenzen rüsten. Ich veröffentlichte sie als Bugfix auf der Entwicklerplattform unter dem Titel "Simple Extension for Image Sequences".

```
1  //new code for image sequences, 16.09.2010
2  //compute file suffix
3  var selectedPath = fp.file.path;
4  var ppii = selectedPath.lastIndexOf(".");
5  var suffix = selectedPath.substring(ppii,selectedPath.length);
6  //ffmpeg path for image sequence:
7  if(suffix == ".bmp" || suffix == ".png" || suffix == ".gif"
8      || suffix == ".jpg" || suffix == ".jpeg")
9  {
10   var bodyPath = selectedPath.substring(0,ppii-1);
11   var digits = 1;
12   while(bodyPath.substr(bodyPath.length-1,1)=="0")
13   {
14    digits=digits+1;
15    bodyPath = bodyPath.substring(0,bodyPath.length-1);
16   }
17   file = Cc["@mozilla.org/file/local;1"]
18              .createInstance(Ci.nsILocalFile);
19   file.initWithPath(bodyPath.concat("%0"+digits+"d").concat(suffix));
20 }
21 //end code for image sequences
22
23 this._inputFile = file.path; //modified: "file" instead of "fp.file"
24 this.sourceFilename = file.leafName; //modified 16.09.2010
```

Dies zog die folgende kurze Diskussion über Bildsequenzen nach sich, die schliesslich zur Endversion im Kapitel 3.3.2 führte.

j^ wrote on 2010-09-24:

> commited something based on this in
> http://bazaar.launchpad.net/~j/firefogg/trunk/revision/223
> right now it would only work if the image is foo0000.jpg are sequences
> starting with 1 common?
> Changed in firefogg:
> status: New → Fix Committed

Beat Trachsler wrote on 2010-09-24:

> Thanks a lot for the optimization of my code. It looks fine.
> And yes, most of our sequences are starting with 1. Therefore I propose
> the following at line 816 of our _detect_image_sequence function:
>
> while(prefix.substr(prefix.length-1, 1) == "0"
> || (digits == 0 && prefix.substr(prefix.length-1, 1) == "1"))
>
> This should make the job!

j^ on 2010-09-27
> Changed in firefogg:
> status: Fix Committed → Fix Released

Schliesslich stellte ich fest, dass das Firefogg GUI unter http://firefogg.org/make/ eine Begrenzung der Bildauflösung auf 1080×1080 enthielt. Da im Rahmen dieser Arbeit fast ausschliesslich Filmsequenzen in der Full HD Auflösung von 1920×1080 erstellt wurden, bat ich

j^ mit den folgenden Zeilen um eine Anpassung des Web GUI.

```
The slider for the image width and height at http://firefogg.org/make/
restricts image size to 1080x1080. The maximum width of a full hd image
is nevertheless 1920. As I want to publish s3d content at YouTube,
my image width is even 3840 (=2*1920). Therefore I suggest that we change
these slider max values to 4096 (width) and 3072 (height).
This corresponds to the max values of the fmt option at youtube
(see http://en.wikipedia.org/wiki/Youtube).

Here the changes needed for fixing this bug:
File "mw.FirefoggGUI.js"
  line 222: 'range' : { 'min': 0, 'max': 4096 },
  line 228: 'range' : { 'min': 0, 'max': 3072 },
```

Auch dieser Vorschlag wurde von j^ schon nach wenigen Stunden umgesetzt, so dass das Firefogg Add-on nun optimal für die Erstellung meiner Videosequenzen gerüstet ist.

D Technische Daten

In diesem Anhang werden die technischen Daten der Hard- und Software zur Produktion und Wiedergabe der Videosequenzen aufgelistet.

D.1 Workstations

Workstation	Dell Precision T7400	STEG xTreme Home i7-950
Prozessor	Intel Xeon Dual-Core, 3 GHz, 64-bit	Intel Core i7 950, 3 GHz, 64-bit
Arbeitsspeicher	8 GB	8 GB
Festplatte	300 GB	1 TB
Grafikkarte	NVIDIA GeForce GTX 285	NVIDIA GeForce GTX 460

D.2 Displays

Shutterbrillen-Technik: Samsung LED Display UE55C7700

Diagonale	55" (138 cm)
Auflösung	1920×1080
HDMI-Anschlüsse	4 (HDMI 1.4a)
3D-fähig	ja, Shutter-Technologie mit Brille (SSG-2100AB)

Polarisationsfilter-Technik: 2 BenQ MP670 DLP Projektoren

Bilddiagonale / Abstand	0.58 m auf 7.62 m
Auflösung	XGA (1024×768)
Eingänge	1 HDMI 1.3, 1 VGA, 1 S-Video, 1 LAN
3D-fähig	ja, Shutter-Technologie (für dieses Projekt irrelevant)

Die Projektoren können beispielsweise mit einem Screen Splitter von Matrox an die VGA Schnittstelle eines Laptops gehängt werden. Damit der Splitter funktioniert, muss vorgängig eine Treibersoftware installiert werden. Für die Projektion mit der günstigeren Polarisationsfiltertechnik werden ausserdem die folgenden Komponenten benötigt, welche mit geringem Aufwand selbst zusammengebaut werden können[24]:

1. **Polarisationsfilterbrillen für linear polarisiertes Licht**
   Die Polarisationsfilter der Brillen sind im 90° Winkel gegeneinander verdreht. Dabei verwendet man normalerweise auf dem linken Auge eine Winkel von 45° und auf dem rechten Auge einen Winkel von 135°.

2. **2 lineare Polarisationsfilter**
   Die Filter werden vor der Linse der beiden Projektoren montiert. Dabei müssen sie genau wie die Brillen mit einem 90° Winkel gegeneinander verdreht montiert werden, so dass die Polarisationsrichtung verschieden ist. Ausserdem sollten die Polarisationsfilter auf die Polarisationsfilterbrillen abgestimmt werden, damit das Licht des linken Projektors auch tatsächlich das linke Auge erreicht.

3. **Metallische Leinwand** (150cm×300cm)
   Die Leinwand kann auch selbst lackiert werden. Der Lack sollte in diesem Fall einen hohen Anteil Zink oder Aluminium enthalten.

[24] Sämtliche Komponenten können bei der Schweizerischen Gesellschaft für Stereoskopie bestellt werden, http://www.stereoskopie.ch/deutsch/home_d.htm

Für eine optimale Wiedergabe empfiehlt es sich, die beiden Beamer übereinander anzuordnen. Dazu wurde in der Werkstatt des physikalischen Instituts Basel eine Halterung aus Holz angefertigt. (Vgl. Abbildung 44) Diese Anordnung liefert zwar rein technisch gesehen keine

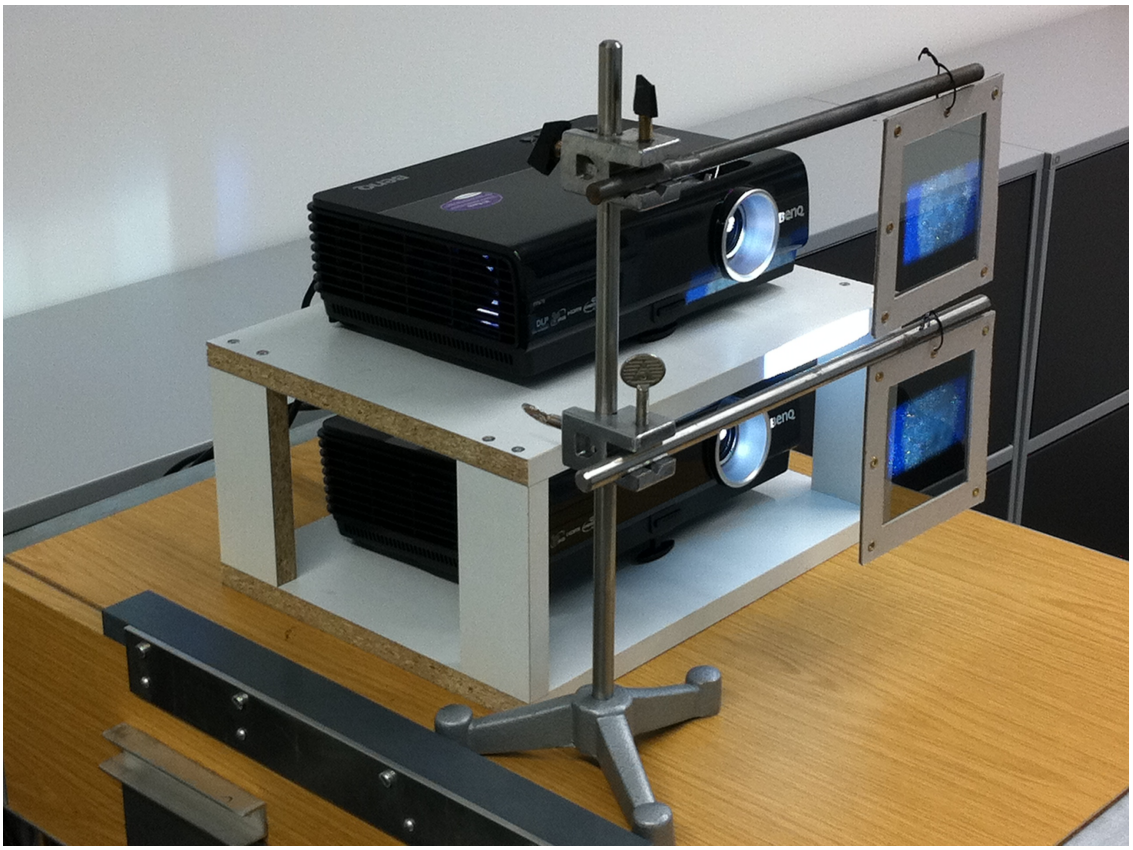


Abbildung 44: Beamerhalterung mit zwei DLP-Beamern der Marke BenQ und linearen Polarisationsfiltern

völlige Separation der beiden Teilvideos, weil auf der metallischen Leinwand immer ein Teil der Polarisation verloren geht. Da das Gehirn diese Geisterbilder jedoch in der Regel eliminiert, solange sie nicht zu stark sind, wird für die meisten Betrachter ein zufriedenstellender räumlicher Eindruck erzeugt.

D.3 Software

POV-Ray

Für diese Arbeit wurde die Version 3.7 beta Release 38 wie in Kapitel 3.1.4 beschrieben um eine stereoskopische Kamera erweitert. Zur Kompilierung werden die Boost-Libraries[25] in der Version 1.37 benötigt. Alternativ kann die stereoskopische Kamera in jeder beliebigen POV-Ray Version als Makro geladen werden.

Python und PIL

Die Bildsequenzen wurden mit Python 2.6.6 und PIL 1.1.7 bearbeitet.

[25] http://sourceforge.net/projects/boost/files/

Firefox 4 und Firefogg

Die Webvideos im webM-Container wurden mit dem Firefogg 2.0.04 Add-on zum Firefox 4 transkodiert.

Apple QuickTime Pro 7

Die Webvideos im mp4-Container wurden mit Apple QuickTime Pro 7.6.8 transkodiert.

NVIDIA 3D Vision Video Player

Zur Wiedergabe auf dem LED-Display wurden die stereoskopischen Videosequenzen mit dem NVIDIA 3D Vision Video Player v1.6.4 und dem NVIDIA GeForce Driver 260.99 abgespielt.

Mozilla Firefox 4

Für die Präsentation mit der Polarisationsfiltertechnik im gemäss den obigen Angaben selbst erstellten Setting wurde der Firefox 4 Browser eingesetzt, weil dieser das Side-by-Side Video mit der doppelten Bildbreite im Vollbildmodus ohne Verzerrung an die beiden Beamer weitergibt.

E Python Programme

E.1 Zusammenfügen der Teilframes zu einem Megaframe

Das folgende Skript erfragt das erste Bild jeder Teilsequenz mittels File Dialog und fügt sämtliche Frames der beiden Teilsequenzen einzeln zu Megaframes der doppelten Bildbreite zusammen.

```
1  import string, os
2  from Tkinter import *
3  import tkFileDialog
4  from tkMessageBox import *
5  from PIL import Image
6
7  #Dialog zum Oeffnen der Files
8  def openImgFile():
9      filename = tkFileDialog.askopenfilename()
10
11     parts = string.rsplit(filename, ".", 1)
12     bodypart = parts[0]
13     suffix = parts[1]
14     bodypart = bodypart[0:len(parts[0])-1]
15     digits = 1
16     while bodypart[len(bodypart)-1] == "0":
17         digits = digits + 1
18         bodypart = bodypart[0:len(bodypart)-1]
19     filename = bodypart + "%0" + str(digits) + "d." + suffix
20
21     return (filename, digits)
22
23 #Dialog zum Speichern der Files
24 def saveImgFile(digits):
25     filename = tkFileDialog.asksaveasfilename(initialfile="output.png")
26
27     parts = string.rsplit(filename, ".", 1)
28     bodypart = parts[0]
29     suffix = parts[1]
30     filename = bodypart + "%0" + str(digits) + "d." + suffix
31     return filename
32
33 #Analysieren der Streams
34 def analyzeImgs(limgname, rimgname):
35     im1 = Image.open(limgname)
36     width=im1.size[0]
37     height=im1.size[1]
38
39     im2 = Image.open(rimgname)
40     if im2.size[0] != width or im2.size[1] != height:
41         showerror('File Stream Error',
42             'Das linke und das rechte Bild passen nicht zusammen!')
43         sys.exit(1)
44     return(width,height)
45
46 #Tk Umgebung initialisieren
47 master = Tk()
48 master.withdraw()
```

```
49
50 #Filenamen fuer linken Stream einlesen
51 (lfilename, digits1) = openImgFile()
52
53 #Filenamen fuer rechten Stream einlesen
54 (rfilename, digits2) = openImgFile()
55
56 #Bilddaten ueberpruefen und Dimension zurueckgeben
57 (imageWidth,imageHeight) = analyzeImgs(lfilename % 1, rfilename % 1)
58
59 #Pfad fuer den resultierenden Stream festlegen
60 filename = saveImgFile(digits1)
61
62 #Streams zusammenfuehren und resultierende Bilder speichern
63 i = 1
64 while os.path.exists(lfilename % i) and os.path.exists(rfilename % i):
65     bilda = Image.open(lfilename % i)
66     bildb = Image.open(rfilename % i)
67
68     im = Image.new("RGB", (2*imageWidth, imageHeight))
69
70     im.paste(bilda, (0,0,imageWidth,imageHeight))
71     im.paste(bildb, (imageWidth,0,2*imageWidth,imageHeight))
72     im.save(filename % i, "PNG")
73     print i
74     i = i + 1
75
76 #Tk Umgebung schliessen
77 master.quit()
```

E.2 Einfügen der reduzierten Teilframes in einen Full HD Frame

Das folgende Skript erfragt das erste Bild jeder Teilsequenz mittels File Dialog, reduziert sämtliche Frames der beiden Teilsequenzen auf die halbe Bildbreite und fügt sie einzeln zu Frames mit der ursprünglichen Auflösung der Teilframes zusammen.

```
1  import string, os
2  from Tkinter import *
3  import tkFileDialog
4  from tkMessageBox import *
5  from PIL import Image
6
7  #Dialog zum Oeffnen der Files
8  def openImgFile():
9      filename = tkFileDialog.askopenfilename()
10
11     parts = string.rsplit(filename, ".", 1)
12     bodypart = parts[0]
13     suffix = parts[1]
14     bodypart = bodypart[0:len(parts[0])-1]
15     digits = 1
16     while bodypart[len(bodypart)-1] == "0":
17         digits = digits + 1
18         bodypart = bodypart[0:len(bodypart)-1]
19     filename = bodypart + "%0" + str(digits) + "d." + suffix
20
```

```
21      return (filename, digits)
22
23  #Dialog zum Speichern der Files
24  def saveImgFile(digits):
25      filename = tkFileDialog.asksaveasfilename(initialfile="output.png")
26
27      parts = string.rsplit(filename, ".", 1)
28      bodypart = parts[0]
29      suffix = parts[1]
30      filename = bodypart + "%0" + str(digits) + "d." + suffix
31      return filename
32
33  #Analysieren der Streams
34  def analyzeImgs(limgname, rimgname):
35      im1 = Image.open(limgname)
36      width=im1.size[0]
37      height=im1.size[1]
38
39      im2 = Image.open(rimgname)
40      if im2.size[0] != width or im2.size[1] != height:
41          showerror('File Stream Error',
42              'Das linke und das rechte Bild passen nicht zusammen!')
43          sys.exit(1)
44      return(width,height)
45
46  #Tk Umgebung initialisieren
47  master = Tk()
48  master.withdraw()
49
50  #Filenamen fuer linken Stream einlesen
51  (lfilename, digits1) = openImgFile()
52
53  #Filenamen fuer rechten Stream einlesen
54  (rfilename, digits2) = openImgFile()
55
56  #Bilddaten ueberpruefen und Dimension zurueckgeben
57  (imageWidth,imageHeight) = analyzeImgs(lfilename % 1, rfilename % 1)
58
59  #Pfad fuer den resultierenden Stream festlegen
60  filename = saveImgFile(digits1)
61
62  #Streams zusammenfuehren und resultierende Bilder speichern
63  i = 1
64  while os.path.exists(lfilename % i) and os.path.exists(rfilename % i):
65      bilda = Image.open(lfilename % i).resize((imageWidth/2,imageHeight),
66                                               Image.ANTIALIAS)
67      bildb = Image.open(rfilename % i).resize((imageWidth/2,imageHeight),
68                                               Image.ANTIALIAS)
69
70      im = Image.new("RGB", (imageWidth, imageHeight))
71
72      im.paste(bilda, (0,0,imageWidth/2,imageHeight))
73      im.paste(bildb, (imageWidth/2,0,imageWidth,imageHeight))
74      im.save(filename % i, "PNG")
75      print i
76      i = i + 1
```

```
77
78 #Tk Umgebung schliessen
79 master.quit()
```

E.3 Zusammenfügen von nebeneinander laufenden Teilvideos

Das folgende Skript fügt die drei nebeneinander laufenden Bildsequenzen der Auflösung 640×1080 zu einer Bildsequenz der Auflösung 1920×1080 zusammen. Dabei soll zuerst die erste Bildsequenz ablaufen, während die übrigen beiden Sequenzen auf dem ersten Bild stehen, anschliessend die mittlere und zuletzt die rechte. Schliesslich sollen nochmals alle Sequenzen nebeneinander ablaufen.

```
1  import string, os
2  from Tkinter import *
3  import tkFileDialog
4  from tkMessageBox import *
5  from PIL import Image
6
7  ##filelist = ["stereopov/szilassi_colored_positive_l001.png",
8  ##             "stereopov/szilassi_colored_negative_l001.png",
9  ##             "stereopov/szilassi_colored_l001.png"]
10
11 filelist = ["stereopov/szilassi_colored_positive_r001.png",
12             "stereopov/szilassi_colored_negative_r001.png",
13             "stereopov/szilassi_colored_r001.png"]
14
15
16 #Laden der Files
17 def loadImgFiles():
18     fnames = []
19     digs = []
20     for filename in filelist:
21         parts = string.rsplit(filename, ".", 1)
22         bodypart = parts[0]
23         suffix = parts[1]
24         bodypart = bodypart[0:len(parts[0])-1]
25         digits = 1
26         while bodypart[len(bodypart)-1] == "0":
27             digits = digits + 1
28             bodypart = bodypart[0:len(bodypart)-1]
29         fnames.append(bodypart + "%0" + str(digits) + "d." + suffix)
30         digs.append(digits)
31     return (fnames, digs)
32
33 #Dialog zum Speichern der Files
34 def saveImgFile(digits):
35     filename = tkFileDialog.asksaveasfilename(initialfile="output.png")
36
37     parts = string.rsplit(filename, ".", 1)
38     bodypart = parts[0]
39     suffix = parts[1]
40     filename = bodypart + "%0" + str(digits) + "d." + suffix
41     return filename
42
43 #Tk Umgebung initialisieren
44 master = Tk()
```

```
45 master.withdraw()
46
47 #Filenamen fuer Sequenzen einlesen
48 (fnames, digs) = loadImgFiles()
49
50 #Pfad fuer die resultierende Sequenz festlegen
51 outputname = saveImgFile(4)
52
53 #Sequenzen zusammenfuehren und resultierende Bilder speichern
54 ii =  1
55 fi = 0
56 for fi in range(4):
57     i = 1
58     bilda = Image.open(fnames[0] % i)
59     bildb = Image.open(fnames[1] % i)
60     bildc = Image.open(fnames[2] % i)
61
62     while os.path.exists(fnames[fi % 3] % i):
63         if fi == 0:
64             bilda = Image.open(fnames[0] % i)
65             bildb = Image.open(fnames[1] % 1)
66             bildc = Image.open(fnames[2] % 1)
67         elif fi == 1:
68             bilda = Image.open(fnames[0] % 1)
69             bildb = Image.open(fnames[1] % i)
70             bildc = Image.open(fnames[2] % 1)
71         elif fi == 2:
72             bilda = Image.open(fnames[0] % 1)
73             bildb = Image.open(fnames[1] % 1)
74             bildc = Image.open(fnames[2] % i)
75         else:
76             bilda = Image.open(fnames[0] % i)
77             bildb = Image.open(fnames[1] % i)
78             bildc = Image.open(fnames[2] % i)
79
80         im = Image.new("RGB", (3*bilda.size[0], bilda.size[1]))
81
82         im.paste(bilda, (0,0))
83         im.paste(bildb, (bilda.size[0],0))
84         im.paste(bildc, (2*bilda.size[0],0))
85         im.save(outputname % ii, "PNG")
86         print ii
87         i = i + 1
88         ii = ii + 1
89
90     fi = fi + 1
91
92 #Tk Umgebung schliessen
93 master.quit()
```

E.4 Überblenden zwischen verschiedenen Kameraeinstellungen

Das folgende Skript verbindet zwei Szenen mit verschiedenem Augenabstand durch Überblenden. Dazu wird in Python der PIL-Befehl `blend` eingesetzt.

```
1  from PIL import Image
2
3  num_of_frames = 7
4  ##images=[ "stereopov5/schwamm_l%04d.png"%(i) for i in range(245,253)]
5  images=[ "stereopov5/schwamm_r%04d.png"%(i) for i in range(245,253)]
6
7  endframe = Image.open(images[7])
8  imageWidth=endframe.size[0]
9  imageHeight=endframe.size[1]
10
11 for i in range(num_of_frames):
12     oldframe = Image.open(images[i])
13
14     newframe = Image.new("RGB", (imageWidth, imageHeight))
15
16     newframe = Image.blend(oldframe, endframe, 0.125*(i+1))
17     newframe.save(images[i], "PNG")
18     print images[i]
```

E.5 Zusammenfügen der Bilder aus verschiedenen Szenen

Das folgende Skript fügt die Bilder aus verschiedenen, mit POV-Ray gerenderten Szenen zu einer zusammenhängenden Bildsequenz zusammen. Als Beispiel dient der DNA-Film, der aus drei Szenen, einem Vorspann, einem Abspann und zwei Zwischentiteln besteht.

```
1  import string, os
2  from Tkinter import *
3  import tkFileDialog
4  from tkMessageBox import *
5  from PIL import Image
6
7  ##filelist = ["stereopov2/vorspann_l001.png",
8  ##            "stereopov2/dna_modell_v_3_l0001.png",
9  ##            "stereopov2/zwischentitel_1_l001.png",
10 ##            "stereopov2/dna_modell_v_4_l0001.png",
11 ##            "stereopov2/zwischentitel_2_l001.png",
12 ##            "stereopov2/dna_replikation_v_1_l001.png",
13 ##            "stereopov2/abspann_l001.png"]
14
15 filelist = ["stereopov2/vorspann_r001.png",
16             "stereopov2/dna_modell_v_3_r0001.png",
17             "stereopov2/zwischentitel_1_r001.png",
18             "stereopov2/dna_modell_v_4_r0001.png",
19             "stereopov2/zwischentitel_2_r001.png",
20             "stereopov2/dna_replikation_v_1_r001.png",
21             "stereopov2/abspann_r001.png"]
22
23 #Laden der Files
24 def loadImgFiles():
25     fnames = []
26     digs = []
```

```
27      for filename in filelist:
28          parts = string.rsplit(filename, ".", 1)
29          bodypart = parts[0]
30          suffix = parts[1]
31          bodypart = bodypart[0:len(parts[0])-1]
32          digits = 1
33          while bodypart[len(bodypart)-1] == "0":
34              digits = digits + 1
35              bodypart = bodypart[0:len(bodypart)-1]
36          fnames.append(bodypart + "%0" + str(digits) + "d." + suffix)
37          digs.append(digits)
38      return (fnames, digs)
39
40  #Dialog zum Speichern der Files
41  def saveImgFile(digits):
42      filename = tkFileDialog.asksaveasfilename(initialfile="output.png")
43
44      parts = string.rsplit(filename, ".", 1)
45      bodypart = parts[0]
46      suffix = parts[1]
47      filename = bodypart + "%0" + str(digits) + "d." + suffix
48      return filename
49
50  #Tk Umgebung initialisieren
51  master = Tk()
52  master.withdraw()
53
54  #Filenamen fuer Sequenzen einlesen
55  (fnames, digs) = loadImgFiles()
56
57  #Pfad fuer die resultierende Sequenz festlegen
58  outputname = saveImgFile(4)
59
60  #Sequenzen zusammenfuehren und resultierende Bilder speichern
61  ii =  1
62  for filename in fnames:
63      i = 1
64      while os.path.exists(filename % i):
65          bild = Image.open(filename % i)
66
67          im = Image.new("RGB", (bild.size[0], bild.size[1]))
68
69          im.paste(bild, (0,0))
70          im.save(outputname % ii, "PNG")
71          print ii
72          i = i + 1
73          ii = ii + 1
74
75  #Tk Umgebung schliessen
76  master.quit()
```

F POV-Ray Programme

F.1 Makros für stereoskopische Kameras

Im Folgenden wird der Inhalt des include-Files stereocam.inc wiedergegeben, welches eine Verallgemeinerung der stereoskopischen Kameras nach Wieser und Bourke [4, 3] zur Verfügung stellt. Das File enthält drei Makros zur Definition einer stereoskopischen Kamera. Der Makro `setStereoCam` definiert die Kamera mithilfe der Kameraposition und des Linksseitigen Dreibeins `right_`, `up_` und `direction_`. Hinzu kommen der (fakultative) Öffnungswinkel α als `angle_` sowie die folgenden stereoskopischen Grössen:

- halber Augenabstand $\frac{e}{2}$ – `o_`
- Abstand f zum Fokuspunkt – `f_`

Mit dem Makro `setExtendedStereoCam` lassen sich zusätzlich die Vektoren `look_at_` und `sky_` definieren. Der Makro `setSimpleStereoCam` kommt hingegen neben den stereoskopischen Grössen mit den Vektoren `look_at_`, `sky_` und `location_` aus.

```
1  /*************************************************************
2   *                                                           *
3   * Stereoscopic Camera (Wieser and Bourke)                   *
4   * Beat Trachsler, KZO Wetzikon 2010                         *
5   *                                                           *
6   * Warning!                                                  *
7   * Don't manually set the camera properties listed below.    *
8   * Otherwise you risk loosing the sterescopic settings.      *
9   *                                                           *
10  * The stereoscopic camera causes a parse warning:           *
11  * "Camera vectors are not perpendicular."                   *
12  * This is unavoidable and can therefore be ignored.         *
13  *                                                           *
14  *************************************************************/
15
16 // setStereoCam
17 // Stereoscopic Camera
18 // right_     - right vector
19 // up_        - up vector
20 // direction_ - direction vector
21 // location_  - camera location
22 // angle_     - horizontal viewing angle in degrees
23 //      angle_ = 0       - no viewing angle specified
24 //      0 < angle_ < 180 - horizontal viewing angle
25 // o_      - eye offset, half of the distance between the eyes
26 // f_      - focal length = distance of zero parallax
27 #macro setStereoCam(right_, up_, direction_, location_,
28             angle_, o_, f_)
29   #local right_unit=vnormalize(right_);
30   #local cam_dir=vnormalize(direction_);
31
32   #if(f_)
33     #local zero_parallax = f_;
34   #else
35     #local zero_parallax = 1;
36   #end
```

```
37
38    // Camera moves along right vector for different sides:
39    #local cam_loc = location_ + o_ * right_unit;
40
41    // Camera direction: right eye looks left and left eye right so
42    // that the rays cross at the stereo window, i.e. at distance
43    // zero_parallax from the camera.
44    #local cam_dir = cam_dir - o_ / zero_parallax * right_unit;
45
46    // Apply appropriate scaling:
47    #if(angle_ > 0 & angle_ < 180)
48      #local cam_dir = cam_dir*vlength(right_)
49                 / (2.*tan(angle_* pi/360));
50    #else
51      #local cam_dir = cam_dir*vlength(direction_);
52    #end
53
54    // Here comes the camera def:
55    perspective
56    right right_
57    up up_
58      direction cam_dir
59    location cam_loc
60  #end
61
62  // setExtendedStereoCam
63  // Extended Stereoscopic Camera with look_at vector
64  // right_      - right vector
65  // up_         - up vector
66  // direction_  - direction vector
67  // sky_        - sky vector
68  // look_at_    - look_at vector
69  // location_   - camera location
70  // angle_      - horizontal viewing angle in degrees
71  //       angle_ = 0       - no viewing angle specified
72  //       0 < angle_ < 180 - horizontal viewing angle
73  // o_       - eye offset, half of the distance between the eyes
74  // f_       - focal length = distance of zero parallax
75  #macro setExtendedStereoCam(right_, up_, direction_, sky_,
76                 look_at_, location_, angle_, o_, f_)
77    #local right_length = vlength(right_);
78    #local handedness = vdot(vcross(up_,direction_),right_);
79    #local cam_dir=vnormalize(look_at_ - location_);
80    #local cam_right=vnormalize(vcross(sky_,cam_dir));
81    #local cam_up=vcross(cam_dir, cam_right)*vlength(up_);
82    #local cam_dir=cam_dir*vlength(direction_);
83    #if(handedness > 0)
84      #local cam_right=cam_right*right_length;
85    #else
86      #local cam_right=-cam_right*right_length;
87    #end
```

```
88    setStereoCam(cam_right, cam_up, cam_dir, location_,
89              angle_, o_, f_)
90  #end
91
92  // setSimpleStereoCam
93  // Simple Stereoscopic Camera with look_at vector
94  // sky_          - sky vector
95  // look_at_      - look_at vector
96  // location_     - camera location
97  // angle_        - horizontal viewing angle in degrees
98  //       angle_ = 0        - no viewing angle specified
99  //       0 < angle_ < 180 - horizontal viewing angle
100 // o_        - eye offset, half of the distance between the eyes
101 // f_        - focal length = distance of zero parallax
102 #macro setSimpleStereoCam(sky_, look_at_, location_,
103                 angle_, o_, f_)
104   #local cam_dir=vnormalize(look_at_ - location_);
105   #local cam_right=vnormalize(vcross(sky_,cam_dir));
106   #local cam_up=vcross(cam_dir, cam_right);
107    setStereoCam(cam_right*image_width/image_height, cam_up,
108             cam_dir, location_, angle_, o_, f_)
109 #end
```

F.2 Blutzellen in POV-Ray

Das folgende POV-Ray Programm von Tibor Gyalog visualisiert 11 rote Blutkörperchen. Das Blutkörperchen wird dabei auf den Zeilen 20 - 60 als POV-Ray Blob verbunden mit einer zylindrischen Scheibe definiert. Auf den Zeilen 107 - 111 wird der Abstand zum Fokuspunkt definiert. Dieser ist bis kurz vor Schluss konstant. Gegen Ende des Films bleibt der Fokuspunkt in der Mitte der Szene, wodurch der Abstand der Kamera zum Fokuspunkt exponentiell wächst (Zeilen 102 und 110).

```
1  /*************************************************************
2   * Persistence Of Vision Ray Tracer Scene Description File *
3   * ----------------------------------------------------------- *
4   * Version with stereoscopic camera, POV-Ray 3.7             *
5   * Final_Frame=2000 Output_File_Type=N  Quality=11           *
6   * ANIMATE WITH CLOCK FROM 0 TO 1                            *
7   *                                                           *
8   * File: blood_x.pov                                         *
9   * Author: Tibor Gyalog                                      *
10  *                                                           *
11  * Red Blood Cells                                           *
12  *************************************************************/
13
14  // ==== Standard POV-Ray Includes ====
15  #include "colors.inc"    // standard color definitions
16  #include "textures.inc"   // standard texture definitions
17  #include "functions.inc"  // internal functions
18
19  // create a smooth blobby shape
20  #declare R1 = seed(0);
```

```
21
22 #declare RadiusVal   = 1.3; // (0 < RadiusVal)
23 #declare StrengthVal = 4; // (+ or -) radiating density
24 #declare RR=5;
25 #declare bloodcell=object{
26   merge{
27     cylinder {
28       <0,-RadiusVal/4,0>,  <0,RadiusVal/4,0>,  2*RR
29     }
30
31     blob {
32       // (0.0 < threshold <= StrengthVal) falloff threshold
33       threshold 0.6
34
35       #declare num = 0;
36       #while (num < 360)
37         #declare anglem=num*6.28/360;
38         sphere { < RR*cos(anglem),
39                    sin(6.28*clock)*0.3*cos(2*anglem+clock*6.28),
40              RR*sin(anglem)>, RadiusVal, StrengthVal }
41         #declare num = num+1;   // increment our counter
42       #end
43       scale 2
44     }
45
46     texture{
47       pigment{color rgbt <240/255,90/255,90/255,0>}
48       // control an object's surface finish
49       finish {
50         ambient 0.55          // ambient surface reflection color
51         // (---diffuse lighting---)
52         diffuse 0.6           // amount [0.6]
53         brilliance 1.0        // tightness of diffuse illumination
54         // (---phong highlight---)
55         phong 0.5             // amount [0.0]
56         phong_size 10         // (1.0..250+) (dull->highly polished)
57       } // finish
58     }
59   }
60 }
61
62 // Create 10 balls along X axis, from 0 to 9
63 #declare CellCount = 0;
64 #while (CellCount < 10)
65  object{bloodcell
66    scale 0.4
67    translate <(rand(R1)-0.5)*50,(rand(R1)-0.5)*50,50*(rand(R1)-0.5)>
68    rotate <180*rand(R1),180*rand(R1)+clock*360,180*rand(R1)>
69  }
70   #declare CellCount = CellCount+1;   // increment our counter
71 #end
```

```
72
73 object{bloodcell
74   scale 0.4
75   rotate <0,clock*360,0>
76   #if(clock >= 0.1)
77   rotate <-(clock-0.1)/.9*180,0,0>
78   #end
79 }
80
81 // set a color of the background (sky)
82 background { color NeonBlue }
83
84 // general light definition
85 light_source {
86   <10, 10, -80>    // position of the light source
87   color rgb 1.0    // color of the light
88   parallel
89 }
90
91 // at first backwards till clock = 0.3
92 // then dance of the blood cells till clock = 0.7
93 // at last exponential "zoom out" till clock = 1
94
95 #declare alph=20;
96 #if (clock < 0.1)
97   #local CAM_LOCATION = <0.0, 2, 10-clock*80*3>;
98 #else
99   #if (clock <0.8)
100    #local CAM_LOCATION = <0.0, 2, 10-0.3*80>;
101  #else
102    #local CAM_LOCATION = <0.0, 2, 10-0.3*80*exp(alph*(clock-0.8))>;
103  #end
104 #end
105
106 #local EYE = -1;   // 1 for right eye, -1 for left eye
107 #if(clock < 0.8)
108   #local FL = vlength(<0.0, 2, 10-0.3*80>); // the focal length
109 #else
110   #local FL = vlength(CAM_LOCATION);
111 #end
112 #local EYESEP = 0.47;  // eye separation
113
114 camera {
115   stereoscopic
116   right x*image_width/image_height
117   location CAM_LOCATION
118   look_at   <0.0, 0.0,  100>
119   angle 90
120   zeroparallax FL
121   eyeoffset EYE * EYESEP / 2
122 }
```

F.3 Der Menger-Schwamm

Das folgende POV-Ray Programm definiert den Menger-Schwamm sowie eine Kamerafahrt ins Innere des Fraktals. Die Kamerabahn wird auf den Zeilen 52 - 70 definiert.

```
1  /*************************************************************
2   * Persistence Of Vision Ray Tracer Scene Description File *
3   * ----------------------------------------------------------- *
4   * Version with stereoscopic camera, POV-Ray 3.7             *
5   * Final_Frame=1151 Output_File_Type=N  Quality=11           *
6   * ANIMATE WITH CLOCK FROM 0 TO 1                            *
7   *                                                           *
8   * File: schwamm_x.pov                                       *
9   * Author: Beat Trachsler                                    *
10  *                                                           *
11  * Menger-Schwamm                                            *
12  *************************************************************/
13
14 #include "stdinc.inc"
15 #include "skies.inc"
16 #include "stars.inc"
17 #include "textures.inc"
18 #include "stones.inc"
19 #include "rad_def.inc"
20 #include "metals.inc"
21 #include "stereocam.inc"
22 #include "hh_ani.inc"
23
24 global_settings {
25   max_trace_level 15
26 }
27
28 #declare macro_ani_1 = 0;
29 #declare macro_ani_2 = 0;
30
31 //Generelle Animationsdaten
32 #declare Frames         = 1151;  //Anzahl Einzelbilder
33 #declare FramesPerSec   = 25;    //Bildrate (Bilder pro Sekunde)
34 #declare TotalTime = (Frames-1)/FramesPerSec; //Gesamtdauer
35
36 //Zeitmarken Einheit: Sekunden / TotalTime
37 #declare T0  =  0.00/TotalTime; //Start, Titel-Sequenz
38 #declare T1  =  2.00/TotalTime; //Bewegung starten
39 #declare T2  = 10.00/TotalTime;
40 #declare T3  = 14.00/TotalTime; //Eintritt in den Schwamm
41 #declare T4  = 20.00/TotalTime;
42 #declare T5  = 24.00/TotalTime;
43 #declare T6  = 28.00/TotalTime;
44 #declare T7  = 32.00/TotalTime;
45 #declare T8  = 36.00/TotalTime;
46 #declare T9  = 40.00/TotalTime;
47 #declare T10 = 44.00/TotalTime;
```

```
48 #declare T11 = 46.00/TotalTime; //Ende
49
50 #declare Iteration_Depth = 4;
51 #declare D = 0.0001;
52 #declare Spline_1 =
53   spline {
54   natural_spline
55   T0, < 12,4,4>,            //Steuerpunkt
56
57   T1,  < 7,4,2>,            //Startpunkt
58   T2,  < 2.667,0,0>,
59   T3,  < 1,0,0>,
60   T4,  < 0.25,0,0>,
61   T5,  < 0,0.111,0>,
62   T6,  < 0,0.556, 0>,
63   T7,  < 0,0.667, -0.25>,
64   T8,  < 0,0.667, -0.556>,
65   T9,  < 0,0.778, -0.667>,
66   T10, < 0,0.889, -0.778>,
67   T11, < 0,0.889, -1.000>, //Endpunkt
68
69   1.20, < 0,0.889, -2>      //Steuerpunkt
70   }
71
72 hh_ani_1( T1, 1, clock, T1, T11, 1)
73 #declare Ks1 = macro_ani_1;
74
75 #if(clock >= T1)
76   #declare CAMERA_POSITION = Spline_1(Ks1);
77 #else
78   #declare CAMERA_POSITION = Spline_1(T1);
79 #end
80
81 #if(clock > T2)
82   #declare LOOK_AT = Spline_1(Ks1+.1);
83 #else
84   #declare LOOK_AT = -CAMERA_POSITION/4;
85 #end
86 #declare EYE = -1;  // -1 fuer das linke Auge, 1 fuer das rechte
87 #declare FL = vlength(LOOK_AT-CAMERA_POSITION);
88 hh_ani_1( FL/30, 0.002, clock, T1, T2, 3)
89 #declare EYESEP = macro_ani_1;
90
91 #if(clock < T4)
92   #declare SKY = y;
93 #elseif(clock<T5)
94   hh_ani_1( 0, 1, clock, T4, T5, 2)
95   #declare SKY = (1-macro_ani_1)*y + macro_ani_1*x;
96 #elseif(clock<T6)
97   hh_ani_1( 0, 1, clock, T5, T6, 2)
98   #declare SKY = (1-macro_ani_1)*x + macro_ani_1*z;
```

```
 99 #elseif(clock<T8)
100   hh_ani_1( 0, 1, clock, T6, T7, 2)
101   #declare SKY = (1-macro_ani_1)*z + macro_ani_1*y;
102 #else
103   hh_ani_2( 0, 1, 0, clock, T8, T9, 2, T9, T10, 2)
104   #declare SKY = (1-macro_ani_2)*y + macro_ani_2*z;
105 #end
106
107
108 // Stereoskopische Kamera
109 camera {
110   perspective
111   setSimpleStereoCam(SKY, LOOK_AT, CAMERA_POSITION, 45,
112     EYE * EYESEP / 2, FL)
113 }
114
115 // Sonnenlicht
116 light_source {
117   <7,4,2> //light position
118   color rgb <1,1,1>*1.6
119   parallel
120   point_at <0,0,0>
121 }
122
123 // Lichtquelle bei der Kamera
124 light_source {
125   CAMERA_POSITION
126   color rgb <1,1,1>
127 }
128
129 // Hintergrund
130 sky_sphere
131 {
132   pigment {
133     color NeonBlue
134   }
135 }
136
137 #macro menger_schwamm(p1,p2,tiefe)
138   #if(tiefe > 0)
139     #local length = p2.x - p1.x;
140     #local height = p2.y - p1.y;
141     #local depth  = p2.z - p1.z;
142       box{
143         <p1.x+1/3*length,p1.y+1/3*height,p1.z-D>,
144         <p1.x+2/3*length,p1.y+2/3*height,p2.z+D>
145       }
146       box{
147         <p1.x-D,p1.y+1/3*height,p1.z+1/3*depth>,
148         <p2.x+D,p1.y+2/3*height,p1.z+2/3*depth>
149       }
```

```
150         box{
151           <p1.x+1/3*length,p1.y-D,p1.z+1/3*depth>,
152           <p1.x+2/3*length,p2.y+D,p1.z+2/3*depth>
153         }
154         #local ii = 0;
155         #while(ii < 3)
156           #local jj = 0;
157           #while(jj < 3)
158             #local kk = 0;
159             #while(kk < 3)
160               #if(!((ii=1 & jj=1) | (ii=1 & kk=1) | (jj=1 & kk=1)))
161                 menger_schwamm(p1+<ii*length/3,jj*height/3,kk*depth/3>,
162                   p1+<ii*length/3,jj*height/3,kk*depth/3>+(p2-p1)/3,
163                   tiefe-1)
164               #end
165               #local kk = kk + 1;
166             #end
167             #local jj = jj + 1;
168           #end
169           #local ii = ii + 1;
170         #end
171     #end
172 #end
173
174 object{
175   difference{
176     box{ <-1,-1,-1>, <1,1,1>}
177     menger_schwamm(<-1,-1,-1>, <1,1,1>, Iteration_Depth)
178   }
179   texture {
180       T_Grnt6
181   }
182 }
```

F.4 Programmbeispiele aus dem DNA-Film

DNA-Film, 1. Szene Das folgende POV-Ray Programm visualisiert die erste Szene mit der Kamerafahrt um die Doppelhelix und ins Innere derselben.

```
1  /*************************************************************
2   * Persistence Of Vision Ray Tracer Scene Description File *
3   * ----------------------------------------------------------- *
4   * Version mit stereoskopischer Kamera, POV-Ray 3.7          *
5   * Final_Frame=1126 Output_File_Type=N  Quality=11           *
6   * ANIMATE WITH CLOCK FROM 0 TO 1                            *
7   *                                                           *
8   * File: dna_model_v_3_x.pov                                 *
9   * Author: beat.trachsler@kzo.ch                             *
10  * www.beat-trachsler.ch                                     *
11  *                                                           *
12  * DNA Visualisierung und Kameraflug durch die DNA           *
13  *************************************************************/
```

```
14
15 #include "colors.inc"
16 #include "finish.inc"
17 #include "molekuele3.inc"
18 #include "hh_ani.inc"
19
20 global_settings {
21   ambient_light rgb<1, 1, 1>
22 }
23
24 #declare macro_ani_1 = 0;
25 #declare macro_ani_2 = 0;
26
27 //Generelle Animationsdaten
28 #declare Frames      = 1126; //Anzahl Einzelbilder
29 #declare FramesPerSec    = 25; //Bildrate
30 #declare TotalTime = (Frames-1)/FramesPerSec; //Gesamtdauer
31
32 //Zeitmarken Einheit: Sekunden / TotalTime
33 #declare T0 = 0.00/TotalTime;  //Start, Alles dunkel
34 #declare T1 = 0.75/TotalTime;  //Licht hochfahren
35 #declare T2 = 2.25/TotalTime;  //Kamerafahrt um die DNA beginnen
36 #declare T3 = 15.75/TotalTime; //Beschriftung einblenden
37 #declare T4 = 17.25/TotalTime; //Beschriftung anzeigen
38 #declare T5 = 18.00/TotalTime; //Beschriftung ausblenden
39 #declare T6 = 19.50/TotalTime; //Kamera ueber die DNA hochfahren
40 #declare T7 = 24.00/TotalTime; //Flug durch die DNA beginnen
41 #declare T8 = 43.50/TotalTime; //Licht hinunterfahren
42 #declare T9 = 45.00/TotalTime; //Ende
43
44 hh_ani_2( 0, 1, 0, clock, T1, T2, 1, T8, T9, 1 )
45 #declare Lq = macro_ani_2;
46
47 hh_ani_1( 0, 1, clock, T2, T3, 2)
48 #declare Kr1 = macro_ani_1;
49
50 hh_ani_1( 0, 1, clock, T2, T3, 2)
51 #declare Lv1 = macro_ani_1;
52
53 hh_ani_1( 0, 1, clock, T6, T7, 2)
54 #declare Kr2 = macro_ani_1;
55
56 hh_ani_1( 0, 1, clock, T7, T9, 3)
57 #declare Kv= macro_ani_1;
58
59 hh_ani_1( 0, 1, clock, T7, T9, 2)
60 #declare Kr3 = macro_ani_1;
61
62 #declare CAMERA_POSITION =  <0,-30,-45>;
63
64 #declare EYE = -1; // -1 fuer das linke Auge, 1 fuer das rechte
```

```
65 #declare FL = 45;  // Abstand zum Fokuspunkt
66 #declare EYESEP = (1-Kr2)*FL / 30+Kr2*.05;  // Augenabstand
67
68 camera {
69   stereoscopic
70   location  CAMERA_POSITION
71   right x*image_width/image_height
72   look_at <0,-30,0>
73   sky <0.2*(1-Kr2),1,0>
74   zeroparallax FL
75   eyeoffset EYE * EYESEP / 2
76
77   rotate Kr1*360*y
78   translate Lv1*30*y
79   rotate Kr2*90*x
80   rotate Kr3*720*y
81   translate <0,-64*Kv,0>
82 }
83
84 light_source {
85   CAMERA_POSITION
86   color rgb <1,1,1> * Lq * Kr2
87   rotate Kr1*360*y
88   translate Lv1*30*y
89   rotate Kr2*90*x
90   rotate Kr3*720*y
91   translate <0,-64*Kv,0>
92 }
93
94 light_source {
95   CAMERA_POSITION
96   color rgb <1,1,1> * 1.2 * Lq * (1-Kr2)
97   spotlight
98   point_at <0,-30-80*Kv,0>
99   rotate -45*y * (1 - Kr2)
100  rotate Kr1*360*y
101  translate Lv1*30*y
102  rotate Kr2*90*x
103 }
104
105 sky_sphere
106 {
107   pigment {
108     color NewMidnightBlue
109   }
110 }
111
112 plane {
113   z, -45+.001
114   pigment { color rgbt <0,0,0, Lq>}
115   rotate Kr2*90*x
```

```
116    translate <0,-64*Kv,0>
117  }
118
119  #union{
120    object{
121      dna
122    }
123    object{
124      dna
125      translate <0,-34,0>
126    }
127    object{
128      dna
129      translate <0,-68,0>
130    }
131    object{
132      dna
133      translate <0,-102,0>
134    }
135    no_shadow
136  }
```

DNA-Film, Abspann Im Folgenden noch das Programm für den Abspann des DNA-Films:

```
1  /*************************************************************
2   * Persistence Of Vision Ray Tracer Scene Description File *
3   * ----------------------------------------------------------- *
4   * Version mit stereoskopischer Kamera, POV-Ray 3.7          *
5   * Final_Frame=601 Output_File_Type=N  Quality=11            *
6   * ANIMATE WITH CLOCK FROM 0 TO 1                            *
7   *                                                           *
8   * File: abspann_x.pov                                       *
9   * Author: beat.trachsler@kzo.ch                             *
10  * www.beat-trachsler.ch                                     *
11  *                                                           *
12  * DNA-Film: Abspann                                         *
13  *************************************************************/
14
15 #include "colors.inc"
16 #include "finish.inc"
17
18 #declare CAMERA_POSITION =  <0,0,-45>;
19
20 #declare EYE = -1;  // -1 fuer das linke Auge, 1 fuer das rechte
21 #declare FL = 45;   // Abstand zum Fokuspunkt
22 #declare EYESEP = 0.3; // Augenabstand
23
24 camera {
25   stereoscopic
26   location  CAMERA_POSITION
27   right x*image_width/image_height
```

```
28    look_at <0,0,0>
29    zeroparallax FL
30    eyeoffset EYE * EYESEP / 2
31  }
32
33
34  light_source {
35    CAMERA_POSITION
36    color rgb <1,1,1> * 1.2
37    spotlight
38    point_at <0,0,0>
39  }
40
41  sky_sphere
42  {
43    pigment {
44      color Black
45    }
46  }
47
48  #object{
49    cylinder{<0,0,0>, <0,1,0>, 1
50      texture {
51        pigment{
52          image_map {
53            png "Abspann.png" map_type 2 interpolate 4 once
54          }
55        }
56      }
57    }
58    translate <0,-0.5,0>
59    scale <15,50,15>
60    rotate y*(25-clock*240)
61    rotate 90*z
62    translate <0,0,-5>
63    no_shadow
64  }
```

G Auswertung der Umfrage zur horizontalen Parallaxe

Fragebogen: fitinIT

1. Welcher der drei Körper gefällt Ihnen am besten?
   Genauer: Bei welchem der drei Körper erscheint der 3D-Effekt am schönsten?
   [a] Der Körper am linken Rand
   [b] Der Körper in der Mitte der Szene
   [c] Der Körper am rechten Rand
2. Sehen Sie bei einem der beiden Körper ein Geisterbild?
   Geisterbild = Teile eines anderen Bildes, d.h. eine Art Aura..
   [A] Beim roten Körper.
   [B] Beim weissen Körper.
3. Geschlecht
   [a] männlich
   [b] weiblich
4. Alter
   [a] < 10
   [b] 10 - 14
   [c] 15 - 19
   [d] 20 - 29
   [e] 30 - 39
   [f] 40 - 49
   [g] 50 - 59
   [h] 60 - 69
   [i] >69

Grundauswertung Teil 1 der Befragung:
fitinit

1) Positive vs. negative Parallaxe

Der Körper am linken Rand	8	(15.09%)
Der Körper in der Mitte der Szene	30	(56.60%)
Der Körper am rechten Rand	15	(28.30%)
Summe	53	
ohne Antwort	0	

2) Auswirkung der Farbe auf die Disparität

Beim roten Körper.	5	(20.00%)
Beim weissen Körper.	20	(80.00%)
Nennungen (Mehrfachwahl möglich!)	25	
geantwortet haben	25	
ohne Antwort	28	

3) Geschlecht

männlich	26	(49.06%)
weiblich	27	(50.94%)
Summe	53	
ohne Antwort	0	

4) Alter

< 10	0	(0.00%)
10 - 14	2	(3.77%)
15 - 19	48	(90.57%)
20 - 29	1	(1.89%)
30 - 39	1	(1.89%)
40 - 49	1	(1.89%)
50 - 59	0	(0.00%)
60 - 69	0	(0.00%)
>69	0	(0.00%)
Summe	53	
ohne Antwort	0	

Literatur

[1] Brian J. Corcoran, Bradley Noyes, Steven Willis. Presentation tools for molecular visualization. Bachelor thesis, Worcester Polytechnic Institute, Dezember 2003.

[2] Paul Bourke. Creating correct stereo pairs from any raytracer, Februar 2001. URL http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/stereorender/.

[3] Paul Bourke. Additional cameras, mostly stereoscopic, for povray, Oktober 2007. URL http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/povcameras/.

[4] Wolfgang Wieser. Creating stereoscopic left-right image pairs with povray, April 2004. URL http://www.triplespark.net/render/stereo/create.html.

[5] Persistence of vision raytracer, version 3.7, 2010. URL http://www.povray.org/.

[6] Paul Bourke. Creating stereoscopic images that are easy on the eyes, Februar 2003. URL http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/stereorender/.

[7] F. Gonzalez and R. Perez. Neural mechanisms underlying stereoscopic vision. *Progress in Neurobiology*, 55(3):191 – 224, 1998.

[8] B. G. Cumming and G. C. DeAngelis. The physiology of stereopsis. *Annual Review in Neuroscience*, 24:203–238, 2001. URL http://hawk.med.uottawa.ca/public/reprints/ARN24_203.pdf.

[9] Frank Klawonn. *Grundkurs Computergrafik mit Java*. Vieweg + Teubner, 2005.

[10] Thomas Lehmann. *Handbuch der Medizinischen Informatik*. Hanser Fachbuchverlag, 2002.

[11] B. Julesz. Binocular depth perception of computer generated patterns. *Bell System Tech.*, 39:1125–1162, 1960.

[12] Paul Bourke. Calculating stereo pairs, Juli 1999. URL http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/stereorender/.

[13] Yei-Yu Yeh and Louis D. Silverstein. Limits of fusion and depth judgment in stereoscopic color displays. *Human Factors*, 32(1):45–60, 1990.

[14] Rhys Hawkins. Digital stereo video: display, compression and transmission. Master's thesis, The Australian National University, Februar 2002. URL http://escience.anu.edu.au/research/papers/02_Rhys_Hawkins/thesis-small.pdf.

[15] Marianne Aas. 3d video production. Master's thesis, Norwegian University of Science and Technology, August 2004.

[16] John M. Zelle, Charles Figura. Simple, lowcost stereographics: Vr for everyone. In *SIGCSE'04*, Norfolk, Virginia, USA, März 2004. ACM.

[17] Jan-Keno Janssen, Ulrike Kuhlmann. Augen frei. *c't*, (10):76–78, 2010.

[18] E. Willman, H. Baghsiahi, F.A. Fernández, D.R. Selviah, S.E. Day, V.C. Kishore, E. Erden, H. Urey, and P.A. Surman. The optics of an autostereoscopic multiview display. In *2010 SID International Symposium Digest of Technical Papers*. Society for Information Display, 2010. URL http://eprints.ucl.ac.uk/19139/.

[19] Qi Hong, Ruibo Lu, Thomas Xinzhang Wu, and Shin-tson Wu. Head mounted display with curved display screen, curved tunable focus liquid crystal micro-lens and first and second curved black masks corresponding independently to one of the right and the left eye. United States Patent 7667783, Februar 2010. URL http://www.freepatentsonline.com/7667783.html.

[20] Blu-ray Disc Founders. *White paper, Blu-ray Disc Format, General*, August 2004. URL http://www.blu-raydisc.com/en/Technical/TechnicalWhitePapers/General.html.

[21] HDMI Licensing, LLC. *High-Definition Multimedia Interface Specification Version 1.4a, Extraction of 3D Signaling Portion*. Hitachi, Ltd. Panasonic Corporation. Philips Consumer Electronics, International B.V. Silicon Image, Inc. Sony Corporation Technicolor, S.A. Toshiba Corporation, März 2010.

[22] x264 - a free h264/avc encoder, 2010. URL http://www.videolan.org/developers/x264.html.

[23] Xvid, 2010. URL http://www.xvid.org/.

[24] Telecommunication Standardization Sector of ITU (ITU-T). *H.264, Advanced video coding for generic audiovisual services*, März 2010. URL http://www.itu.int/rec/T-REC-H.264/en.

[25] webm, an open web media project, 2010. URL http://www.webmproject.org/.

[26] Sylvain Roques. How to make 3d-pictures by computer, Mai 2010. URL http://www.stereoscopy.com/faq/computer-stereo.html.

[27] Stereopov, stereoscopic raytracing with pov-ray, 2002. URL http://stereopov.ichthyostega.de/.

[28] Free blu ray burning software to burn blu ray disc, 2010. URL http://www.techmixer.com/free-blu-ray-burning-software-to-burn-blu-ray-disc/.

[29] Python imaging library, 2010. URL http://www.pythonware.com/products/pil/.

[30] 3d content: Getting started, 2010. URL http://www.google.com/support/youtube/bin/answer.py?hl=en&answer=157640.

[31] W3C. Html5, Oktober 2010. URL http://dev.w3.org/html5/spec/Overview.html.

[32] Mark Pilgrim. Dive into html5 - video on the web, 2010. URL http://diveintohtml5.org/video.html.

[33] Poseray, version 3.12.2, 2009. URL http://mysite.verizon.net/sfg0000/.

[34] Paul May. Dna, Januar 2000. URL http://www.chm.bris.ac.uk/motm/dna/dna.htm.

[35] 3d kinobrille, August 2010. URL http://www.3d-kinobrille.ch/.

[36] Virtualdub, 2010. URL http://www.virtualdub.org/.

[37] Ffmpeg, 2010. URL http://ffmpeg.org/.